Chapter 4

Modern Hybrids of Squirrel Search Algorithm

Slow convergence, local optima stagnation, scalability, and non-adaptiveness are a few of the major issues with EC that limit its performance in high-dimensional industrial applications, pattern mining, and various ML applications. The preceding chapter primarily focused on solving the convergence and local optima stagnation issues of the self-adaptive variant of PSO (DWCN-PSO). However, Squirrel Search Algorithm (SSA) has significantly better exploitation capability than PSO but is non-adaptive. Also, a scalability issue limits its ability to solve modern applications. Therefore, the primary focus of this chapter is to address the poor diversity, adaptiveness, and scalability issues of EC by designing and developing modern hybrids of SSA, namely Q-Learning-based SSA, chaotic SSA, and finally, Quantum-assisted Chaotic SSA. We are also attempting to overcome the limitations of the ML approach in terms of its inability to address the "curse of dimensionality" by proposing modern hybrids and applying them to optimal feature selection problems.

4.1 Introduction

With the onset of the Industry 4.0 era, enormous and widespread changes have occurred in all aspects of the industry. Due to the rapid advancement of technology, a voluminous amount of data is routinely collected, and processing this data itself is a tedious task. Real-world applications frequently involve high dimensionality as well as massive amounts of data. As a result, automated tools are required to handle such huge and high-dimensional data. Data mining and ML have recently become essential fields and are at the forefront of numerous applications, including remote sensing, agricultural data, financial data, biological data analysis, and so on. Nowadays, even the healthcare industry employs ML and data mining processes to automate complex manual processing steps in order to make quick and accurate decisions [216, 217].

Data mining aims to gain and apply the information that emerges from dealing with massive databases of varying sizes. In contrast, ML techniques are used in a supervised, unsupervised and semi-supervised way. Among these, classification is the supervised approach that is widely used in real-world applications. Classification is also a data mining technique that predicts data instance group membership. It is a data analysis method that deals with class labels in various data types and aids in the discovery of meaningful patterns and associations in huge data sets. Such research will help in the deeper interpretation of massive amounts of data, resulting in more accurate prediction models. However, dimensionality is one of the most significant issues that might impede the data mining process due to the requirement of high processing costs. Therefore, feature selection is essential for simplifying the learning model, enhancing performance, reducing data dimensionality, and speeding up the learning process [149, 218].

Furthermore, the massively acquired datasets frequently contain redundant, useless, and noisy features. It is worth noting that the information contained by the features is more valuable than the volume of data, and the performance of ML techniques is heavily reliant on the quality of features and data pre-processing. One of the effective

ways to address this concern is through feature selection [219]. Feature selection is an essential strategy for simplifying the learning model, enhancing performance, reducing data dimensionality, and speeding up the learning process [149, 218]. The general aim is to reduce the amount of negative, redundant, and disruptive features in a dataset for fast and efficient data analysis while preserving accuracy. The complexity of the feature selection problem can be understood by an illustration. Suppose there are N features in a dataset initially; the size of the state space will grow exponentially, i.e., 2^N subsets of features, making feature selections a challenging and NP-hard problem with complex search space. It is practically impossible to evaluate all the subsets using modern computation power.

Traditional approaches for feature selection are computationally efficient, but they are suitable for unimodal search space and perform better on small-scale datasets without considering feature interaction problems. For large-scale datasets or multi-modal search space, feature selection has a very complex search space due to the feature interaction problem. As a result, heuristics or random search techniques are being used to discover effective results in a fair amount of time while adhering to an evaluation function and a stopping criterion. Many approaches have been proposed combining EC with the feature selection technique [220]. Most of them used hybridization with supervised learning techniques and are well-known as wrapper-based approaches. The wrapper approach works by searching the subset of features and selecting the best subset based on the predictive accuracy of the learning algorithm.

Generally, most of the feature selection methods suffer from local optima stagnation as well as high computational cost, and to overcome these issues, an efficient global search technique needs to be designed. Furthermore, scalability is another challenge that has arisen as a result of the emergence of big data [7]. In early 1989, a dataset with more than 20 features and the selection of features from this data was referred to as large-scale feature selection [153]. However, the number of features in several areas,

such as gene analysis, has grown from thousands to millions in recent years, increasing the computation cost and necessitating an advanced search mechanism, both of which have their own limitations. As a result, simply having a lot of computing power isn't going to solve the problem. Designing novel methods and searching mechanisms for handling this type of complex optimization problem is an emerging area of research. Fitness function and evaluation measures are also the deciding factors for large-scale problems. There is a vast scope for improving existing EC techniques for large-scale optimization problems. Thus, continuous efforts have been made by researchers over the last few years to improve the performance of existing EC approaches for large-scale optimization problems by integrating it with chaos theory [221], opposition-based learning [222], and so on.

