Advanced Time Series Analysis and Optimization Techniques for Agricultural Applications

by

Qianqian Li

A dissertation

submitted to the Graduate School of Science and Engineering for Education

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Engineering

Supervisor: Prof. Zheng Tang

Associate Supervisor: Prof. Shangce Gao

University of Toyama

Gofuku 3190, Toyama-shi, Toyama 930-8555 Japan

2023

(Submitted November 27, 2023)

Abstract

Greenhouse farming always relies on sensors to monitor the dynamic climate conditions and generate time-related data. Accurate predictions of these time series are crucial for successful greenhouse cultivation. While many studies have focused on the chaotic characteristics of time series and developed various machine learning models, they often neglect the intrinsic features such as seasonality and tendency.

This study introduces a novel hybrid algorithm for efficiently solving complex optimization problems and a novel prediction model called SDN, which combines Seasonal-trend Decomposition as a preprocessing step and Single Dendrite Neuron as a nonlinear fitter to address greenhouse time series predictions.

The proposed hybrid algorithm, a Covariance Matrix Adaptation Evolution Strategy-based Manta Ray Foraging Optimization (CMAES-MRFO), is applyed to several benchmark problems and compare its performance with that of the individual techniques. And the decomposition provides SDN with the ability to process each component individually, and the well-designed neuron structure grants SDN time efficiency. Results from experiments indicate that the proposed SDN outperforms commonly used machine learning models and demonstrates robustness in the presence of custom parameters and outliers in datasets, increasing the likelihood of its practical application in greenhouse farming.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Evolutionary Algorithm

Evolutionary algorithms (EAs) are a class of computational intelligence approaches for tackling complex optimization problems [1]. These methods typically replicate biological evolution in a mathematical fashion. A prime example is ant colony optimization (ACO) [2], which draws inspiration from the collective behavior of ants. Simulated annealing (SA) [3] is based on the physical annealing process, while particle swarm optimization (PSO) [4, 5] is inspired by the hunting patterns of birds. The artificial bee colony (ABC) algorithm [6] borrows its concept from the foraging behavior of bee colonies.

EAs have become a popular approach for solving complex optimization problems in recent years. These methods are inspired by the process of biological evolution, which involves the survival of the fittest and the gradual improvement of species over time. EAs apply these principles to the optimization of complex functions, with the goal of finding the best possible solutions.

One of the advantages of EAs is their ability to handle a wide range of optimization problems [7], including those with complex or non-linear relationships. They can also be used to optimize functions with multiple objectives, which is a challenging task for traditional optimization methods. Moreover, EAs are typically very flexible and can be adapted to suit different problem domains.

EAs have been applied to a variety of real-world problems, such as scheduling [8], image processing [9], robotics [10], and finance [11]. Some popular examples of EAs include genetic algorithms, evolutionary strategies, particle swarm optimization, ant colony optimization, and artificial bee colony optimization which have been introduced in the paragraph above, each of these methods has its own strengths and disadvantage, and researchers continue to develop new and improved EAs for specific problem domains.

Overall, EAs have emerged as a powerful tool for solving complex optimization problems in a variety of domains. Their ability to handle non-linear and multi-objective functions makes them particularly useful for real-world applications. As research in this area continues to advance, it is likely that EAs will become even more widely used and effective in the years to come.

1.1.1 Genetic Algorithms

Genetic Algorithms (GA) belong to the class of Evolutionary Algorithms (EA) and are inspired by the principles of natural selection and genetics [12]. They are employed to solve optimization problems by generating a set of potential solutions and progressively refining them over successive generations.

Initially, the GA process creates a population of solutions, where each individual is represented as a string of binary digits or as a set of real values, known as the genotype. A fitness function is then used to evaluate the fitness of each solution in the population, which reflects its ability to solve the given problem.

Next, the fittest individuals are selected from the population and subjected to genetic operators, such as crossover and mutation, to create new offspring. Crossover involves swapping parts of the genotypes of two selected individuals to produce a new offspring. Mutation introduces random changes to the genotype of an individual, generating a new variant.

The new offspring replace some of the least fit individuals in the population, creating a new population of solutions. This iterative process is repeated over multiple generations until a satisfactory solution is found or a termination criterion is met.

GAs have been successfully employed in a variety of optimization problems [13], such as function optimization, feature selection, and parameter tuning in machine learning. The efficacy of GAs depends on the selection of genetic operators, population size, and other parameters, which can be tailored for specific problems.

1.1.2 Evolution Strategies

Evolution Strategies (ES) are a type of optimization algorithm that belongs to the broader family of Evolutionary Algorithms (EA). These algorithms are inspired by biological evolution but, instead of simulating natural selection, ES uses a self-adaptive approach to optimize a population of candidate solutions [14].

At the beginning of the search process [15], an initial population of candidate solutions is generated randomly. Each individual in the population is represented by a set of real-valued parameters, which are mutated in a self-adaptive manner to produce new offspring.

The offspring are evaluated using an objective function, and the best individuals are selected to form the next generation. The mutation operators in ES are designed to learn from the current population and adjust the mutation step sizes accordingly to improve search performance.

ES has been found to be highly effective in solving a wide range of optimization problems, including function optimization, neural network training, and control problems. They are especially useful for high-dimensional optimization problems, where gradient-based methods may not be suitable.

One of the main advantages [16] of ES is their robustness in noisy or stochastic environments, where the objective function may have noise or uncertainty. Additionally, ES can handle constraints and multi-objective optimization problems. However, the performance of ES is sensitive to the choice of mutation operators and population size, which can significantly affect the search quality and convergence speed.

1.1.3 Genetic Programming

Genetic Programming (GP) is a subfield of Evolutionary Computation that employs the concepts of natural selection and genetic operators to evolve computer programs that can solve a given problem [17]. The GP approach assumes that a computer program can be represented as a tree structure, where each node in the tree represents either an operation or a variable. The aim of GP is to evolve the structure and values of the program tree to solve a particular problem.

At the beginning of the GP process, an initial population of randomly generated programs is created, and each program is evaluated based on a fitness function that quantifies its performance in solving the problem. The programs with the highest fitness values are selected for reproduction, and genetic operators such as crossover and mutation are applied to generate new offspring. Crossover involves exchanging subtrees between two parent programs, while mutation entails randomly modifying a subtree in a single parent program.

The offspring are then evaluated, and the process is repeated over several generations until a satisfactory solution is found or a stopping criterion is met. One of the significant benefits of GP is its ability to generate diverse and complex solutions to a problem. GP has been successfully applied to various domains, including image and signal processing, classification, and robotics [18].

However, one of the challenges of GP is controlling the size and complexity of the evolved programs. This may lead to bloat, a phenomenon where the evolved programs become excessively large and complex without improving their fitness [19]. To address this problem, several techniques have been proposed, including fitness-based pruning and dynamic depth control.

1.1.4 Multi-Objective Optimization

Multi-Objective Optimization (MOO) is a widely studied optimization paradigm that deals with the optimization of multiple, often competing, objectives simultaneously. In practical real-world scenarios, including engineering design, finance, and resource allocation, several objectives must be considered for optimal decision making. MOO aims to find a set of solutions that represent a compromise between the different objectives [20].

Unlike traditional single-objective optimization problems, where there is a single optimal solution, MOO identifies a set of Pareto-optimal solutions that represent the best trade-offs between the objectives. A solution is Pareto-optimal if it cannot be improved in any one objective without sacrificing at least one of the other objectives. The set of Pareto-optimal solutions is referred to as the Pareto front.

A range of optimization techniques can be utilized for MOO, including Evolutionary Algorithms, gradient-based methods, and heuristic search. Common algorithms for MOO include Non-dominated Sorting Genetic Algorithm (NSGA), Multi-Objective Particle Swarm Optimization (MOPSO), and Differential Evolution (DE).

MOO has various practical applications [21], such as finance, where it can be used to optimize investment portfolios considering multiple objectives, such as maximizing returns and minimizing risk. In engineering design [22], MOO can be used to optimize products with competing objectives, such as maximizing strength and minimizing weight. However, MOO can be challenging, as the number of Pareto-optimal solutions can be vast, and identifying the most suitable solution from the Pareto front can be challenging.

1.2 Manta Ray Foraging Optimization

Manta Ray Foraging Optimization (MRFO) is a recently proposed metaheuristic algorithm that simulates the foraging behavior of manta rays in the ocean. This algorithm has been shown to be effective in solving various optimization problems, especially in high-dimensional search spaces.

The MRFO algorithm is inspired by the foraging behavior of manta rays, which is a type of filter-feeding behavior [23]. Manta rays are known to move through the water while filtering plankton from the water using their large gill plates. This behavior enables them to efficiently obtain food from a large area of the ocean.

The MRFO algorithm consists of three main components: the search space initialization, the search strategy, and the movement mechanism. The search space initialization is performed by randomly generating a set of potential solutions within the problem domain. The search strategy involves using the filter-feeding behavior of manta rays to guide the search process towards promising regions of the search space. Finally, the movement mechanism involves moving the search agents towards better solutions using a set of adaptive rules.

The effectiveness of the MRFO algorithm has been demonstrated in various optimization problems, such as feature selection, clustering, and function optimization. Compared to other metaheuristic algorithms, MRFO has shown competitive performance and better convergence characteristics, especially in high-dimensional search spaces.

1.3 Hybrid Algorithms

Hybrid algorithms have become increasingly popular in solving complex optimization problems due to their ability to balance exploration and exploitation. Evolutionary algorithms are often prone to the issue of getting stuck in local optima [24], which can be overcome by combining different algorithms. Hybrid algorithms combine two or more optimization techniques to leverage their strengths and overcome their limitations. The combination of different optimization techniques can create a more powerful approach for finding the global optimum of the problem [25, 26].

To further enhance the performance of EAs, researchers have proposed various hybrid algorithms that combine the strengths of different optimization techniques. For example, PSO-CS combines the iterative scheme of PSO with the search strategy of the cuckoo search (CS) [27], while CCWFSSE uses collaborative coevolution (CC) to process spherical evolution (SE) and wingsuit flying search (WFS) [28]. In addition, several other hybrid algorithms have been proposed in the literature, as summarized in [29–31]. These hybrid algorithms have shown promising results in addressing the local-optima-trapping issue, and have demonstrated improved performance in solving complex optimization problems.

The recent developments in hybrid algorithms have led to the creation of many effective optimization techniques that have been successfully applied to many real-world problems. The purpose of our study is to introduce a new hybrid algorithm that combines the advantages of two popular optimization techniques to solve a specific optimization problem.

1.4 Covariance Matrix Adaptive Evolution Strategy

The Covariance Matrix Adaptation Evolution Strategy (CMAES) is a renowned and powerful optimization algorithm that belongs to the Evolution Strategies (ES) family [32]. ES are a class of stochastic optimization algorithms that draw inspiration from the natural process of evolution, emulating the genetic mechanisms of reproduction, mutation, and selection. CMAES, in particular, is highly regarded for its ability to effectively optimize complex and high-dimensional problems.

One of the distinguishing features of CMAES is its adaptive covariance matrix update mechanism during the optimization process. This enables CMAES to dynamically capture the underlying structure of the search space and adjust its search strategy accordingly [33]. The adaptive update of the covariance matrix is based on statistical information gleaned from past search iterations, allowing CMAES to dynamically adapt its search distribution, step sizes, and mutation strengths, guiding the search towards promising regions of the search space.

CMAES has found extensive applications in a diverse range of optimization problems, spanning machine learning, computer vision, robotics, and engineering, owing to its effectiveness in discovering high-quality solutions with minimal function evaluations. Over the years, the algorithm has been continuously refined and extended with various variants, such as the active CMAES, the multimodal CMAES, and the surrogate-assisted CMAES, to further enhance its performance in specific problem domains.

1.5 Greenhouse Cultivation

The challenges faced in ensuring a sustainable agriculture and food supply for the future [34], due to factors such as climate change, a rapidly growing population, and decreasing fertile lands [35–37]. As a solution, greenhouse agriculture is presented as a feasible and sustainable option, which can help address these issues by providing a controlled environment for growing crops all year, even in adverse outdoor conditions.

The advancement of greenhouse agriculture has had a positive impact on global food supply, contributing to the reduction of hunger issues worldwide. This has also made it possible for consumers to purchase a variety of fruits and vegetables at local stores, even when they are not in season.

Greenhouses serve as a protective environment for plants. Greenhouses are enclosed structures that are covered with a material that allows light to pass through. The purpose of these structures [38] is to create a controlled environment inside, which enhances the growth of plants and leads to an increase in both the quality and quantity of crops produced. The controlled environment inside the greenhouse enables growers to improve crop productivity, regulate quality, cultivate crops that wouldn't thrive in outdoor conditions, and extend the growing season. This type of cultivation also provides greater crop protection and reduces the need for chemical use compared to traditional outdoor farming methods.

In traditional agriculture [39], farmers must regularly monitor environmental factors such as temperature, humidity, light intensity, and soil moisture in order to determine the best time, place, and conditions for growing crops.With greenhouse farming, crops are grown in controlled environments where the conditions can be optimized for specific plant types. Automation technology in greenhouses allows farmers to remotely monitor and adjust conditions, providing flexibility and convenience.

Artificial neural networks (ANNs) have been chosen for various studies to simulate, predict, optimize, and control processes, leading to an improvement in the greenhouse's overall yield. To optimize the ANNs database, mathematical analysis methods have been developed that establish relationships between variables, reduce the complexity of variables, and simplify the structure of the network.

1.6 Artificial Neural Networks

Back in 1943, Warren S. McCulloch & Walter Pitts propused a logical calculus mechanism [40] by character of nervous activity, which is widely recognized as a pioneering work that kickstarted the development of artificial intelligence and cognitive science. The paper introduced the concepts of logical neurons (specifically threshold neurons) and neural networks. The authors demonstrated that feedforward neural networks consisting of threshold neurons can represent any arbitrary Boolean function, making them universal approximators in the realm of Boolean functions. And in 1958, The questions being considered are twofold: firstly, in what manner is information stored or retained, and secondly, how does the information held in storage or memory impact the processes of recognition and behavior. Rosenblatt, F. [41] suggested a theoretical nervous system known as a perceptron, which is founded on certain physical parameters.The primary limitation of this neural network model is that it is confined to a sole layer of adaptable connections.