In the previous chapter, GDWCN-PSO, an adaptive variant of PSO based on a directed weighted complex network and GA, has been proposed to address the premature convergence issue and utilized for a complex optimization problem. Since SSA [64] has better exploitation ability than PSO and has good cooperative characteristics among squirrels, it has been chosen for further improvement to suit modern ML applications. In the same spirit, interdisciplinary concepts like chaos theory, quantum mechanism, and AI techniques such as reinforcement learning are also utilized to address the limitations of EC methods. Among these, reinforcement learning does not require a complete model of the underlying problem since learning is performed by gathering experience, referred to as trial and error. It enables the use of independent learning agents, making it very suitable for fully decentralized problems. Moreover, it uses a single update formula, which lessens its computational cost. Further, Q-Learning (QL), a concept of reinforcement learning, is capable of enhancing the diversity of solutions; however, it has a weaker exploitation ability.

On the other hand, few EC methods have great exploitation ability; however, most of the existing optimizers are not adaptive, limiting their applications in large-scale

problems or interesting industrial and ML tasks. Thus, integrating the strength of QL and EC is now gaining attention for designing efficient techniques for solving modern complex problems. Furthermore, another concept, quantum computing, has recently gained popularity among optimization and ML research communities due to its exceptional representation ability in search space, diversity, and thus accelerating the development of quantum-inspired models for future applications and computation resources.

Interdisciplinary research is now more demanding in every field and opens a plethora of opportunities. Recently, genomic data analysis (consisting of millions of features) has been quite demanding and needs efficient automated tools. Moreover, genomic data may appear to be simple to handle in terms of volume, but it actually necessitates a complex process due to the complexity, heterogeneity, and hybrid of its features. Besides that, only a small number of genes out of a large number of genes are strongly correlated with a particular disease, and researchers have suggested that these few genes are sufficient biomarkers for specific disease identification [223, 224]. Selecting optimal genes can be posed as an optimization problem and gain the benefit of interdisciplinary research. Thus, we have devised our chapter's objectives in light of the challenging, complex modern applications and EC methods.

4.2 Motivation and Contributions

EC methods are well-known global optimizers that can solve complex optimization problems ranging from single modal to multi-modal, convex to non-convex optimization, and single objective to multi-objective. However, the main issue is their frequent premature convergence, which results in an inadequate contribution to data mining. The majority of existing optimizers are not adaptive in nature. Besides, they are computationally expensive, and designing an efficient optimizer is an open research area. Aside from this, NFL theorem [225] also supports the invention and modification of

existing EC, proving that there is no universal optimizer that provides the best optimal solution for all optimization problems. Recent trends also emphasize combining the strength of different approaches, such as ML and EC, to design efficient models while overcoming the individual limitations of methods, respectively. Further, one potential hybridization is to add a reinforcement learning aspect to these EC approaches. This concept is related to adaptive optimization, in which an intelligent element assists the optimization process to achieve a more intelligent search. A training variable can change parameters or assist the optimization system in making decisions during the search strategy. Reinforcement learning does not require a complete model of the underlying problem and enables the use of independent learning agents. QL is also insensitive to the initial exploration point and thus enhances the stability of the models. In brief, local optima stagnation, solution diversity, slow convergence, and inability to make a decision based on prior experience during the search strategy of existing optimizers need to be addressed for its successful application to problems of a new era.

On the other hand, optimal gene selection in bioinformatics for efficient analysis using ML is a challenging problem due to its high-dimensional features n, which exponentially expands the search space 2^n. Here, in the case of genomic data, the value of n can goes to more than a thousand. Besides these, genomic features are highly correlated, with only a few genes contributing to disease identification. While existing optimizers suffer from scalability issues, making the problem more difficult. Moreover, filter-based approaches ignore feature interaction, while embedded approaches increase model complexity. Wrapper-based approaches are computationally expensive, but performance-wise, wrapper-based approaches are preferable. Only providing computational resources will not solve the problems associated with the wrapper-based approach; we must also investigate optimizers for this purpose. Recently, the world's pandemic situation has accelerated genomic research for disease categorization and drug discovery. Besides that, quantum computing is attracting a lot of interest from the

computer science research communities due to its exceptional search exploration and representation ability. Therefore, we are motivated to solve the challenges encountered by both existing optimizers and genomic analysis.

In this chapter, we have attempted to overcome scalability issues as well as low convergence with existing optimizers while maintaining the quality of the solution by proposing a Q-Learning-based Squirrel Search Algorithm (QL-SSA) in the first part and later a novel Quantum-assisted Chaotic Squirrel Search Algorithm (QCSSA), which is a hybridization of interdisciplinary concepts of quantum computation, chaos theory, and SSA.

The main contributions of this chapter can be summarized as follows:

A. **Q-Learning-based Squirrel Search Algorithm (QL-SSA)**

1. Proposed a novel adaptive QL-SSA for solving optimal feature subset selection and classification problems by combining the strength of reinforcement learning technique and SSA.