1.6.1 Back-propagation Algorithm

Although ANNs had been in existence for some time, their practical utility was not realized until the introduction of the back-propagation (BP) algorithm in 1986 [42].The process involves iteratively modifying the strengths of the connections within the network in order to reduce the discrepancy between the network's output and the target output. This continual adjustment of weights causes previously unrecognized internal units, known as hidden units, to acquire significance in the domain of the task at hand. By way of these hidden units, the patterns and regularities that underlie the task are captured by the network's interactions. It is the capability to generate advantageous novel characteristics that sets back-propagation apart from earlier, less sophisticated approaches such as the perceptron-convergence technique.

1.6.2 Applications of Artificial Neural Networks

During recent decades, the advancement of ANNs has been substantial, facilitating their prevalent utilization across various disciplines.There are several studies that provide additional evidence supporting the effectiveness, efficiency, and success of ANNs in addressing both complex and non-complex problems across various domains of life.ANNs offer a significant benefit in that they can simplify the modeling process and enhance the accuracy of complex natural systems with large input sets [43]. ANNs are a cutting-edge and valuable modeling tool that has been widely adopted in problem-solving and machine learning applications [44].

Modern information processing faces highly complex problems that can be difficult to solve using traditional methods. ANNs have the capability to imitate or even replace human thinking, enabling them to automatically diagnose and solve complex problems that are beyond traditional approaches. ANNs possess remarkable fault tolerance, robustness, and self-organizing abilities, allowing them to remain in optimal working condition even in the presence of highly damaged connections. This makes them highly suitable for electronic equipment applications in military systems.

ANNs have gained widespread adoption in control systems due to their unique model structure and inherent nonlinear simulation capabilities, coupled with their remarkable features of high adaptability and fault tolerance. Nonlinear adaptive learning mechanisms are integrated with various controller frame structures to enhance their performance. The basic control structures include supervision and control, direct inverse mode control, model reference control, internal model control, predictive control, and optimal decision control, among others.

To analyze changes in commodity prices, it is crucial to conduct a comprehensive examination of the various factors that affect the supply-demand relationship in the market. However, traditional statistical economics methods have their limitations, which makes it challenging to predict price changes scientifically. In contrast, ANNs can effectively handle incomplete, fuzzy, or uncertain data, making them a superior tool for price forecasting compared to traditional methods. By considering the complex and constantly evolving factors that influence market prices, ANNs can establish an accurate and reliable model. This model can then be used to predict changes in commodity prices and provide precise and objective evaluations.

1.7 Time Series Prediction

Time series that are collections of temporal observations, have garnered significant attention and spurred numerous studies and developments in the field of machine learning and artificial intelligence. These studies have encompassed a range of research aspects, from dimensionality reduction to data segmentation. However, one of the most critical subjects in this field is time series prediction, which involves forecasting future trends and tendencies.

To predict the value of a time series at a future time point $t + h$, the available observations from the time series at time t are typically used. The specific definition or mathematical formulation of this prediction task may vary depending on the context. To keep things simple, we will focus on sequences of single values in this discussion, but the techniques we consider can be easily applied to sequences of vectors as well. In theory, the value of $t + h$ could change continuously with time, such as in the case of temperature. In practice, x is usually measured at discrete points in time, with samples taken at regular intervals. The sampling rate determines the maximum level of detail in the resulting model, but higher resolution does not always translate to better predictions. In fact, using only every Nth point in the series can sometimes lead to superior results.

The changing climate conditions inside a greenhouse can be expressed as a collection of time series data denoted as $X = \{x_t \mid t \in \mathbb{N}^*\}$, where x_t represents the value of a specific climate parameter at a particular time point. The set of all these climate parameters within the greenhouse is represented by $C = X^p \mid p \in \mathbb{N}^*$. The climate parameters in the greenhouse are interrelated, such as the impact of light on temperature, humidity, and air humidity. Moreover, the growth of crops inside the greenhouse can also affect these parameters as they utilize energy from the environment. Each time series X^p in C is a complex feature with unpredictable irregularities and noise that are difficult to measure.

Recent studies have utilized both conventional statistical techniques, such as ARIMA [45], and advanced artificial neural networks (ANNs) [46] to predict greenhouse time series, but these methods still encounter challenges in terms of accuracy and efficiency. Thus, there is a requirement for more appropriate processing methods and intelligent models to overcome these issues.

1.7.1 Time Series Data

Time series data are often characterized by a high degree of complexity and unpredictability, with various factors affecting the data in unpredictable ways. ANNs have emerged as a popular choice for time series prediction tasks due to their strong nonlinear fitting capability, which arises from their multi-layered and node-connected structure [47]. For instance, a three-layer feedforward ANN has been used to predict COVID-19 deaths [48], and a five-layer ANN with a recurrent mechanism has been used to address various prediction tasks [49].

Among different types of ANNs, RNNs are particularly well-suited for time series data, as they can store representations of recent inputs and evolve into a ”long short-term memory” mechanism [50]. LSTM-based models, which are a type of RNN, have been used in various time series prediction tasks, including greenhouse time series. However, the use of complex ANNs can compromise efficiency and consume vast computational resources [51], especially for large and deep structures. To address these limitations, alternative machine learning models, such as SVMs and decision trees, have been explored for time series prediction tasks [52,53]. Nonetheless, ANNs remain a popular and effective tool for analyzing and predicting complex time series data.

1.7.2 Greenhouse Time Series

The study of greenhouse time series data is a critical aspect of the agriculture industry, as it helps greenhouse managers control and optimize the growth of crops. However, these time series data possess more subtle features than common time series, making them more challenging to analyze and predict accurately. Despite the advances in artificial neural network (ANN) technology, current research still focuses on the chaotic characteristics of these time series, leading to suboptimal solutions. The existing literature on greenhouse time series prediction has been characterized by the use of complex ANN architectures, which often compromise efficiency and consume vast computational resources.

For example, Buevich et al. proposed a stepwise autoregressive and recurrent network to predict greenhouse gas concentrations, but the model used 20 hidden layers without evaluating the computational costs [54]. Liu et al. used an LSTM-based model to capture short-term climate changes, but the training set included almost all data during a crop's growth cycle, which is not relevant for controlling midway dynamic climate parameters [46]. Cai et al. used an ensemble learning-based decision tree for greenhouse temperature prediction, but the model did not address the most typical features of this data [55].

Two critical features of the greenhouse time series are seasonality and tendency, which can be found in the irregularities and noises in the data. The periodic cycle of the greenhouse climate can be shorter than natural seasons [56–58], and different growing periods of crops can create a short-term tendency in dynamic climate parameters [59,60]. However, to the best of our knowledge, no existing work considers both features simultaneously. Therefore, there is a need for a more compact and less computationally expensive ANN that can effectively predict the greenhouse time series, considering both seasonality and tendency features.

1.8 Predictive Model

Data preprocessing is a crucial step in developing accurate predictive models. In the case of greenhouse time series prediction, it is especially important due to the complexity and subtlety of the data. The use of the STD method for preprocessing provides several advantages over other methods. For example, the ability to decompose any time series into three subseries allows for a more granular analysis of the data. The seasonal, trend, and residual components each contain unique information that can be used to improve the accuracy of the predictive model [61]. Additionally, the locally weighted regression used in the STD method enhances its robustness, making it well-suited for dealing with missing or invalid data.

The choice of ANN is also critical to the success of the predictive model. The DNM is a novel and promising approach that can provide enhanced accuracy while also being computationally efficient. By utilizing dendrites to model the input data, the DNM can capture complex patterns in the data without requiring as many hidden layers as other ANNs. This not only reduces computational costs but also increases interpretability of the model, making it easier to understand the underlying mechanisms driving the predictions.

Overall, the proposed SDN approach represents a significant advancement in greenhouse time series prediction. By combining the strengths of the STD method for preprocessing and the DNM as the ANN, the model can effectively handle the unique features of the data and provide accurate predictions. This has important implications for greenhouse management, as it can help optimize the performance of the greenhouse climate and improve crop yields.

Furthermore, the numerical nature of STD provides a significant advantage over other preprocessing methods that require additional mathematical modeling for each time series. This eliminates the need for extra time-consuming computations, making the process more efficient [62, 63]. Additionally, the ability to separately process each subseries means that the simpler methods can be applied to the seasonal and trend components, which in turn reduces the computational resources required for the entire process.

1.8.1 Dendritic Neuron Model

In order to fit the residual component, the proposed SDN model uses the Dendritic Neuron Model (DNM), a natural-inspired neural model that mimics the behavior of a single dendrite neuron. The unique structure of DNM is designed to ensure nonlinear modeling and error correction through a sigma-pi architecture [64,65]. Furthermore, its feedforward signal transfer method enables the reciprocation of used functions, and its four types of synapses' outputs accurately mimic the morphology of a single neuron, providing enhanced interpretability for the model [65].

The DNM has already been validated on various tasks such as classification, approximation, and prediction, and has made significant contributions to the machine learning research community [66–68]. What makes our proposed SDN unique is the modification made to the connecting function in the original DNM's dendrite layer, which further promotes the efficiency of the model.

To test the effectiveness of the proposed SDN model, we have used it on nine real-world datasets, including temperature, humidity, and CO_2 concentration data of cucumber, pepper, and tomato. By testing the SDN model on these diverse datasets, we are able to demonstrate its versatility and robustness in greenhouse time series prediction.

1.9 Outline

This study presents novel contributions to the field of greenhouse time series prediction.

1 A novel SDN model is proposed which follows the general trend of ”preprocessing-predicting” modeling pattern, but uses the most proper preprocessing method (i.e., seasonal-trend decomposition) that takes into account the intrinsic features of dynamic climate parameters in a greenhouse. This approach allows the SDN to separately predict the decomposed subseries, which results in improved accuracy and efficiency.

2 The nonlinear fitter used in SDN is not only a copy of dendritic neuron model, but is also modified in a novel way. The DNM is pruned to a single dendrite and the connecting function is exchanged in order to restrain the gradient disappearance when back-propagation is employed. Additionally, the mechanism of DNM has been researched further.

3 The SDN model can save significant computing resources when compared to deep learning-based models, thanks to its compact structure. Additionally, the SDN model generates more accurate results than other widely acknowledged intelligent models.

In summary, the proposed SDN model exhibits excellent time efficiency and predictive accuracy and thus offers a suitable solution to the problem of greenhouse time series prediction.

Chapter 2

Hybrid Algorithm CMAES-MRFO

In the study, we proposed a hybrid algorithm, CMAES-MRFO, that leverages the local search ability of CMAES to enhance the performance of MRFO in generating improved solutions. To evaluate the effectiveness of CMAES-MRFO, we employ the CEC'2017 benchmark functions, which are commonly used for benchmarking optimization algorithms. Experimental results demonstrate that CMAES-MRFO outperforms the original MRFO algorithm as well as other recently proposed algorithms. The proposed hybrid algorithm exhibits superior performance in terms of solution quality and convergence speed, showcasing the efficacy of integrating CMA-ES with MRFO for optimization tasks. The results of this study contribute to the advancement of evolutionary algorithms for solving complex optimization problems and highlight the potential of CMAES-MRFO as a competitive hybrid optimization algorithm.

2.1 Strategies of MRFO

The Manta Ray Foraging Optimization (MRFO) is an evolutionary algorithm that draws inspiration from the foraging behavior of manta rays in the ocean [23]. MRFO offers a novel approach for solving optimization problems. The algorithm leverages three specific foraging strategies observed in manta rays, namely chain foraging, cyclone foraging, and somersault foraging. These strategies are utilized to guide the search process towards regions of the search space with high potential and to prevent the algorithm from getting trapped in local optima. MRFO has demonstrated its effectiveness in solving various optimization problems, particularly those characterized by high-dimensional search spaces.

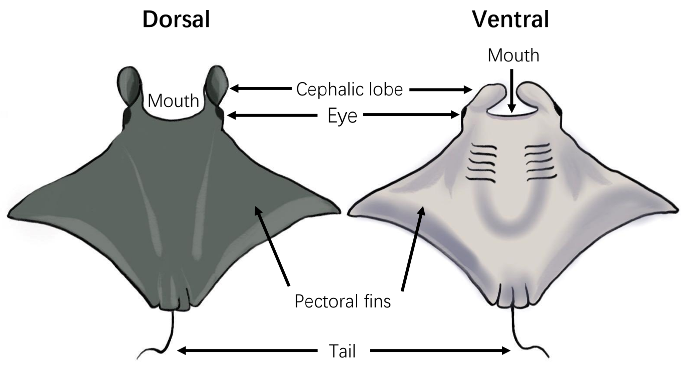

Figure 2.1: Manta Ray

2.1.1 Chain Foraging

The first foraging strategy exhibited by manta rays is a chain-shaped behavior. When a large number of manta rays engage in group hunting, they form a feeding chain with their heads and tails aligned in a linear fashion. The manta rays not only swim towards the food source but also follow the path of the preceding individual in the chain. This chain foraging strategy is characterized by the following behaviors:

$$x_i(t+1) = \begin{cases} x_i(t) + r_1 \cdot X_{best,i} + \alpha \cdot X_{best,i} & i = 1 \\ x_i(t) + r_1 \cdot X_{i-1,i} + \alpha \cdot X_{best,i} & 2 \leq i \leq N \end{cases} \tag{2.1}$$

where $i \in \mathbb{N}$, N is the population size, $X_{best,i} = x_{best} - x_i(t)$, $X_{i-1,i} = x_{i-1}(t) - x_i(t)$, $x_i(t)$ is the position of i-th individual at time t, x_{best} is the position of high food concentration, $\alpha = 2 \cdot r_1 \cdot \sqrt{|\log(r_1)|}$ is a weight coefficient, and r_1 is a random vector in the range of [0, 1].