2. The local search mechanism is modified using QL, enabling the search agent (squirrels) to learn from other search agents' experiences and make it an adaptive, intelligent, and robust method.

3. To validate the efficiency of the proposed QL-SSA for feature selection applications, two classifiers, KNN and SVM-RBF, and 20 real-world datasets are chosen while maintaining heterogeneity in the number of samples, features, and classes.

4. Furthermore, the results are validated on 2 genomic datasets to determine the efficacy of QL-SSA on high-dimensional datasets.

5. The reported results show that QL-SSA is more stable than SSA as average and best performance values are nearly the same, which is a good characteristic.

B. **Quantum-assisted Chaotic Squirrel Search Algorithm (QCSSA)**

6. Proposed a novel quantum-assisted chaotic version of SSA by integrating the concepts of quantum mechanics and chaos theory.

7. We have investigated three chaotic maps as a pseudo-random number generator to enhance optimizer convergence, which resulted in three variants of chaotic squirrel search algorithms (CSSA).

8. Concepts of quantum computing have been integrated with SSA in order to enhance solution diversity and exploration-exploitation balance. It resulted in three versions of QCSSA along with the quantum-assisted squirrel search algorithms (QSSA).

9. All proposals are validated against twenty-six classical benchmark functions of varying modality and complexities. Subsequently, they applied to high-dimensional optimal gene selection application for disease categorization on nine genomic datasets using a wrapper-based approach with SVM (RBF kernel).

10. A statistical test at 5% significance level is also performed, and the critical difference diagram is presented to provide the ranking to the optimizers for optimal gene selection.

11. Extensive experiments are performed on both benchmark functions and genomic datasets. It shows that for optimal gene selection, the QCSSA with neuron map outperforms the rest proposal as well as other contemporary optimizers in terms of fitness value and the number of selected optimal genes. Another quantum variant with a sine map provides even higher accuracy by utilizing a few more features, which are still very less in number than the competitors.

12. This study demonstrates that our proposals are well suited for mathematical functions as well as high-dimensional data, and they are capable of resolving the convergence and scalability issues with existing optimizers.

4.3 Background

This section briefly introduced the basic concepts needed to comprehend the proposed work.

4.3.1 Squirrel Search Algorithm

SSA [64] was inspired by the southern flying squirrels, which are considered to have aerodynamic properties due to patagia (a parachute-like membrane). It helps them to modify the lift and drag operation while moving from one tree to another. Basically, the mathematical model of SSA is based on intelligent dynamic optimum food foraging behavior and a special type of locomotion known as the gliding approach of squirrels. In [64], authors proposed a mathematical model in which the entire randomly generated population is assumed to be n, and each squirrel is present on a single tree. The entire forest is assumed to have only three types of trees: a hickory tree - a source of hickory nuts, an optimal food source; three oak trees - a source of acorn nuts, a suboptimal food source; and some normal trees. During the initial phase, each squirrel was assigned a random location and was later relocated to a location with a better food source in subsequent iterations. Climate conditions are also taken into account; for example, to meet their nutrient requirements in autumn, they typically eat acorn nuts, which are abundant and quickly fulfill their daily energy requirement. Further, they explore by gliding from one tree to another in search of the best food resource (hickory) and storing it for the winter when they need high energy, resulting in optimal utilization of nutritional food resources. By storing some hickory nuts in nests or cavities, they become less active and avoid the risk of predation, resulting in increased survival.

The overall process of food search consisted primarily of three types of movements. The best value based on fitness function is assumed to be on the optimal food source (hickory tree). The next three squirrels are at oak or acorn trees (suboptimal food source), while the rest are at normal trees. Because some squirrels on the normal trees have already visited the oak trees, it is assumed that they will move to the hickory tree, while others will move to the acorn trees. Thereafter, except for the squirrel on the hickory tree, the other squirrels will relocate and move to a more optimal food source. All of the squirrels on the acorn tree will move toward the hickory tree, while some of the squirrels on the normal trees will move toward the acorn tree and others toward the hickory tree. After relocation, the seasoning monitor condition is checked, and if it is satisfied, the squirrels are randomly relocated. The squirrel location and associated fitness value present at hickory are returned as the optimal solution at the end of the final iteration.

4.3.2 Feature Selection

EC methods, which are commonly used in wrapper-based approaches, demonstrate remarkable potential in solving feature selection problems. For example, a GA is used for feature selection after non-contributing genes are removed using the gene masking technique [226]. In another research, the recursive memetic algorithm was applied to seven microarray data sets for best gene selection and outperformed the memetic and GA [176]. Recently, an ABC, along with a Support Vector Machine (SVM), has been used to identify relevant genes and detect cancer using RNA sequence data [227]. Similarly, other works have been reported that solve high-dimensional gene selection problems using the NIAs [8, 228]. Researchers also applied a hybrid approach that combined both filter and wrapper-based approaches in a method called an embedded method to find relevant features [229]. However, the relevant gene selection problem has been thoroughly investigated, and the aforementioned approaches have their own

strengths and limitations, such as convergence and local optima stagnation, as well as scalability issues with existing NIAs.