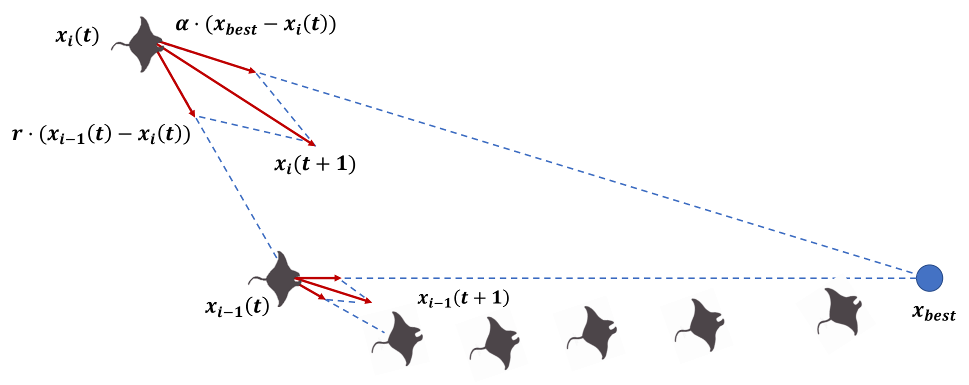
$x_i(t)$
$\alpha \cdot (x_{best} - x_i(t))$
$r \cdot (x_{i-1}(t) - x_i(t))$
$x_i(t+1)$
$x_{i-1}(t)$
$x_{i-1}(t+1)$
x_{best}

Figure 2.2: Chain Foraging

2.1.2 Cyclone Foraging

The second foraging strategy employed by manta rays is a cyclone-shaped behavior. When searching for plankton in deep waters, individual manta rays align their movements towards the plankton as their reference position, and approach the food source in a spiral manner. The cyclone foraging strategy involves assigning a new random position as a reference throughout the search space, which effectively reduces the likelihood of getting trapped in a local optimum. This strategy allows for increased exploration and enhances the algorithm's ability to escape local optima.

In order to determine the selection of current best or random position, the variable t/T is introduced. T is the maximum number of iterations. When $t/T \leq rand$, the cyclone foraging is formulated as:

$$x_i(t+1) = \begin{cases} x_i(t) + r_1 \cdot X_{rand,i} + \beta \cdot X_{rand,i} & i = 1, \\ x_i(t) + r_1 \cdot X_{i-1,i} + \beta \cdot X_{rand,i} & 2 \leq i \leq N \end{cases} \tag{2.2}$$

$$x_{rand} = x_{low} + rand \cdot (x_{up} - x_{low}) \tag{2.3}$$

where $X_{rand,i} = x_{rand} - x_i(t)$, β is the weight coefficient, x_{rand} is a random position within the search space, x_{up} and x_{low} are the upper and lower boundaries, respectively. Besides, when $t/T > rand$, it is formulated as:

$$x_i(t+1) = \begin{cases} x_i(t) + r_1 \cdot X_{best,i} + \beta \cdot X_{best,i} & i = 1, \\ x_i(t) + r_1 \cdot X_{i-1,i} + \beta \cdot X_{best,i} & 2 \le i \le N. \end{cases} \tag{2.4}$$

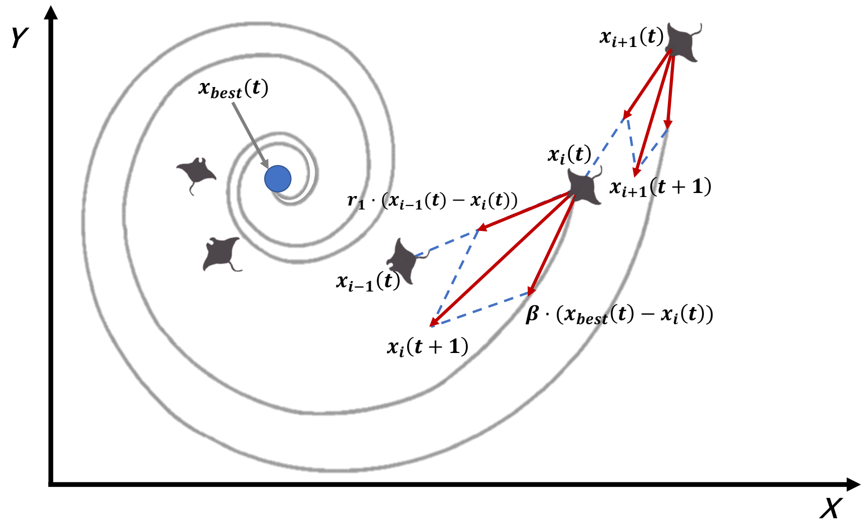

Figure 2.3: Cyclone Foraging

2.1.3 Somersault Foraging

The third foraging strategy exhibited by manta rays is known as somersault foraging, which is considered the most ornamental among their foraging behaviors. When manta rays locate a food source, they perform local periodic somersaults around the food, continuously updating their current position based on the best location found. This behavior allows them to fine-tune their position in relation to the food source, enabling them to converge towards the optimal solution in a dynamic and adaptive manner.

The somersault foraging is formulated as:

$$x_i(t+1) = x_i(t) + S \cdot (r_2 \cdot x_{best} - r_3 \cdot x_i(t)) \quad 1 \leq i \leq N \tag{2.5}$$

where S determines the somersault factor that indicates the somersault range of manta rays. We use $S = 2$ as suggested in [23]. r_2 and r_3 are two random numbers in the range of [0, 1].

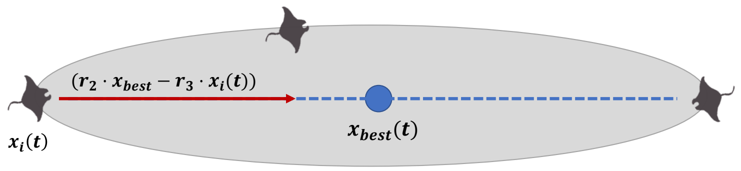

Figure 2.4: Somersault Foraging

2.2 Processing of CMAES

The mechanism of CMAES is a statistical-based adaptive evolving that generates the new individuals by random sampling of the constructed probability distributions [32]. Further, CMAES determines the search parameters (covariance matrix and global step size) in a statistical way, and the search parameters establish the direction and intensity of the new individuals' variation.

The CMAES process is as follows:

Step1: A multivariate normal distribution is utilized to generate a sampling search for the base point and to select the top λ individuals that perform better.

$$z_k^{(g+1)} = m_k^{(g)} + \sigma^{(g)} N(0, C^{(g)}), k = 1, \cdots, \lambda, \tag{2.6}$$

where $z_k^{(g+1)}$ is the k-th individual in the $(g+1)$-th generation, $m_k^{(g)}$ is the weighted mean of the selected individuals in the g-th generation, $\sigma^{(g)}$ is the g-th generation's global step size, $C^{(g)}$ is the covariance matrix of the g-th generation, and $\lambda \geq 2$ is the population size.

Step2: Update the weighted mean m.

$$m_k^{(g+1)} = m_k^{(g)} + \frac{1}{\lambda} \sum_{j=1}^{\lambda} \left(z_j^{(g+1)} - m_k^{(g)} \right) \tag{2.7}$$

where $z_j^{(g+1)}$ is the $(g+1)$-th generation's j-th individual among the top λ individuals.

Step3: Update the next generation of evolution route R_c and covariance matrix C as:

$$\begin{aligned} R_c^{(g+1)} =& (1 - o_c) R_c^{(g)} \\ &+ \frac{\sqrt{o_c(2 - o_c)\mu_{ef}}}{\sigma^{(g)}} (m^{(g+1)} - m^{(g)}) \end{aligned} \tag{2.8}$$

$$\begin{aligned} C^{(g+1)} =& (1 - o_1 - o_\mu) C^{(g)} + o_1 R_c^{(g+1)} (R_c^{(g+1)})^T \\ &+ o_\mu \sum_{j=1}^{\lambda} \omega_j y_j^{(g+1)} (y_j^{(g+1)})^T \end{aligned} \tag{2.9}$$

where o_c is a weight coefficient in the range of $[0, 1]$, μ_{ef} is the variance efficient, σ is the global step size, o_1 and o_μ are the covariance matrix learning rates in rank-1 and rank-μ, respectively. ω_j is the weight coefficient for individual z_j, and $y_j^{(g+1)} = (z_j^{(g+1)} - m^{(g)})/\sigma^{(g)}$ is variation step size.

Step4: Update the evolution route R_σ and global step size σ.

$$\begin{aligned} R_\sigma^{(g+1)} =& (1 - o_\sigma) R_\sigma^{(g)} + \frac{\sqrt{o_\sigma(2 - o_\sigma)\mu_{ef}}}{\sigma^{(g)}} \\ &\cdot (C^{(g)})^{-\frac{1}{2}} \cdot (m^{(g+1)} - m^{(g)}) \end{aligned} \tag{2.10}$$

$$\sigma^{(g+1)} = \sigma^{(g)} \cdot e^{\frac{o_\sigma \cdot ||R_\sigma^{(g+1)}||}{d_\sigma \cdot E||N(0,I)||} - \frac{o_\sigma}{d_\sigma}} \tag{2.11}$$

where o_σ is the learning rate of R_σ, d_σ is the damping parameter, and $E||N(0,I)||$ is the Euclidean parametric expectation of a random vector that follows $N(0,I)$.

Step5: Repeat steps 1 to 4 till the stop condition is met.

2.3 CMAES-MRFO

In CMAES-MRFO, the foraging strategy makes the individuals widely spread at the search space and effectively reduces the probability of getting stuck in a local optimum, while the CMAES guarantees the convergence once there are enough individuals locked in better zones. The mechanism of CMAES-MRFO is expressed as:

$$X_{MRFO} = \begin{cases} X_{chain}, & rand \geq 0.5 \\ X_{cyclone}, & rand < 0.5 \end{cases} \tag{2.12}$$

where X_{MRFO} is the output of MRFO, X_{chain} is the output of chain foraging, $X_{cyclone}$ is the output of the cyclone foraging. Equation (2.12) refers to the process where we perform a global search to obtain the best individual by calculating individuals' fitness. Then, the obtained best individual is imported to the local search operator CMAES for fast convergence. The hybridization with CMAES is formulated as:

$$Z_{CMAES} = X_{MRFO} + \sigma \cdot N(0,C). \tag{2.13}$$

Similar to Eq. (2.6), X_{MRFO} is imported into the local search operator CMAES, and it is considered as the first-generation weighted mean $m^{(0)}$ to generate the sampling space. Furthermore, the CMAES-MRFO is concluded in Algorithm 1.

Algorithm 1: Pseudo-Code of CMAES-MRFO

```
// Initialization
Initialize the population size $N$.
Initialize the maximum iterations $T$.
Calculate the fitness function of each individual and obtain $x_{best}$.
Set $t = 0$, $\sigma^{(0)} = 1.0$, and $C^{(0)} = eye(dimension)$.
// Main loop
while stopping criterion is not achieved do
    for each individual i do
        if $rand \geq 0.5$ then
            Chain foraging $X_{chain}$, using Eq. (2.1)
        else
            if $t/T \leq rand$ then
                Update the individual's next generation based on random positions, using
                  Eq. (2.2) and (2.3)
            else
                Update the individual's next generation based on current best positions,
                  using Eq. (2.4)
        Calculate the individual's fitness and update the best individual.
        Then $m^{(0)} = X_{MRFO}$
        for each individual j do
            Generate new sample individuals, using Eq. (2.6)
            Update $m$, using Eq. (2.7)
            Update $R_c$, $R_\sigma$, $C$, and $\sigma$, using Eq. (2.8), (2.9), (2.10), and (2.11)
    $t = t + 1$
return the individual with the best fitness
```

2.4 Experimental Results

The proposed CMAES-MRFO is tested on the CEC'2017 benchmark function suit. All the experiments are implemented under MATLAB R2022a with an AMD Ryzen 5 5600H chip. The population size is set as 100, and the dimension is set to 30. In addition, we run each function 30 times to avoid random errors. Then we calculate the mean and standard deviation of each function's result. To confirm the significance of CMAES-MRFO, we compare it to some newly-coming EAs including AHA and RSA as well as the original MRFO and CMAES. Table 2.1 summarizes the results of all algorithms and highlights the best ones. It is obviously that the proposed CMAES-MRFO performs the best.

Furthermore, the Wilcoxon test is a non-parametric test for hypothesis testing of two samples. Table 2.2 shows the results of the Wilcoxon test, with p-both referring to the general p-value. We define p-left as an indicator to determine whether the median error of CMAES-MRFO is smaller than comparison algorithms. The table shows that p-both and p-left are all less than the significance level (0.05), indicating that the CMAES-MRFO is significantly superior to all its competitors.

Fig. 2.5 and 2.6 show the convergence graphs of F9 and F23. We can see that the CMAES-MRFO has found a better solution than other algorithms. From Fig. 2.7 and 2.8, we can tell that the CMAES-MRFO outperforms both of its original ideas on F21 and F8. The box-and-whisker plots shown in Fig. 2.9 and 2.10 indicate that the CMAES-MRFO holds a strong robustness on F9 and F24.

The analyses show that the proposed CMAES-MRFO outperforms the other competitors in general. Therefore, we can presume that CMAES-MRFO achieves a good balance between exploration and exploitation. Nevertheless, in Table 2.1, 16 out of 29 standard deviations of CMAES-MRFO perform better, which is only a slight advantage compared to 25 out of 29 of the means. For example, F28 possesses the best mean, but the standard deviation is optimal for AHA. This suggests that the proposed CMAES-MRFO still needs further adjustment in terms of stability.