4.4 Proposed QL-SSA

4.4.1 Q-Learning

QL [230] is a reinforcement learning algorithm [231] that learns by interacting with the environment in order to obtain the best reward based on the outcomes of previous actions. Let $P = \{p_1, p_2, p_3, ...p_n\}$ be a set of states and $\hat{Q} = \{\hat{q_1}, \hat{q_2}, \hat{q_3}, ...\hat{q_n}\}$ be the set of actions that the search agent can select for each state (p_i) from P. Whenever an action ($\hat{q_j}$) is performed for a state (p_j) from P, the agent receives an award. The objective of the agent is to map (P$\rightarrow$ Q) such that the reward function given below is maximized.

$$R(p_j) = r_j + \beta r_{j+1} + \beta^2 r_{j+2} + ... \tag{4.1}$$

where ($0 \leq \beta \leq 1$) is the discount parameter. If the value of (β) is closer to 0, the search agent is inclined to choose the immediate rewards, and if the value is closer to 1, the agent tends to choose the long-term reward. The search agent chooses to either explore a new solution or exploit the obtained solution based on which maximizes the reward function $R(p_j)$.

In reinforcement learning, Temporal difference (TD) [230] is the most used approach and it is a fusion between Monte Carlo (MC) algorithm and Markov Decision Process (MDP) [232]. The recursive QL algorithm is used in this approach. It calculates the immediate reward $\widetilde{Q}$ on executing an action $\hat{q_j}$ from state p_j using this equation:

$$\widetilde{Q}(p_j, \hat{q_j}) = r(p_j, \hat{q_j}) + \beta \; max\widetilde{Q}(\alpha \; (p_j, \hat{q_j}), \hat{q_a}) \tag{4.2}$$

where $\alpha \; (p_j, \hat{q_j})$ is the state that arose after performing action $\hat{q_j}$ on state p_j.

One-step Q-Learning:

$$\widetilde{Q}(p_j, \hat{q}_j) = (1-\gamma)\widetilde{Q}(p_j, \hat{q}_j) + \gamma\ (r(p_j, \hat{q}_j)) + \beta\ max\ \widetilde{Q}(\alpha(p_j, \hat{q}_j), \hat{q}_a) \tag{4.3}$$

where γ is the learning rate and is in the range [0,1].

4.4.2 Application to Optimal Feature Selection and Classification

SSA is an optimization algorithm that mimics the dynamic foraging approach using the gliding process of squirrels during food search. SSA is briefly described in Section 4.3.1. The proposed method QL-SSA integrates the concept of QL with SSA, in which squirrels are relocated in the search space using QL techniques as shown in Figure 4.1.

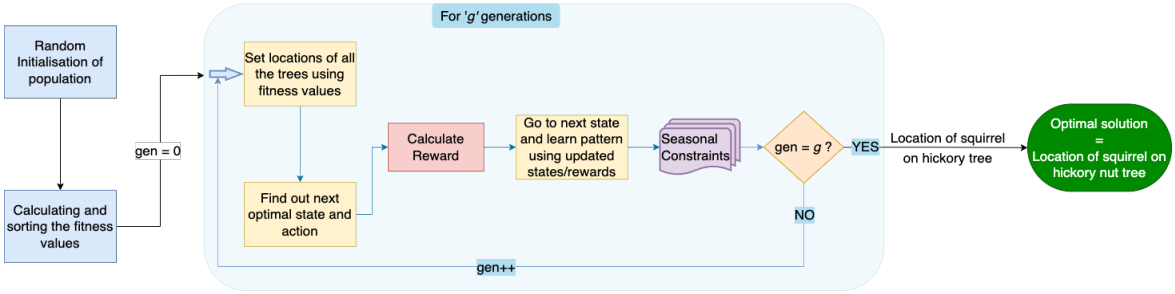


Figure 4.1: An overview of the steps involved in the QL-SSA method.

Premature convergence or local optima stagnation is a common issue in the meta-heuristic approach in a multi-modal search space. It is due to the fact that each individual squirrel has a lack of memory or intelligence, and hence they are incapable of remembering the previously reached optima. Therefore we have proposed QL-SSA to overcome this limitation of SSA. Instead of a local search of standard SSA, the QL approach has been utilized for intelligent and adaptive searches in the proposal. Each squirrel act as a learning agent which performs some action in a given state, and on the basis of the action, it either gets an immediate reward or long-term reward or

penalize. The states that comprise the environment are all feasible solutions feature subsets. While the solution is a boolean vector indicating which features are part of the feature subset. For a feature selection problem, an action consists of adding or deleting a feature from the current subset, which involves flipping a bit in the current solution. Further, the reward corresponding to (state, action) is computed using accuracy on a particular feature subset.

The steps involved in feature selection are briefly presented in Algorithm 4.1, where each squirrel will be assigned an initial random state. This state will determine which features are used for the classification task. Each state will contain binary information for each of the features, i.e., If the state contains 1 in i^{th} position, which implies that i^{th} feature must be considered for the classification; otherwise, we do not have to consider this feature. For each squirrel, we then use classification training for the data using only the features corresponding to its state and find the classification accuracy on the test data. The squirrel's fitness will improve as its accuracy improves. The population is sorted based on this fitness value before beginning the primary optimization steps.

The squirrel with the best fitness value in each iteration is found and assumed it to be present at the most optimal food source, i.e., Hickory tree. Next, three squirrels will be present on the Oak trees (Acorn) and all others on the Normal trees. Some squirrels on the Normal trees have already visited the Acorn trees, so they are assumed to move toward the Hickory tree, while others are assumed to move toward the Acorn trees. Then in iteration, each of the squirrels except the squirrel on the Hickory tree will relocate and move towards a more optimal food source. All the squirrels on the Acorn tree will move toward the Hickory tree, while some on the Normal tree will move toward the Acorn tree and some toward the Hickory tree. This relocation process is mentioned later. After relocation, we check for the seasoning monitoring condition, i.e., whether the winter season has approached or not. If this condition is satisfied, then we randomly relocate the squirrels. After all the iterations are completed, we return the

Algorithm 4.1: SSA-FS

Input: Data for classification, Number of squirrels N, flipping rate flip

Output: Features to be selected for maximum accuracy and minimum features (second priority)

Generate N squirrels with random feature states **FS**

```
// Each key is of dimension = Number of features, and value as 0 or 1
/* Start optimisation algorithm                                        */
```

for *squirrel* ∈ **FS do**
  fitness$_{squirrel}$ ← -accuracy(data, feat)
end
Sort the population FS using fitness values.
cur_iteration ← 1
while *cur_iteration* ≤ *max_iteration* **do**
  Hickory ← FS$_1$
  Acorn ← FS$_{[2:4]}$
  Normal_a ← FS$_{[5:27]}$
  Normal_h ← FS$_{[28:50]}$
  for *squirrel* ∈ *Acorn* **do**
    FS$_{squirrel}$ ← Relocate(squirrel, Hickory, flip)
  end
  for *squirrel* ∈ *Normal_a* **do**
    FS$_{squirrel}$ ← Relocate(squirrel,Acorn$_0$, flip)
  end
  for *squirrel* ∈ *Normal_h* **do**
    FS$_{squirrel}$ ← Relocate(squirrel, Hickory, flip)
  end
  l ← Seasoning constant
  if *Seasoning monitoring condition gets satisfied* **then**
    Randomly relocate squirrels on acorn trees
  end
  for *feat* ∈ **FS do**
    fitness$_{feat}$ ← -accuracy(data, feat)
  end
  Sort the population **FS** using fitness values.
end
return feature state with best fitness values(Squirrel on Hickory tree) from **FS**