Table 2.1: Experimental results of all algorithms on CEC'2017

	CMA-ES-MRFO		**CMA-ES**		**MRFO**		**AHA**		**RSA**	
	Mean	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**
F1	**0.000E+00**	0.000E+00	**0.000E+00**	0.000E+00	3.404E+03	3.789E+03	3.804E+03	4.697E+03	4.920E+10	6.332E+09
F3	**2.774E-09**	1.520E-08	2.021E+02	2.951E+02	1.856E+00	2.109E+00	2.677E+01	4.543E+01	7.442E+04	6.475E+03
F4	**2.236E+00**	4.045E+00	6.022E+01	3.314E+01	3.495E+01	3.627E+01	8.081E+01	3.084E+01	8.194E+03	2.750E+03
F5	**6.746E+01**	2.273E+01	7.208E+02	2.265E+02	1.606E+02	3.208E+01	1.071E+02	2.353E+01	3.960E+02	3.327E+01
F6	**1.933E-01**	4.296E-01	1.001E+02	1.351E+01	1.505E+01	1.182E+01	2.464E-01	5.292E-01	8.023E+01	8.547E+00
F7	**9.756E+01**	2.777E+01	3.501E+03	9.327E+02	2.509E+02	6.657E+01	1.437E+02	3.558E+01	6.533E+02	2.699E+01
F8	**5.900E+01**	1.749E+01	6.210E+02	1.602E+02	1.400E+02	2.900E+01	1.051E+02	2.424E+01	3.143E+02	1.336E+01
F9	**6.766E+01**	6.553E+01	1.473E+04	3.117E+03	2.349E+03	1.121E+03	1.361E+03	9.709E+02	8.356E+03	8.182E+02
F10	**3.090E+03**	7.028E+02	4.945E+03	5.816E+02	3.704E+03	5.897E+02	3.245E+03	6.450E+02	6.808E+03	4.447E+02
F11	**6.416E+01**	3.411E+01	1.949E+02	7.034E+01	9.458E+01	3.308E+01	6.608E+01	2.801E+01	7.647E+03	2.541E+03
F12	**2.616E+03**	1.513E+03	2.888E+03	1.174E+03	4.903E+04	2.073E+04	1.152E+05	9.365E+04	1.331E+10	3.262E+09
F13	**3.242E+02**	1.771E+02	5.987E+03	1.979E+03	2.085E+04	1.945E+04	1.165E+04	1.027E+04	7.614E+09	3.637E+09
F14	**1.901E+02**	5.306E+01	2.963E+02	6.345E+01	3.904E+03	3.920E+03	1.243E+03	1.541E+03	4.495E+06	4.835E+06
F15	**1.697E+02**	1.302E+02	2.490E+03	2.456E+03	4.927E+03	5.779E+03	2.308E+03	2.436E+03	5.799E+08	3.789E+08
F16	6.559E+02	2.738E+02	**5.448E+02**	2.778E+02	9.233E+02	2.459E+02	9.237E+02	2.804E+02	3.883E+03	9.415E+02
F17	**2.127E+02**	1.503E+02	2.987E+02	2.163E+02	4.361E+02	1.766E+02	3.425E+02	1.865E+02	3.475E+03	2.780E+03
F18	**1.967E+03**	1.825E+03	6.283E+03	4.967E+03	1.210E+05	8.972E+04	2.115E+04	1.339E+04	1.832E+07	1.220E+07
F19	**1.359E+02**	8.473E+01	1.806E+03	4.429E+03	7.837E+03	1.019E+04	3.747E+03	3.045E+03	6.167E+08	3.899E+08
F20	**2.096E+02**	9.346E+01	1.450E+03	2.820E+02	4.418E+02	1.717E+02	3.408E+02	1.380E+02	8.773E+02	1.161E+02
F21	**2.644E+02**	2.945E+01	5.269E+02	2.970E+02	3.197E+02	3.809E+01	2.720E+02	1.529E+01	6.001E+02	5.285E+01
F22	**1.002E+02**	7.977E-01	6.114E+03	1.165E+03	1.002E+02	7.541E-01	1.006E+02	1.274E+00	5.873E+03	1.202E+03
F23	**4.260E+02**	3.064E+01	1.932E+03	7.031E+02	5.010E+02	4.352E+01	4.383E+02	2.768E+01	1.005E+03	6.338E+01
F24	**4.916E+02**	2.113E+01	6.026E+02	1.234E+02	5.762E+02	5.870E+01	5.076E+02	2.818E+01	1.035E+03	2.572E+02
F25	3.912E+02	1.530E+01	**3.868E+02**	5.429E-02	3.913E+02	1.147E+01	3.963E+02	1.523E+01	2.400E+03	6.125E+02
F26	1.808E+03	6.067E+02	1.092E+03	5.416E+02	2.226E+03	1.416E+03	**7.638E+02**	1.081E+03	7.804E+03	1.010E+03
F27	**5.423E+02**	1.366E+01	8.036E+02	1.169E+03	5.566E+02	2.132E+01	5.425E+02	1.504E+01	1.301E+03	5.640E+02
F28	**3.154E+02**	4.065E+01	3.431E+02	5.889E+01	3.344E+02	5.462E+01	4.043E+02	1.871E+01	3.571E+03	7.608E+02
F29	7.158E+02	1.606E+02	7.910E+02	1.795E+02	8.655E+02	2.499E+02	**7.038E+02**	1.582E+02	3.333E+03	1.321E+03
F30	**3.208E+03**	7.838E+02	1.263E+04	5.182E+03	5.779E+03	2.706E+03	4.779E+03	1.701E+03	2.397E+09	9.667E+08

Table 2.2: Results Obtained By The Wilcoxon Test

CMAES-MRFO	*p*-both	*p*-left
vs.CMAES	0.0019187	0.0009594
vs.MRFO	0.0102889	0.0051445
vs.AHA	0.0272245	0.0136122
vs.RSA	1.35E-07	6.751E-08