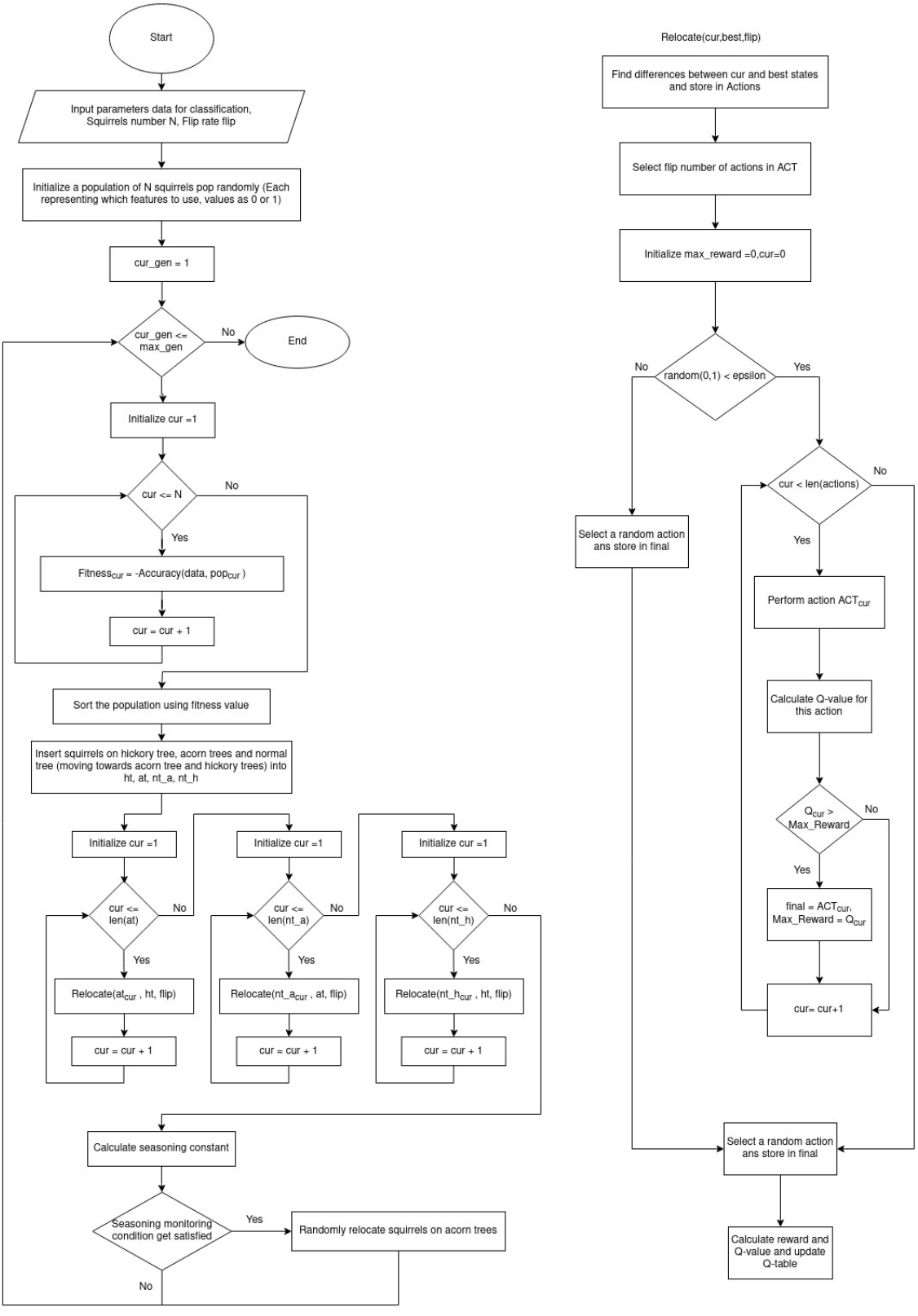


Figure 4.2: Flowchart for QL-SSA-FS

Algorithm 4.2: Relocation and Q-Learning

Input: State **cur** which is to be relocated, State **best** towards which relocation occurs

Output: New feature state fs

```
act ← Difference in features between cur and best states.
action ← Select flip number of actions from act
max_reward ← 0
if random()> epsilon then
    /* Exploitation                                                    */
    for ac ∈ action do
        new_state ← change ac bit in cur
        if accuracy(data, new_state) + Q_table(curr,ac) > max_reward then
            max_reward ← accuracy(data, new_state) + Q_table(curr,ac)
            max_act ← ac
        end
    end
else
    /* Exploration                                                    */
    max_act ← random(action)
end
next_state ← Change max_act bit in cur
acc_old ← accuracy(data, cur)
acc_new ← accuracy(data, next_state)
Calculate reward using current and old accuracy and features
Update Q-table using the Q̃-value found using the equation
```

state of the squirrel present on the hickory tree. The proposed approach's entire process is presented in Figure 4.2, and is briefly detailed in Algorithm 4.1 and Algorithm 4.2.

One of the most crucial steps in the algorithm's convergence is the relocation of squirrels to a tree with a richer food source. QL is a value-based reinforcement learning approach. This indicates that it is based on updating the values of variables through the use of equations. The "$\widetilde{Q}$" stands for quality. This quality refers to the reward received after doing any state-related action. In each step, the agent must learn a policy that

maximizes the total reward.

$$Role\,of\,agent \rightarrow Maximize(\sum_{i=0}^{steps} \gamma^i reward_i) \tag{4.4}$$

The values will be stored in a Q-table, which will be initially empty. Initially, we determine the difference in the bits in both feature states, the current state and the state toward which it must be relocated. The agent (squirrel) interacts with its environment in one of two ways. First, it searches for all possible actions for a given state and chooses the one with the highest reward. This interaction is referred to as exploitation. The second option is to choose an action randomly, which is known as exploring. It is also necessary for the exploration of different states in order to gain better knowledge. $\widetilde{Q}$ for a given state and action can be calculated using

$$\widetilde{Q}(state, action) = (1-\alpha)\widetilde{Q}(state, action) + \alpha(reward + \gamma max(\widetilde{Q}(new\_state))) \tag{4.5}$$

Here α is the learning rate, which may be simply be described as how much we approve the new value in comparison to the old value. γ is a discounting factor used to balance the immediate and future reward as seen in the previous two equations (4.4) and(4.5). The value obtained after performing an action on a state is referred to as the reward. In our case, the reward obtained is:

$$Reward = \begin{bmatrix} Acc(new) & if Acc(new) > Acc(old) \\ Acc(new) - Acc(old) & elif Acc(new) < Acc(old) \\ \frac{Acc(new)}{2} & elif feat(new) < feat(old) \\ -\frac{Acc(new)}{2} & elif feat(new) > feat(old) \end{bmatrix} \tag{4.6}$$

where 'Acc' denotes classification Accuracy, and 'feat' denotes the features. The steps for relocation are described in Algorithm 4.2.