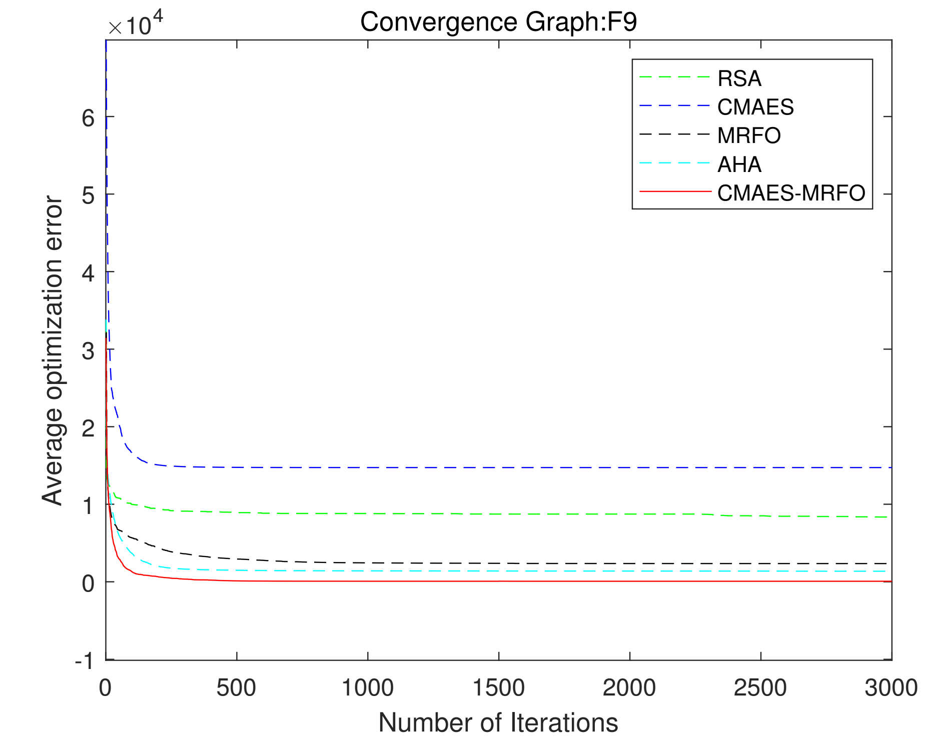

Figure 2.5: Convergence graph of all algorithms on F9

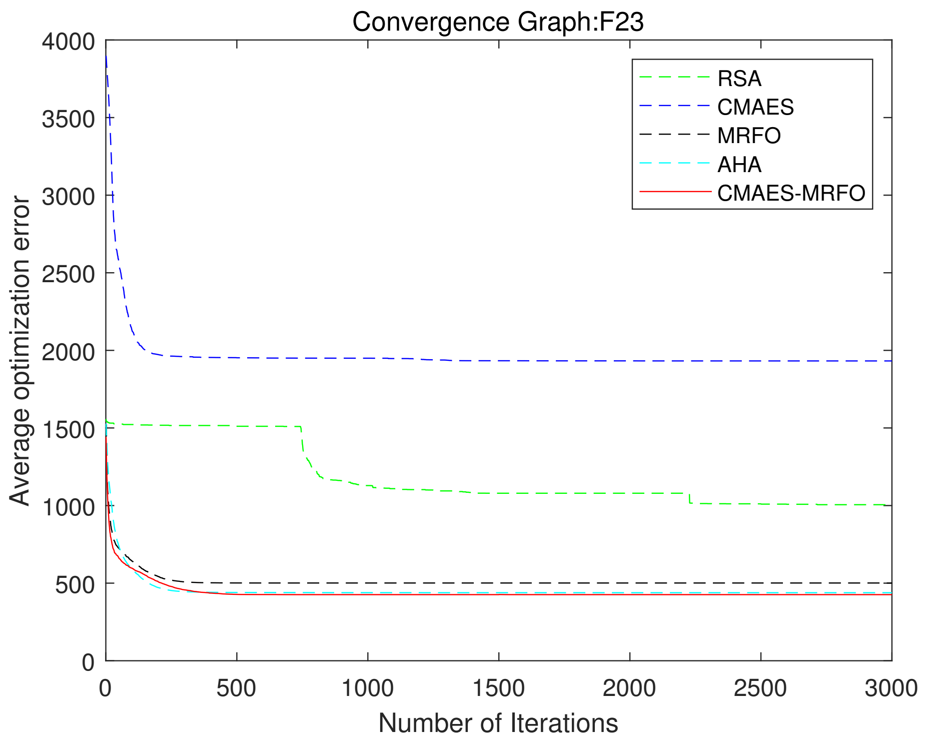

Figure 2.6: Convergence graph of all algorithms on F23

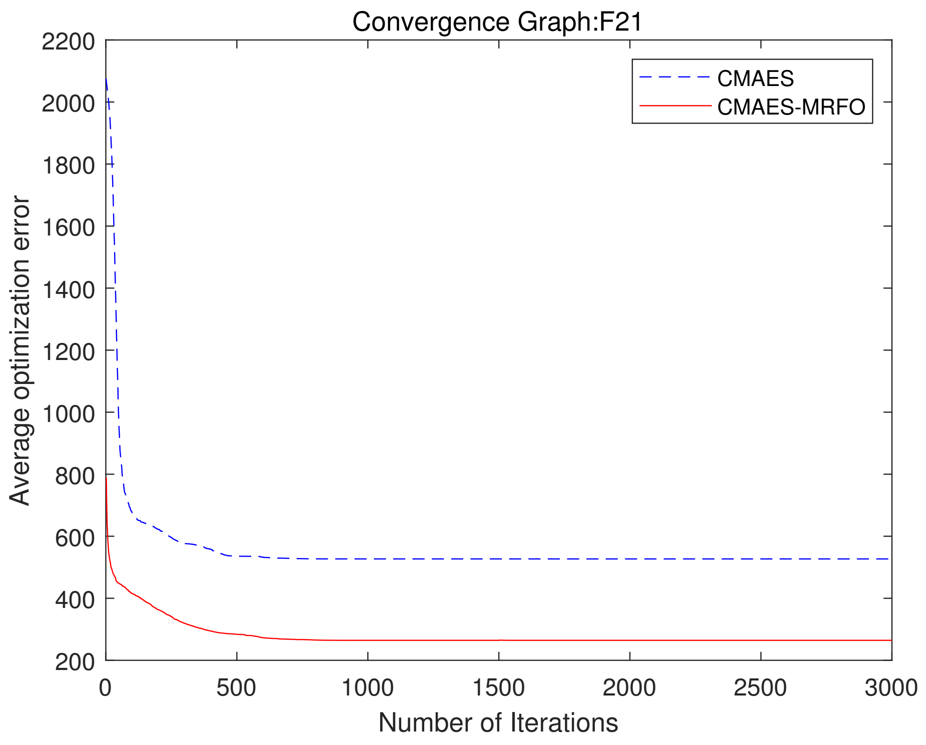

Figure 2.7: Convergence graph of CMAES-MRFO and CMAES on F21

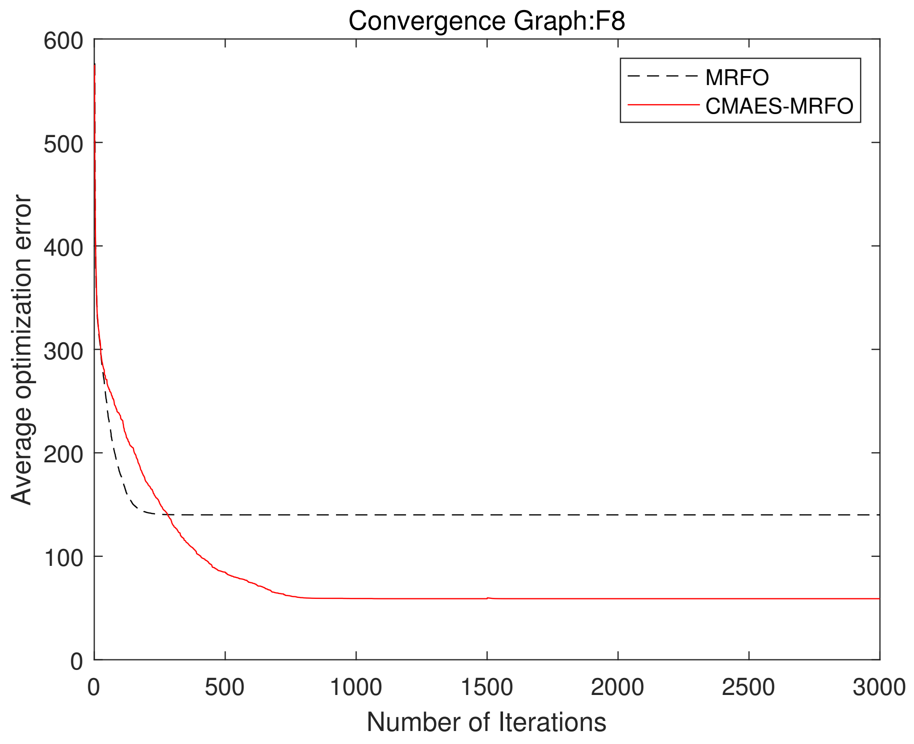

Figure 2.8: Convergence graph of CMAES-MRFO and MRFO on F8

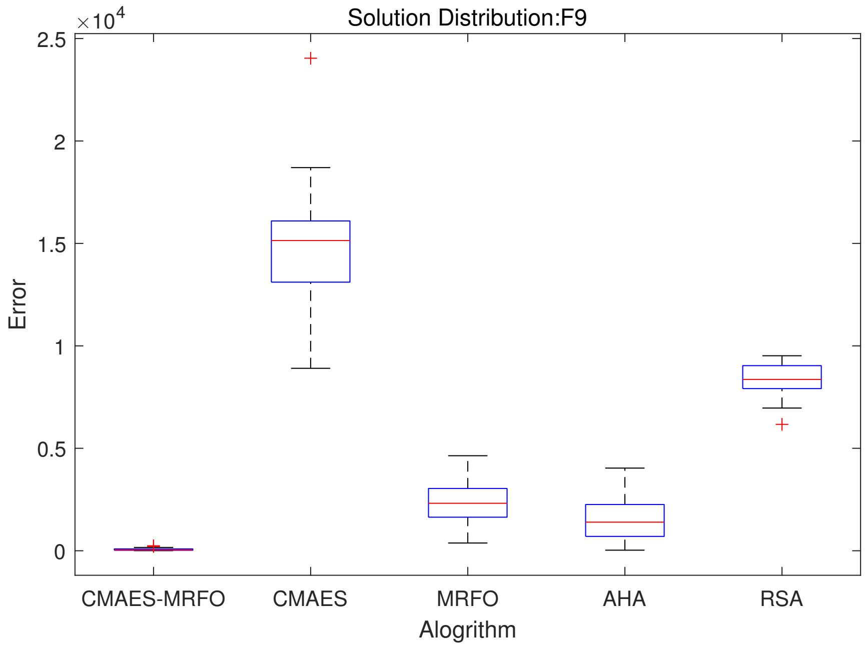

Figure 2.9: Box-and-whisker plot of all algorithms on F9

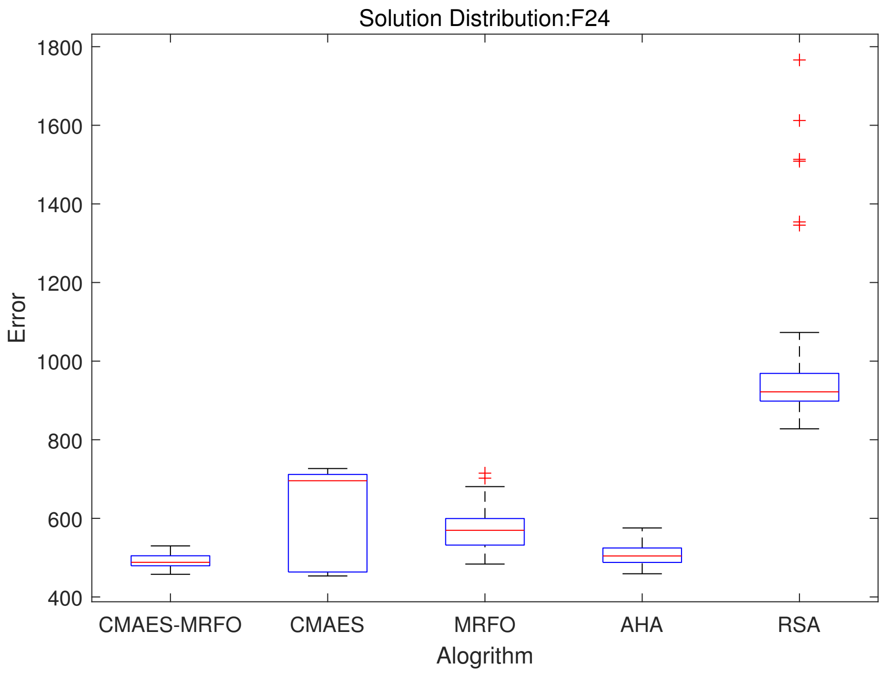

Figure 2.10: Box-and-whisker plot of all algorithms on F24

Chapter 3

SDN Predicting Model

The design of the SDN is based on the fundamental idea of optimizing the prediction process in greenhouse time series data analysis. The objective is to achieve the best possible accuracy and efficiency, while also minimizing the computational resources required for the task. To this end, the SDN is composed of two essential modules that work together to achieve this goal. The first module is the preprocessing method, which is designed to decompose the complex and dynamic climate parameters of the greenhouse into more manageable subseries. This allows the SDN to analyze and predict each subseries separately, resulting in significantly improved accuracy and efficiency.

The second module of the SDN is the nonlinear fitter, which is responsible for the actual prediction process. The nonlinear fitter is based on the dendritic neuron model, which is widely recognized as an effective and efficient method for modeling complex data. However, the SDN takes this a step further by modifying the dendritic neuron model in a unique and clever way. By pruning the DNM to a single dendrite and exchanging the connecting function, the SDN is able to prevent the gradient disappearance that is often encountered when using back-propagation in traditional neural networks. This modification also allows for further research into the underlying mechanism of the DNM.

In addition to its novel design, the SDN offers significant advantages over traditional deep learning-based models. Not only is the SDN much more compact and resource-efficient, but it also generates more accurate results than other widely acknowledged intelligent models. Therefore, the SDN is an ideal solution for greenhouse time series prediction, meeting the need for both high accuracy and time efficiency in this field.

3.1 Seasonal-tend Decomposition

The time series X from the greenhouse is decomposed into three subseries, namely seasonal, trend, and residual, by the preprocessing method STD. As a result of this decomposition, the original greenhouse time series X can be represented as

$$X = S + T + R \tag{3.1}$$

where these subseries exhibit the same pattern as X, where each time node t corresponds to a value. The cyclical subseries S can be directly used in its prediction, whereas subseries T, which only contains tendency information, can be predicted using simpler methods like polynomial fitting. The residual subseries R, which requires a significant amount of computing resources, is predicted using the modified single dendrite neuron. The SDN framework is illustrated in Fig.3.1, with green and blue shadows highlighting the key features of the model: 1) preprocessing enables the use of different calculation methods, and 2) the single dendrite neuron efficiently predicts the residual component.

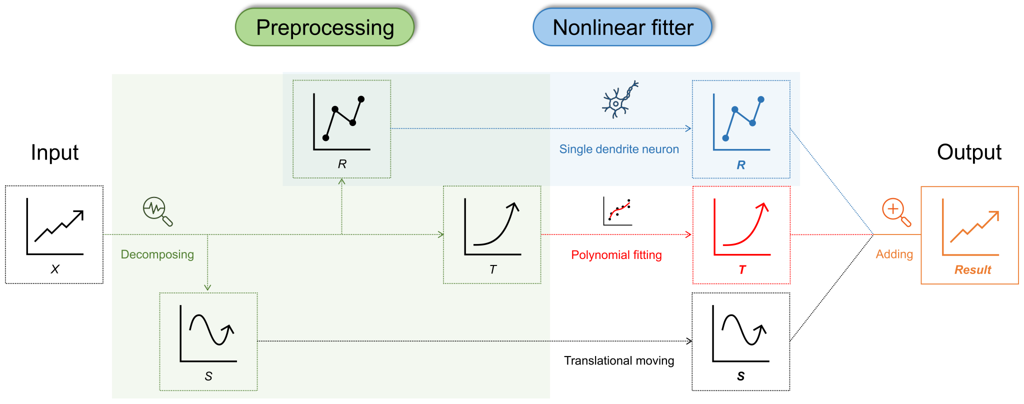

Figure 3.1: Framework of SDN

The primary procedure of the STD model involves the separation of the original greenhouse time series X into three distinct subseries, namely seasonal, trend, and residual, respectively. And this model do follows the general trend of 'preprocessing-predicting'.

3.1.1 Decomposition procedure

The STD algorithm is implemented through a loop-based updating mechanism, which utilizes locally weighted regression (loess) as a smoothing function. The updating loop comprises six essential steps, as listed below:

Step 1: *Detrending*. Compute the detrended series $X - T^{(k)}$ where k marks the k-th loop. Noted that if there are missing values in X's time nodes, the detrended series is also missing at those positions.

Step 2: *Cycle-subseries Smoothing*. Each cycle-subseries of the detrended series is smoothed by loess and the collection of all smoothed values is a temporary seasonal series, denote as $C^{(k+1)}$.

Step 3: *Low-Pass Filtering of Smoothed Cycle-subseries*. A low-pass filter is applied to $C^{(k+1)}$. The filter contains thrice moving average followed by a loess smoothing.

Step 4: *Detrending of Smoothed Cycle-subseries*. The seasonal component from the $(k+1)$-th loop is $S^{(k+1)} = C^{(k+1)} - L^{(k+1)}$ where $L^{(k+1)}$ is subtracted to prevent low-frequency power from entering the seasonal component.

Step 5: *Deseasonalizing*. Compute the deseasonalized series $X - S^{(k+1)}$. Noted that if there are missing values in X's time nodes, the deseasonalized series is also missing at those positions.

Step 6: *Trend Smoothing*. The deseasonalized series is also smoothed by loess, and the smoothed values are computed at all time nodes even those with missing values.

Fig. 3.2 presents a more concise representation of these six steps.

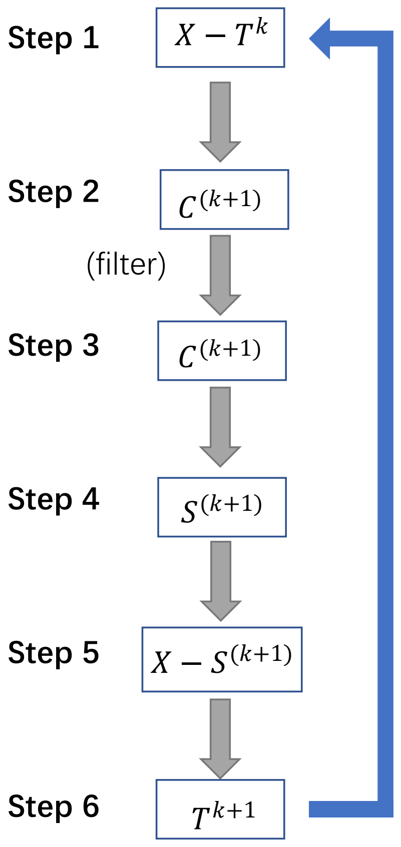

Figure 3.2: The updating loop comprises six essential steps

From step 2, 3, and 4 the seasonal component S is smoothed, and the trend component T is smoothed in step 6. Fig. 3.3 shows an example of a decomposed random series with 200 time nodes and period 40.

3.1.2 STD parameters

Noted that loess is widely used in smoothing the obtained seasonal and trend components, two parameters related to loess are denoted by:

- n_f: fraction of data to use in fitting loess regression (set to be 0.6).
- n_d: fractional distance within which to use linear-interpolation instead of weighted regression. A non-zero value of n_d significantly decreases the computation time (set to be 0.01).

In addition, the STD is certified to decompose any time series with any feasible period. Thus, the periodicity in time series, in units of one observation, is also an artificially determined parameter, namely,

- n_p: the most significant periodicity of the observed time series (set to be 1440 in this case study).

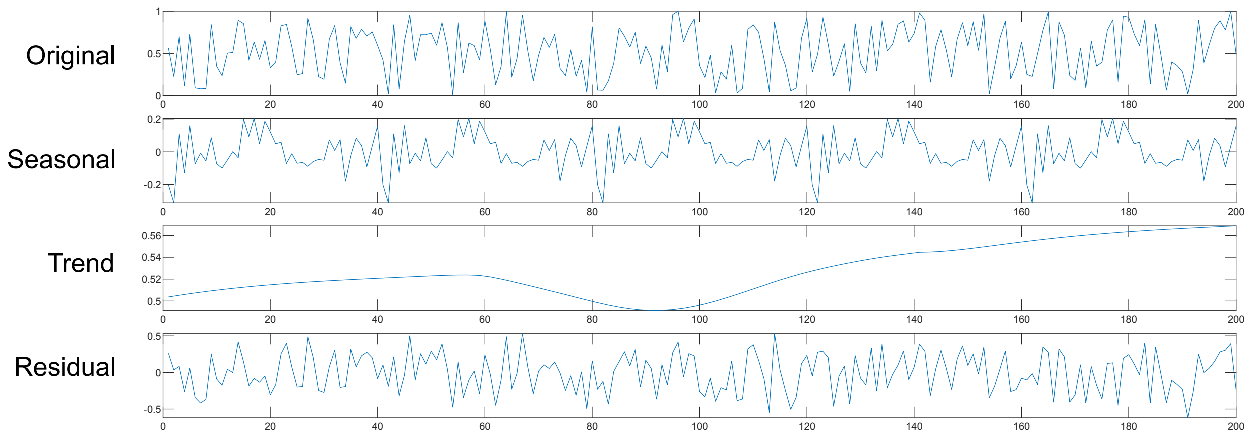

Figure 3.3: An example for STD.

3.2 Glance over dendritic neuron model

As shown in Fig. 3.4, the original dendritic neuron model has a four-layered structure which uses sigmoid function twice in synapse and soma layer, respectively. And the logical *AND* & *OR* operation are simulated by multiplication & summation in dendrite and membrane layer. Noting the input as x and assuming that there are N inputs and M dendrites, the whole procedure can be described as follows:

- Synapse layer: calculates the i-th synapse's output at j-th dendrite Y_{ij} as

$$Y_{ij} = \frac{1}{1 + e^{-\kappa(\omega_{ij} x_i - \theta_{ij})}}. \tag{3.2}$$

- Dendrite layer: multiplies each output Y_{ij} from N synapses on the same dendrite, denoted by

$$Z_j = \prod_{i=1}^{N} Y_{ij}. \tag{3.3}$$

- Membrane layer: sums up the output Z_j from M dendrites, denoted by

$$V = \sum_{j=1}^{M} Z_j. \tag{3.4}$$

- Soma layer: gives the final output O by another sigmoid function as

$$O = \frac{1}{1 + e^{-\kappa(V - \theta_s)}}. \tag{3.5}$$

Especially, the values of parameters ω_{ij} and θ_{ij} in Eq. (3.2) gives DNM the ability to perform four types of connections as shown in Fig. 3.5.

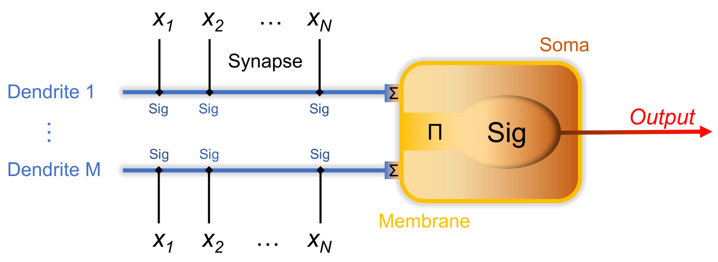

Figure 3.4: Structure of DNM.

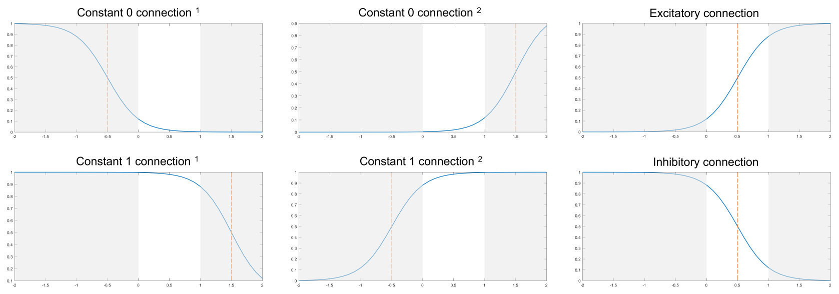

Figure 3.5: DNM's four types of connections.

3.2.1 Modified single dendrite neuron

Although the original DNM successfully mimics the morphology of a single neuron, the sigmoid function in synapse layer usually causes gradient disappearance referring to back-propagation's training. Besides, the decision of dendrites' number M seemingly influenced by the problem scale, but in practice many of the dendrites are deleted due to the multiplicative operation in dendrite layer.

In light of the issues mentioned above, the proposed nonlinear fitter in SDN places a greater emphasis on the technological perspective. To achieve this, we have employed a single dendrite neuron in our design, which has been modified to replace the synapse layer's sigmoid with a linear function. This ensures the viability of the backpropagation (BP) algorithm, which is essential for training the model.The use of a single dendrite neuron structure also provides added benefits in terms of reducing computational resources. This is achieved through the use of a simplified structure, which reduces the amount of information that needs to be processed at each layer.

To provide a better understanding of our proposed design, we have included a figure, as shown in Fig. 3.6. This figure illustrates the new design of our single dendrite neuron, and the connecting details of each layer are listed below. Our proposed nonlinear fitter in SDN addresses the aforementioned issues while also improving the efficiency and effectiveness of the model.

3.3 Synapse layer

The synapse layer receives the raw data and implement the first processing step. In the original DNM, the constant 0 connection will vanish all the information on this dendrite. Hence, after the modification, the synapse layer's connecting function is defined as

$$Y_j = \omega_j r_t + \theta_j \tag{3.6}$$

where Y_j is the j-th output, r_t is the normalized input from subseries R decomposed by STD, and ω_j & θ_j represent for the parameters that need to be trained. Here it should be noted that j remarks the position of inputs while t represents the time nodes in decomposed subseries R.

This kind of design allows more excitatory or inhibitory synapses because of the linear function, meanwhile, the activating level can be controlled by ω_j & θ_j. Therefore, a single dendrite can obtain the ability to store and process all the information from raw data and transfers them to the next layer.

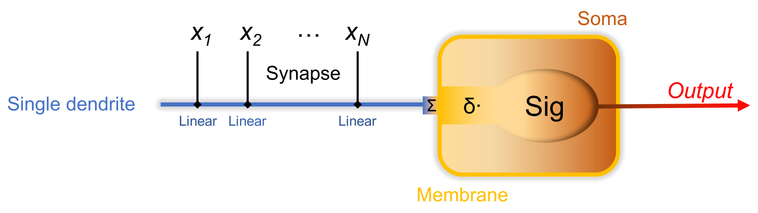

Figure 3.6: Single dendrite neuron in SDN.

3.3.1 Dendrite layer

The dendrite layer gathers the information from each synapse by using a multiplicative function. Assuming that there are N synapses, the dendrite layer's output is calculated by

$$Z = \prod_{j=1}^{N} Y_j \tag{3.7}$$

where multiplication is formally same as logic *AND* operation which indicate that all the features from synapses are used since the model only employs a single dendrite. It also benefits from the synapse layer's linear function that no constant 0 connections will affect the output of dendrite layer.

3.3.2 Membrane layer

The membrane layer processes the signal from dendrite layer by multiplying a factor δ as

$$V = \delta Z \tag{3.8}$$

where $\delta \in (0, 1]$ can be regarded as the passing rate of the membrane when ion exchange happens inside a cell. Technically, this design also makes the neuron more flexible than the original DNM.

3.3.3 Soma layer

The soma layer employs a sigmoid function as usual to determine the final output of the neuron, shown as

$$O = \frac{1}{1 + e^{-\kappa(V-\theta_s)}} \tag{3.9}$$

where κ is a positive constant and $\theta_s \in [0, 1]$.

This procedure also mimics the morphology of a neuron. When the coming signal V from the membrane layer exceeds its threshold θ_s, the soma body implements the excitatory conduction and the neuron is activated. Otherwise, the neuron shows an inhibitory action.

3.3.4 Back-propagation training

Due to the simple structure and easy-differentiable connecting functions, BP becomes the most efficient training algorithm for single dendrite neuron. In this supervised learning situation, the loss function is formulated as

$$E = \frac{1}{2}(O - \mathcal{T})^2 \tag{3.10}$$

where $\mathcal{T}$ represents the teaching signal and O is the output from single dendrite neuron. The error E is decreased by correcting ω_j and θ_j with learning rate η by a gradient descent way throughout every learning epoch, shown as

$$\Delta\omega_j(\epsilon_i) = -\eta \frac{\partial E}{\partial \omega_j(\epsilon_i)} \tag{3.11}$$

$$\omega_j(\epsilon_{i+1}) = \omega_j(\epsilon_i) + \Delta\omega_j(\epsilon_i) \tag{3.12}$$

$$\Delta\theta_j(\epsilon_i) = -\eta \frac{\partial E}{\partial \theta_j(\epsilon_i)} \tag{3.13}$$

$$\theta_j(\epsilon_{i+1}) = \theta_j(\epsilon_i) + \Delta\theta_j(\epsilon_i) \tag{3.14}$$

where ϵ_i denotes the i-th learning epoch. Besides, the differentials of E with respect to ω_j and θ_j are computed referring to the chain rule, shown as

$$\frac{\partial E}{\partial \omega_j} = \frac{\partial E}{\partial O} \cdot \frac{\partial O}{\partial V} \cdot \frac{\partial V}{\partial Z} \cdot \frac{\partial Z}{\partial Y_j} \cdot \frac{\partial Y_j}{\partial \omega_j} \tag{3.15}$$

$$\frac{\partial E}{\partial \theta_j} = \frac{\partial E}{\partial O} \cdot \frac{\partial O}{\partial V} \cdot \frac{\partial V}{\partial Z} \cdot \frac{\partial Z}{\partial Y_j} \cdot \frac{\partial Y_j}{\partial \theta_j} \tag{3.16}$$

where the partial differential components are calculated as

$$\frac{\partial E}{\partial O} = O - \mathcal{T} \tag{3.17}$$

$$\frac{\partial O}{\partial V} = \frac{\kappa e^{-\kappa(V-\theta_s)}}{(1 + e^{\kappa(V-\theta_s)})^2} \tag{3.18}$$

$$\frac{\partial V}{\partial Z} = \delta \tag{3.19}$$

$$\frac{\partial Z}{\partial Y_j} = \frac{\prod_{j=1}^{N} Y_j}{Y_j} \tag{3.20}$$

$$\frac{\partial Y_j}{\partial \omega_j} = r_{t_i} \tag{3.21}$$

$$\frac{\partial Y_j}{\partial \theta_i} = 1. \tag{3.22}$$

Noted that Y_j itself cannot be included in its differential when j is fixed as formulated in Eq. (3.20).

3.4 Proceeding of SDN

Due to the STD's preprocessing, the SDN allows a separate predicting procedure. Firstly, the seasonal component S is translated to the corresponding time node at prediction's starting point with no other changes. Then, without loss of generality, the trend component T is fitted by the least square method with quadratic polynomial. Next, the residual component R is predicted by the modified single dendrite neuron. Finally, add up three obtained subseries to generate the predicted result of SDN. Regarding to the designing of SDN, some noteworthy remarks are listed as follows:

- The use of STD provides two remarkable superiorities, 1) the possibility of separate predicting procedure translates into SDN's time efficiency, 2) the mathematical feature of outlier insensitivity gives SDN anti-outlier ability as well.
- The modification of single dendrite neuron not only solves the original DNM's gradient disappear and dendrite vanishing problems, but also improves the time efficiency of SDN.
- Compared to the existing models, SDN has simpler structure and faster speed, resulting in

improved possibility for it to be employed in practical usage scenarios.

Chapter 4

Experiment and Discussion

4.1 Preliminary

4.1.1 Data declaration

The datasets used in this study are gathered by environmental monitoring sensors in a real greenhouse. Each value of a single dynamic climate parameter is monitored per minute. Due to the growing crops' difference, these datasets have disparate lengths. In experimentation, the former 70% are used as training set while the rest 30% are testing set. Details are shown in Table 4.1.

Noted that the last column "Lyapunov exponent" (LE) denotes the maximum LE value of each series. The LE is calculated by Wolf's method [69] and it determines a notion of predictability for a dynamical system (in this case is time series). A positive maximum LE usually means the system is chaotic, this can be a convincing evidence for researchers to employ machine learning-based models to tackle such prediction tasks.

Table 4.1: Details of datasets.

Crop label	Climate parameter	Measuring unit	Monitoring period	Sequence length	Lyapunov exponent
Cucumber	Temperature	°C	05.14 13:00 - 07.29 10:00	85,148	0.0817
	Humidity	%			0.2003
	CO_2 concentration	ppm			0.0306
Pepper	Temperature	°C	05.14 01:00 - 07.05 22:00	66,920	0.0122
	Humidity	%			0.1143
	CO_2 concentration	ppm			0.1872
Tomato	Temperature	°C	08.28 09:00 - 02.05 15:00	192,143	0.0493
	Humidity	%			0.0358
	CO_2 concentration	ppm			0.0056

4.1.2 Normalization

The original DNM's inputs are normalized to the range of [0, 1] to better satisfy the sigmoid function in its synapse layer. Similarly, in SDN's nonlinear fitter, the same normalization is kept for modified single dendrite neuron so that the computational cost can be reduced. Noted that only residual subseries R are predicted by single dendrite neuron, the normalization is formulated by

$$r_t = \frac{r_t(original) - min(R)}{max(R) - min(R)} \tag{4.1}$$

where $min(R)$ and $max(R)$ are the minimum and maximum value of subseries R. Moreover, the output of single dendrite neuron is denormalized so that the result can match the predicted subseries S and T for final evaluation.

4.1.3 Evaluation metrics

Three metrics are employed to evaluated the prediction accuracy of SDN and its competitors, i.e., the mean square error (MSE), mean absolute error (MAE), and mean absolute percentage error

(MAPE). Formulated as

$$MSE = \frac{1}{n}(O - \mathcal{T})^2 \tag{4.2}$$

$$MAE = \frac{1}{n}|O - \mathcal{T}| \tag{4.3}$$

$$MAPE = \frac{1}{n}|\frac{O - \mathcal{T}}{\mathcal{T}}| \tag{4.4}$$

where n is the data length.

4.2 Performance comparison

4.2.1 Competitors

The predicting performance of SDN is fully compared with both original DNM and some of the most widely-used machine learning models, including multi-layer perceptron (MLP), adaptive neuro-fuzzy inference system (ANFIS), RNN, and SVM. Each competitor is run under the same condition (i7-11700 @ 2.50GHz chip and MATLAB R2022a software), the details of some other settings are summarized in Table 4.2.

Particularly, the network structure and parameters of competitors are widely considered. In MLP and RNN, different hidden layers are chosen. In ANFIS, two generation methods are employed to build the fuzzy inference system, i.e., grid partitioning and fuzzy c-means clustering. In SVM, the radial basis function and the sigmoid function are used as SVM's kernel function. Besides, the best results of these competitors are selected to compare with only one group of fixed parameter-generated SDN to verify the SDN's superior performance. Last but not the least, the parameter groups of SDN are further discussed in 4.3.1.

Table 4.2: Details of SDN and its competitors.

Model	Detail			
	structure	input dimension	training iteration	number of experiments
SDN	modified single dendrite	2	200	27
DNM	2 dendrites 3 dendrites	2	200	2
MLP	10 hidden layers 20 hidden layers	2	200	2
ANFIS	grid partitioning fuzzy c-means clustering	2	200	2
RNN	10 hidden layers 20 hidden layers	2	200	2
SVM	radial basis function sigmoid function	2	200	2

4.2.2 Results

Table 4.3, 4.4, and 4.5 show the compared results of all competitors under three evaluation metrics mentioned in 4.1.3. From the tables we can see that the SDN wins five, seven, and seven times at nine datasets referring to MSE, MAE, and $MAPE$, respectively. Which obviously proves that the SDN outperforms other models.

In addition, the time cost of each model is listed in Table 4.6, where the SDN shows a incredible time efficiency than any other competitors. This amazing feature makes SDN becomes the most potential model that could be used in real greenhouse climate monitoring. Given that the sensor-gathered time series in a greenhouse are usually minutely observed (e.g., in this case study), thus, the predicting model is better to be fast and requires less training cost, which are exactly the SDN has achieved.

Table 4.3: Mean square error (MSE) of all competitors.