4.4.3 Experimental Results and Discussion

This section presents the experimental results of QL-SSA for optimal feature subset selection. The basic description of the datasets which have been used to validate the efficacy of the proposed QL-SSA feature selection approach is also provided. Comparative results using two different classifiers on 22 datasets are reported, including 2 high dimensional genomic datasets. The reported results were analyzed and discussed briefly. QL-SSA has been used as a wrapper approach, and hence two different classifiers or learning algorithms, KNN and SVM-RBF, have been utilized as evaluators. KNN has some nice characteristics, such as being naturally non-linear, being able to recognize whether distributed data is linear or non-linear, and performing admirably when there are a large number of data points. When we have a small number of points in many dimensions, we can use SVM in a linear or non-linear manner by using a kernel. Since SVM should be able to identify the expected linear separation, it frequently performs very well. SVM is effective in removing outliers because it only uses the most pertinent points to determine a linear separation (support vectors). These two classifiers are also frequently used to solve classification problems involving benchmark data from the real world. All experiments were run on a CPU compute node equipped with a 2.4 GHz Intel-Xeon Skylake 6148 processor and 192 GB of RAM.

4.4.3.1 Datasets Description

To evaluate the performance of the proposed QL-SSA algorithm, 20 publically available datasets from UCI machine learning repository[1] have been utilized. A brief description of datasets is provided in Table 4.1. Datasets with significant variation in terms of the number of classes (2-15), number of features (4-90), and samples (32-1000) have been selected for in-depth analysis.

[1] https://archive.ics.uci.edu/ml/index.php

Table 4.1: Basic description of datasets

Dataset	#Features	#Instances	#Classes	Dataset	#Features	#Instances	#Classes
BC Wisconsin	10	699	2	Iris	4	150	2
Congress	16	435	2	Lung Cancer	56	32	3
Diabetes	8	768	2	Lymphography	18	148	8
Zoo	16	101	7	Movement Libras	90	360	15
German	20	1000	2	Sonar	60	208	2
Glass	9	214	7	Spect	22	267	2
Heart-C	13	303	5	Vehicle	18	846	4
Heart-Statlog	13	270	2	Vowel	10	901	15
Hepatitis	19	155	2	WDBC	30	569	2
Ionosphere	34	351	2	Wine	13	178	3

4.4.3.2 Performance Evaluation and Comparative Analysis

The performance of the proposed model for optimal feature subset selection is analyzed based on accuracy as a performance evaluation metric and the number of selected features. The main objective of the proposed approach is to maximize the accuracy of the model. Prior to evaluation, parameter tuning is done. For this, we have experimented with a different setting and fixed optimal parameters empirically for optimizer and reinforcement learning. Values of α, β, and γ are experimented with in the range of [0,1]. KNN and SVM-RBF are used for the classification of instances. 10-fold cross-validation is performed for the classification task, which is also recommended by [233] for real-world datasets. Once the parameters are tuned, the whole experiments on each dataset are carried out 10 times, and computed statistics are recorded and reported. Best accuracy, average accuracy, and average selected feature subsets are reported for further analysis. Values for parameters of the proposed model are detailed in Table 4.2.

We have also compared the proposed QL-SSA with the baseline approach (SSA) using KNN and SVM-RBF. The obtained statistics are reported in Table 4.3 and Table 4.4, respectively. The following discussion for comparative analysis is based on the average values obtained by both approaches using two different classifiers. From Table

Table 4.2: Parameters used for SSA and QL

SSA		**Q-LEARNING**	
Parameter	Value	Parameter	Value
Population Size	50	α	0.1
No. of generation	10	γ	0.99
Flips	5	ϵ	0.01

4.3, it has been observed that QL-SSA outperforms the baseline approach on 15 out of 20 datasets. While it performs equally on 1 dataset (Iris) and gives a competitive performance on 2 datasets (Wine, Ionosphere). On the other hand, the base model gives better accuracy and the number of selected feature subsets on 2 datasets (Sonar, MovementLibras).