Dataset		Model					
		SDN	DNM	MLP	ANFIS	RNN	SVM
Cucumber	Temp	**4.97E-02**	1.48E+00	1.22E+01	6.81E-01	2.72E-01	2.75E+01
	Humi	**2.07E+00**	1.36E+02	5.12E+01	1.09E+01	1.40E+01	6.84E+02
	CO_2	3.83E+03	1.45E+05	1.13E+05	**8.46E+02**	1.73E+03	7.65E+04
Pepper	Temp	**1.49E-01**	1.11E+01	4.21E+00	1.21E+00	5.66E-01	4.66E+01
	Humi	**2.62E+00**	6.38E+01	5.27E+01	8.65E+00	5.31E+00	5.81E+02
	CO_2	1.49E+04	2.42E+04	4.60E+04	6.55E+04	**1.39E+04**	3.03E+04
Tomato	Temp	3.31E+00	2.79E+02	6.59E+01	**2.89E-01**	1.38E+00	8.61E+03
	Humi	**6.59E-01**	8.31E+01	2.22E+01	6.41E+00	2.71E+00	1.42E+02
	CO_2	5.96E+02	1.07E+05	1.23E+04	**5.37E+02**	3.28E+04	1.37E+05
Total wins		**5**	0	0	3	1	0

Table 4.4: Mean absolute error (MAE) of all competitors.

Dataset		Model					
		SDN	DNM	MLP	ANFIS	RNN	SVM
Cucumber	Temp	**1.61E-01**	6.15E-01	3.05E+00	6.22E-01	4.74E-01	3.83E+00
	Humi	**1.15E+00**	9.47E+00	6.52E+00	3.01E+00	3.30E+00	2.36E+01
	CO_2	2.35E+01	3.63E+02	1.52E+02	**1.78E+01**	3.08E+01	1.38E+02
Pepper	Temp	**2.75E-01**	2.22E+00	1.69E+00	7.32E-01	4.69E-01	5.19E+00
	Humi	**1.30E+00**	5.89E+00	5.78E+00	2.51E+00	1.62E+00	2.05E+01
	CO_2	**2.27E+01**	5.14E+01	8.58E+01	6.55E+04	2.83E+01	7.87E+01
CTomato	Temp	7.01E-01	1.54E+01	5.44E+00	**3.88E-01**	9.58E-01	1.14E+01
	Humi	**6.62E-01**	7.98E+00	2.78E+00	2.38E+00	1.23E+00	8.24E+00
	CO_2	**1.61E+01**	2.59E+02	9.31E+01	2.13E+01	1.52E+02	3.19E+02
Total wins		**7**	0	0	2	0	0

4.3 Discussion

4.3.1 Parameters in SDN

The SDN's soma layer has two customed parameters which control the final output of the nonlinear fitter, i.e., the positive constant κ and threshold θ_s as mentioned in 3.3.3. These two parameters along with the learning rate η are set manually. Based on the previous knowledge, each parameter takes three proper values, which means $3^3 = 27$ groups of parameters are tested, as listed in Table 4.7.

Table 4.5: Mean absolute percentage error (*MAPE*) of all competitors.

Dataset		Model					
		SDN	DNM	MLP	ANFIS	RNN	SVM
Cucumber	Temp	**4.58E-09**	6.31E-07	1.70E-06	8.24E-07	2.79E-07	5.19E-06
	Humi	**5.30E-08**	4.70E-06	1.72E-06	1.42E-06	8.82E-07	1.17E-05
	CO_2	1.44E-06	1.02E-05	7.62E-06	**5.67E-07**	1.77E-06	1.28E-05
Pepper	Temp	**1.06E-07**	2.59E-06	1.90E-07	1.32E-06	1.35E-07	8.01E-06
	Humi	**3.08E-07**	4.59E-06	1.17E-06	1.67E-06	4.76E-07	1.62E-05
	CO_2	**4.71E-06**	4.96E-06	1.11E-05	6.55E+04	5.19E-06	9.88E-06
Toamto	Temp	1.38E-06	5.41E-06	5.22E-07	**3.70E-07**	4.38E-07	2.69E-06
	Humi	**1.10E-08**	1.79E-06	1.51E-07	4.18E-07	6.78E-08	1.70E-07
	CO_2	**1.78E-07**	3.70E-06	1.42E-06	3.80E-07	3.20E-06	6.98E-06
Total wins		**7**	0	0	2	0	0

Table 4.6: Time cost of all competitors.

Dataset		Model					
		SDN	DNM	MLP	ANFIS	RNN	SVM
Cucumber	Temp	**2.3**	430.4	6.1	35.6	7.2	75.7
	Humi	**2.3**	404.2	4.5	35.1	11.6	450.6
	CO_2	**2.5**	454.7	4.3	37.3	12.0	298.7
Pepper	Temp	**1.6**	303.2	4.5	27.7	9.9	27.6
	Humi	**1.6**	277.3	4.3	27.5	12.3	129.4
	CO_2	**1.6**	304.6	4.3	129.2	9.7	187.4
Toamto	Temp	**7.5**	6374.1	10.0	79.9	22.4	532.0
	Humi	**8.0**	6247.9	10.0	77.1	22.4	1097.5
	CO_2	**7.6**	6232.0	7.8	77.7	22.5	3442.7
Total wins		**9**	0	0	0	0	0

After a comprehensive experimentation and statistical analyzation, the Friedman rank test of all 27 groups of parameters-generated predicting results under three evaluation metrics is shown in Table 4.8. From the results we can tell that group 10 ($\kappa = 1$, $\theta = 0$, and $\eta = 0.2$) has the highest average ranking, so that the parameters used in SDN are determined. Also, it is worth noting that the mean values of Friedman rankings are close to each other, and the minimum is even bigger than 10, which suggest that the parameters' difference only has very little impact on the results. To verify this point of view, a Wilcoxon test is implemented, and the p-values are shown in Table 4.9. From the table we can find that under the MAE's evaluation, only group 12 and 21 perform significantly worse than group 10, which indicates that there are barely no difference between 27 groups of parameters-generated predicting results. In other words, the SDN has a strong robustness that can almost ignore the customed parameters' influence.

4.3.2 Influence of outliers

A noteworthy point in Table 4.3 is that the SDN loses on all three CO_2 concentration datasets. One reason is that there are outliers among the test set of datasets, as shown in Fig. 4.1. During the predictions, SDN can ignore the outliers due to STD's preprocessing, while the victor on MSE (like RNN) just winning because of overfitting. As shown in Fig. 4.2. This is another advantage of SDN, given that the sensors in a greenhouse could have mechanical faults sometime and provide an outlier value of dynamic climate parameters. On the other hand, this phenomenon only occurs in MSE comparison, that is because the MSE includes a square calculation which magnify the error a lot.

Table 4.7: 27 groups of SDN's parameters.

	Parameter				Parameter				Parameter		
Group	κ	θ_s	η	Group	κ	θ_s	η	Group	κ	θ_s	η
1	*5*	*0*	*0.2*	10	*1*	*0*	*0.2*	19	*10*	*0*	*0.2*
2	*5*	*0.5*	*0.2*	11	*1*	*0.5*	*0.01*	20	*10*	*0.5*	*0.2*
3	*5*	*1*	*0.2*	12	*1*	*1*	*0.2*	21	*10*	*1*	*0.2*
4	*5*	*0*	*0.05*	13	*1*	*0*	*0.05*	22	*10*	*0*	*0.05*
5	*5*	*0*	*0.1*	14	*1*	*0*	*0.1*	23	*10*	*0*	*0.1*
6	*5*	*0.5*	*0.05*	15	*1*	*0.5*	*0.05*	24	*10*	*0.5*	*0.05*
7	*5*	*0.5*	*0.1*	16	*1*	*0.5*	*0.1*	25	*10*	*0.5*	*0.1*
8	*5*	*1*	*0.05*	17	*1*	*1*	*0.05*	26	*10*	*1*	*0.05*
9	*5*	*1*	*0.1*	18	*1*	*1*	*0.1*	27	*10*	*1*	*0.1*

4.4 Conclusion

The increasing requirements of solving food shortage and ensuring food security have taken greenhouse cultivation into scientists' consideration. In this modern technology, monitoring and controlling the dynamic climate parameters in a greenhouse are the most important matter. The greenhouse time series generated by sensors can be predicted by machine learning-based models. In this study, we propose a novel seasonal-trend decomposition-based single dendrite neuron framework (SDN) to predict greenhouse time series. Due to the most optimal preprocessing method and well-designed nonlinear fitter, our proposed SDN performs obviously better than its competitors including the original DNM, MLP, ANFIS, RNN, and SVM.

The experimental results also confirm two of the remarkable features of SDN, 1) tremendous time efficiency, and 2) insensitivity to outliers. These two advantages hugely improve the possibility to employ SDN in practical usage scenarios, given that the purpose of developing predicting model is for it to monitoring dynamic climate parameters in a greenhouse rather than consuming massive of computational resources just for another deep structure. From this perspective, the proposed SDN is significantly better than any other existing greenhouse time series predicting models.

Table 4.8: Friedman test of different parameters-generated predicting results under MSE, MAE, and $MAPE$.

	Friedman test rank					Friedman test rank					Friedman test rank			
Group	MSE	MAE	$MAPE$	Mean	Group	MSE	MAE	$MAPE$	Mean	Group	MSE	MAE	$MAPE$	Mean
10	**11.89**	**8.56**	**12.78**	**11.07**	6	12.78	12.89	12.00	12.56	1	13.67	13.22	15.33	14.07
19	10.44	11.67	11.56	11.22	13	13.67	9.44	14.89	12.67	22	14.33	11.11	17.22	14.22
9	12.67	11.78	10.11	11.52	18	11.89	11.44	15.33	12.89	15	13.67	14.33	17.89	15.30
25	12.22	12.56	11.00	11.93	7	12.33	14.56	12.00	12.96	12	15.33	16.00	15.11	15.48
8	15.44	12.56	7.89	11.96	23	13.67	12.00	13.67	13.11	4	15.11	13.67	18.56	15.78
24	12.78	12.89	10.67	12.11	3	13.56	13.78	12.56	13.30	11	17.00	15.78	15.78	16.19
14	13.33	10.56	12.67	12.19	26	13.89	16.89	9.44	13.41	20	17.11	22.22	15.44	18.26
2	11.67	13.56	11.67	12.30	16	13.22	12.44	15.89	13.85	27	19.11	23.33	18.33	20.26
5	11.67	10.78	14.89	12.44	17	13.00	13.00	15.78	13.93	21	22.56	27.00	19.56	23.04

Table 4.9: Wilcoxon test of group 10 v.s. other groups-generated predicting results under MSE, MAE, and $MAPE$.

v.s. Group	p-values			v.s. Group	p-values			v.s. Group	p-values		
	MSE	MAE	$MAPE$		MSE	MAE	$MAPE$		MSE	MAE	$MAPE$
10	-	-	-	6	≥0.2	0.129	≥0.2	1	≥0.2	≥0.2	≥0.2
19	≥0.2	≥0.2	≥0.2	13	≥0.2	≥0.2	≥0.2	22	≥0.2	≥0.2	≥0.2
9	≥0.2	≥0.2	≥0.2	18	≥0.2	0.098	≥0.2	15	≥0.2	0.074	≥0.2
25	≥0.2	≥0.2	≥0.2	7	≥0.2	0.074	≥0.2	**12**	0.129	**0.004**	≥0.2
8	≥0.2	≥0.2	≥0.2	23	≥0.2	≥0.2	≥0.2	4	≥0.2	≥0.2	≥0.2
24	≥0.2	≥0.2	≥0.2	3	≥0.2	≥0.2	≥0.2	11	≥0.2	0.074	≥0.2
14	≥0.2	≥0.2	≥0.2	26	≥0.2	≥0.2	≥0.2	20	≥0.2	0.020	≥0.2
2	≥0.2	≥0.2	≥0.2	16	≥0.2	0.129	≥0.2	27	≥0.2	0.008	≥0.2
5	≥0.2	≥0.2	≥0.2	17	≥0.2	0.074	≥0.2	**21**	≥0.2	**0.004**	0.129

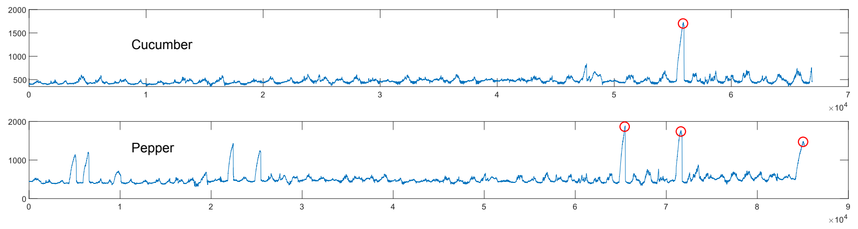

Figure 4.1: The outliers in CO_2 concentration of cucumber and pepper datasets.

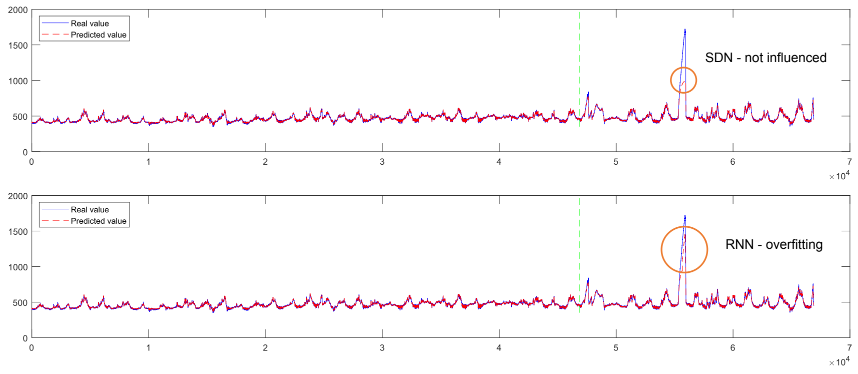

Figure 4.2: The outliers' influence on the predictions generated by SDN & RNN on pepper's CO_2 concentration datasets.

Bibliography

[1] M. M. Drugan, "Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms," *Swarm and Evolutionary Computation*, vol. 44, pp. 228–246, 2019.

[2] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[3] P. J. Van Laarhoven and E. H. Aarts, "Simulated annealing," in *Simulated Annealing: Theory and Applications*. Springer, 1987, pp. 7–15.

[4] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-International Conference on Neural Networks*, vol. 4. IEEE, 1995, pp. 1942–1948.

[5] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[6] D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," *Applied Mathematics and Computation*, vol. 214, no. 1, pp. 108–132, 2009.

[7] Z.-H. Zhan, L. Shi, K. C. Tan, and J. Zhang, "A survey on evolutionary computation for complex continuous optimization," *Artificial Intelligence Review*, pp. 1–52, 2022.

[8] M. Gen and L. Lin, "Multiobjective evolutionary algorithm for manufacturing scheduling problems: State-of-the-art survey," *Journal of Intelligent Manufacturing*, vol. 25, pp. 849–866, 2014.

[9] V. Jyothi, B. R. Kumar, P. K. Rao, and D. R. K. Reddy, "Image fusion using evolutionary algorithm (ga)," *Int. J. Comp. Tech. Appl.*, vol. 2, no. 2, pp. 322–326, 2011.

[10] R. Kala, A. Shukla, and R. Tiwari, "Dynamic environment robot path planning using hierarchical evolutionary algorithms," *Cybernetics and Systems: An International Journal*, vol. 41, no. 6, pp. 435–454, 2010.

[11] C. A. C. Coello, "Evolutionary multi-objective optimization in finance," in *Handbook of Research on Nature-Inspired Computing for Economics and Management*. IGI Global, 2007, pp. 74–89.

[12] R. L. Haupt and S. E. Haupt, *Practical genetic algorithms*. John Wiley & Sons, 2004.

[13] B. Alhijawi and A. Awajan, "Genetic algorithms: theory, genetic operators, solutions, and applications," *Evolutionary Intelligence*, pp. 1–12, 2023.

[14] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies–a comprehensive introduction," *Natural computing*, vol. 1, pp. 3–52, 2002.

[15] H.-G. Beyer, *The theory of evolution strategies*. Springer Science & Business Media, 2001.

[16] D. V. Arnold and H.-G. Beyer, "A comparison of evolution strategies with other direct search methods in the presence of noise," *Computational Optimization and Applications*, vol. 24, pp. 135–159, 2003.

[17] W. B. Langdon, "Genetic programming and data structures: genetic programming+ data structures= automatic programming!" 1998.

[18] P. G. Espejo, S. Ventura, and F. Herrera, "A survey on the application of genetic programming to classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 2, pp. 121–144, 2009.

[19] M. Affenzeller, S. Wagner, S. Winkler, and A. Beham, *Genetic algorithms and genetic programming: modern concepts and practical applications*. Crc Press, 2009.

[20] M. Caramia, P. Dell'Olmo, M. Caramia, and P. Dell'Olmo, "Multi-objective optimization," *Multi-objective Management in Freight Logistics: Increasing Capacity, Service Level, Sustainability, and Safety with Optimization Algorithms*, pp. 21–51, 2020.

[21] N. Gunantara, “A review of multi-objective optimization: Methods and its applications,” *Cogent Engineering*, vol. 5, no. 1, p. 1502242, 2018.

[22] R. T. Marler and J. S. Arora, “Survey of multi-objective optimization methods for engineering,” *Structural and multidisciplinary optimization*, vol. 26, pp. 369–395, 2004.

[23] W. Zhao, Z. Zhang, and L. Wang, “Manta ray foraging optimization: An effective bio-inspired optimizer for engineering applications,” *Engineering Applications of Artificial Intelligence*, vol. 87, p. 103300, 2020.

[24] A. A. Heidari, I. Aljarah, H. Faris, H. Chen, J. Luo, and S. Mirjalili, “An enhanced associative learning-based exploratory whale optimizer for global optimization,” *Neural Computing and Applications*, vol. 32, no. 9, pp. 5185–5211, 2020.

[25] K. Hussain, M. N. M. Salleh, S. Cheng, and Y. Shi, “On the exploration and exploitation in popular swarm-based metaheuristic algorithms,” *Neural Computing and Applications*, vol. 31, no. 11, pp. 7665–7683, 2019.

[26] H. Yang, Y. Yu, J. Cheng, Z. Lei, Z. Cai, Z. Zhang, and S. Gao, “An intelligent metaphor-free spatial information sampling algorithm for balancing exploitation and exploration,” *Knowledge-Based Systems*, p. 109081, 2022.

[27] F. Wang, H. Zhang, K. Li, Z. Lin, J. Yang, and X.-L. Shen, “A hybrid particle swarm optimization algorithm using adaptive learning strategy,” *Information Sciences*, vol. 436, pp. 162–177, 2018.

[28] J. Yang, Y. Zhang, Z. Wang, Y. Todo, B. Lu, and S. Gao, “A cooperative coevolution wingsuit flying search algorithm with spherical evolution,” *International Journal of Computational Intelligence Systems*, vol. 14, no. 1, pp. 1–19, 2021.

[29] J. Zhao, S. Liu, M. Zhou, X. Guo, and L. Qi, “Modified cuckoo search algorithm to solve economic power dispatch optimization problems,” *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 4, pp. 794–806, 2018.

[30] Y. Wang, S. Gao, Y. Yu, Z. Wang, J. Cheng, and T. Yuki, “A gravitational search algorithm with chaotic neural oscillators,” *IEEE Access*, vol. 8, pp. 25 938–25 948, 2020.

[31] X. Li, H. Yang, J. Li, Y. Wang, and S. Gao, “A novel distributed gravitational search algorithm with multi-layered information interaction,” *IEEE Access*, vol. 9, pp. 166 552–166 565, 2021.

[32] N. Hansen and A. Ostermeier, “Completely derandomized self-adaptation in evolution strategies,” *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[33] N. Hansen, S. D. Müller, and P. Koumoutsakos, “Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es),” *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003.

[34] R. Rayhana, G. Xiao, and Z. Liu, “Internet of things empowered smart greenhouse farming,” *IEEE Journal of Radio Frequency Identification*, vol. 4, no. 3, pp. 195–211, 2020.

[35] P. C. West, J. S. Gerber, P. M. Engstrom, N. D. Mueller, K. A. Brauman, K. M. Carlson, E. S. Cassidy, M. Johnston, G. K. MacDonald, D. K. Ray *et al.*, “Leverage points for improving global food security and the environment,” *Science*, vol. 345, no. 6194, pp. 325–328, 2014.

[36] H. C. J. Godfray, J. R. Beddington, I. R. Crute, L. Haddad, D. Lawrence, J. F. Muir, J. Pretty, S. Robinson, S. M. Thomas, and C. Toulmin, “Food security: the challenge of feeding 9 billion people,” *science*, vol. 327, no. 5967, pp. 812–818, 2010.

[37] M. Cole, M. Augustin, M. Robertson, and J. Manners, “The science of food security. npj science of food, 2 (14),” *Doi. org/10.1038/s41538-018-0021-9*, 2018.

[38] A. Escamilla-García, G. M. Soto-Zarazúa, M. Toledano-Ayala, E. Rivas-Araiza, and A. Gastélum-Barrios, “Applications of artificial neural networks in greenhouse technology and overview for smart agriculture development,” *Applied Sciences*, vol. 10, no. 11, p. 3835, 2020.

[39] M. J. Hoque, M. R. Ahmed, and S. Hannan, “An automated greenhouse monitoring and controlling system using sensors and solar power,” *European Journal of Engineering and Technology Research*, vol. 5, no. 4, pp. 510–515, 2020.

[40] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[41] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.” *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[43] J. Mahanta, “Introduction to neural networks, advantages and applications,” *Towards Data Science*, vol. 13, 2017.

[44] Y.-c. Wu and J.-w. Feng, “Development and application of artificial neural network,” *Wireless Personal Communications*, vol. 102, pp. 1645–1656, 2018.

[45] L.-Y. Chiu, D. J. A. Rustia, C.-Y. Lu, and T.-T. Lin, “Modelling and forecasting of greenhouse whitefly incidence using time-series and arimax analysis,” *IFAC-PapersOnLine*, vol. 52, no. 30, pp. 196–201, 2019.

[46] Y. Liu, D. Li, S. Wan, F. Wang, W. Dou, X. Xu, S. Li, R. Ma, and L. Qi, “A long short-term memory-based model for greenhouse climate prediction,” *International Journal of Intelligent Systems*, vol. 37, no. 1, pp. 135–151, 2022.

[47] R. J. Frank, N. Davey, and S. P. Hunt, “Time series prediction and neural networks,” *Journal of intelligent and robotic systems*, vol. 31, pp. 91–103, 2001.

[48] L. S. de Oliveira, S. B. Gruetzmacher, and J. P. Teixeira, “Covid-19 time series prediction,” *Procedia Computer Science*, vol. 181, pp. 973–980, 2021.

[49] H. Nasiri and M. M. Ebadzadeh, “Mfrfnn: multi-functional recurrent fuzzy neural network for chaotic time series prediction,” *Neurocomputing*, vol. 507, pp. 292–310, 2022.

[50] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[51] S. Elsayed, D. Thyssens, A. Rashed, H. S. Jomaa, and L. Schmidt-Thieme, "Do we really need deep learning models for time series forecasting?" *arXiv preprint arXiv:2101.02118*, 2021.

[52] L. Jiang, Z. Tao, J. Zhu, J. Zhang, and H. Chen, "Exploiting pso-svm and sample entropy in bemd for the prediction of interval-valued time series and its application to daily pm2. 5 concentration forecasting," *Applied Intelligence*, pp. 1–15, 2022.

[53] M. Rakhra, P. Soniya, D. Tanwar, P. Singh, D. Bordoloi, P. Agarwal, S. Takkar, K. Jairath, and N. Verma, "Crop price prediction using random forest and decision tree regression:-a review," *Materials Today: Proceedings*, 2021.

[54] A. Buevich, A. Sergeev, A. Shichkin, and E. Baglaeva, "A two-step combined algorithm based on narx neural network and the subsequent prediction of the residues improves prediction accuracy of the greenhouse gases concentrations," *Neural Computing and Applications*, vol. 33, pp. 1547–1557, 2021.

[55] W. Cai, R. Wei, L. Xu, and X. Ding, "A method for modelling greenhouse temperature using gradient boost decision tree," *Information Processing in Agriculture*, vol. 9, no. 3, pp. 343–354, 2022.

[56] G. Van Straten, L. Van Willigenburg, and R. Tap, "The significance of crop co-states for receding horizon optimal control of greenhouse climate," *Control Engineering Practice*, vol. 10, no. 6, pp. 625–632, 2002.

[57] G. Samuolienė, R. Sirtautas, A. Brazaitytė, and P. Duchovskis, "Led lighting and seasonality effects antioxidant properties of baby leaf lettuce," *Food chemistry*, vol. 134, no. 3, pp. 1494–1499, 2012.

[58] J.-T. Ding, H.-Y. Tu, Z.-L. Zang, M. Huang, and S.-J. Zhou, "Precise control and prediction of the greenhouse growth environment of dendrobium candidum," *Computers and electronics in agriculture*, vol. 151, pp. 453–459, 2018.

[59] T. Moon and J. E. Son, "Knowledge transfer for adapting pre-trained deep neural models to predict different greenhouse environments based on a low quantity of data," *Computers and Electronics in Agriculture*, vol. 185, p. 106136, 2021.

[60] A. Rojas-Rishor, J. Flores-Velazquez, E. Villagran, and C. E. Aguilar-Rodríguez, "Valuation of climate performance of a low-tech greenhouse in costa rica," *Processes*, vol. 10, no. 4, p. 693, 2022.

[61] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning, "Stl: A seasonal-trend decomposition," *J. Off. Stat*, vol. 6, no. 1, pp. 3–73, 1990.

[62] M. Theodosiou, "Forecasting monthly and quarterly time series using stl decomposition," *International Journal of Forecasting*, vol. 27, no. 4, pp. 1178–1195, 2011.

[63] H. He, S. Gao, T. Jin, S. Sato, and X. Zhang, "A seasonal-trend decomposition-based dendritic neuron model for financial time series prediction," *Applied Soft Computing*, vol. 108, p. 107488, 2021.

[64] B. Lenze, "How to make sigma-pi neural networks perform perfectly on regular training sets," *Neural Networks*, vol. 7, no. 8, pp. 1285–1293, 1994.

[65] L. A. Lyutikova, "Sigma-pi neural networks: error correction methods," *Procedia computer science*, vol. 145, pp. 312–318, 2018.

[66] S. Gao, M. Zhou, Y. Wang, J. Cheng, H. Yachi, and J. Wang, "Dendritic neuron model with effective learning algorithms for classification, approximation, and prediction," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 2, pp. 601–614, 2018.

[67] S. Gao, M. Zhou, Z. Wang, D. Sugiyama, J. Cheng, J. Wang, and Y. Todo, "Fully complex-valued dendritic neuron model," *IEEE transactions on neural networks and learning systems*, 2021.

[68] X. Luo, X. Wen, M. Zhou, A. Abusorrah, and L. Huang, "Decision-tree-initialized dendritic neuron model for fast and accurate data classification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 9, pp. 4173–4183, 2021.

[69] A. Wolf, J. B. Swift, H. L. Swinney, and J. A. Vastano, "Determining lyapunov exponents from a time series," *Physica D: nonlinear phenomena*, vol. 16, no. 3, pp. 285–317, 1985.

Acknowledgements

I would like to express my deepest gratitude to my PhD advisor Dr.Tang and Dr.Gao, for the invaluable guidance and support throughout my academic journey. Without the encouragement and insightful comments, this achievement would not have been possible.

I would also like to thank the members of my dissertation committee for their time, effort, and valuable feedback. Their comments and suggestions have greatly contributed to the quality of my research.

I am also grateful to my colleagues and friends who have provided me with support, encouragement, and inspiration throughout this journey. Their encouragement and motivation have kept me going during difficult times.

Lastly, I would like to thank my family for their unwavering support, love, and understanding. Their constant support and belief in me have been my source of strength and motivation.

In addition, I would like to extend a special thank you to the fictional characters Venat and Edelgard von Hresvelg for inspiring me to pursue my passions and reminding me to always strive for greatness. Their stories have taught me important lessons about perseverance, determination, and the power of imagination. I am truly grateful for their presence in my life, even though they exist only in the world of fantasy.

I am deeply grateful for all the help and support that I have received during my PhD, and I will cherish this experience for the rest of my life.