With another classification model (SVM-RBF), it can be concluded from the reported results in Table 4.4 that QL-SSA is a clear winner on 16 out of 20 datasets and performs equivalently on the Iris dataset with 97.33% accuracy and 3 average number of feature set. However, it gives a competitive performance on 3 datasets (Sonar, Movement Libras, and Ionosphere). From the above discussions, we have noticed that the performance measure of QL-SSA on these 3 datasets, Sonar, Movement Libras, and Ionosphere is either competitive or lesser than the baseline. It may be because as the dimension of data increases, a small number of generations is not sufficient for accomplishing the task, so we have performed experiments with more generations (50) on these three datasets and conducted an in-depth analysis. From the obtained results, it has been observed that with the KNN classifier, SSA and QL-SSA both have achieved 95.83% as the best accuracy, while in the cases of average as well as worst accuracy measure QL-SSA gives better results (95.03%) and (94.72%) as compared to SSA (94.13%, 93.33%) respectively.

A similar trend has been shown by QL-SSA with SVM for average and worst accuracy measures (94.61%, 93.89%) in comparison to SSA (93.63%, 91.94%). Similarly, the

Table 4.3: Results using KNN classifier

Dataset	QL-SSA-FS			SSA-FS		
	Best Acc (%)	Avg Acc (%)	Avg Features	Best Acc (%)	Avg Acc (%)	Avg Features
BC Wisconsin	96.29	**96.29**	4	96.29	96.15	4
Congress	97.95	**97.70**	7	97.50	97.05	8
Diabetes	70.65	**70.65**	3	70.65	70.16	4
German	73.00	**72.42**	13	73.00	72.05	11
Glass	75.00	**75.00**	7	75.00	75.00	7
Heart-C	58.71	**58.48**	4	58.39	57.71	5
Heart-StatLog	82.59	**82.52**	5	82.59	80.41	5
Hepatitis	71.33	**70.27**	7	72.67	69.33	6
Ionosphere	93.89	93.28	15	94.72	**93.42**	8
Iris	97.33	97.33	3	97.33	97.33	3
Lung-Cancer	97.50	**97.03**	23	97.50	97.00	22
Lympography	43.33	**41.80**	8	43.33	39.80	7
MovementLibras	89.72	89.36	34	91.94	**90.86**	24
Sonar	93.81	92.86	28	95.62	**94.29**	21
Spect	75.56	**74.67**	11	75.56	73.63	11
Vehicle	74.24	**73.75**	8	74.24	72.24	9
Vowel	99.45	**99.40**	8	99.45	99.38	9
WDBC	94.56	**94.39**	14	94.39	94.30	14
Wine	95.56	94.94	5	95.56	**95.17**	6
Zoo	100.00	**100.00**	10	100.00	99.91	11
Win/Tie/Lose		**15/1/4**			**4/1/15**	

best accuracy achieved on Sonar by QL-SSA with KNN is 98.10%, which is better than SSA (97.14%). While with SVM, both achieved (96.67%). Further, the average and worst accuracy values obtained by QL-SSA with KNN are 96.19%, 94.76% compared to SSA, which are 94.95%, 92.86% respectively. In addition to this, on Movement Libras, a similar trend has been observed for the average and worst case, while for the best case, SSA achieves a higher value of 92.50% compared to QL-SSA 91.49%. It is clearly

Table 4.4: Results using SVM classifier

Dataset	QL-SSA-FS			SSA-FS		
	Best Acc (%)	Avg Acc (%)	Avg Features	Best Acc(%)	Avg Acc(%)	Avg Features
BC Wisconsin	96.29	**96.29**	4	96.29	96.26	4
Congress	97.95	**97.59**	7	97.95	97.30	7
Diabets	70.65	**70.65**	3	70.65	70.00	4
German	73.00	**72.56**	12	74.20	72.04	12
Glass	75.00	**75.00**	6	75.00	74.91	7
Heart-C	58.71	**58.42**	5	58.71	57.55	5
Heart-StatLog	82.22	**81.52**	5	82.59	81.30	5
Hepatitis	72.00	**70.20**	7	72.00	68.67	7
Ionosphere	94.17	93.25	13	95.28	**93.64**	9
Iris	97.33	97.33	3	97.33	97.33	3
Lung-Cancer	97.50	**97.25**	26	97.50	96.75	18
Lympography	43.33	**42.74**	8	42.67	39.80	7
MovementLibras	90.28	89.36	36	92.22	**90.31**	29
Sonar	94.29	93.05	28	96.67	**93.95**	20
Spect	76.30	**74.55**	13	75.56	73.48	11
Vehicle	74.12	**73.35**	8	74.24	72.81	9
Vowel	99.45	**99.45**	8	99.34	99.13	9
WDBC	94.39	**94.37**	16	94.39	94.21	13
Wine	95.00	**94.94**	7	95.56	94.83	7
Zoo	100.00	**100.00**	10	100.00	99.91	11
Win/Tie/Lose		**16/1/3**			**3/1/16**	

visible that QL-SSA outperforms SSA for feature selection problems with stable and consistent results.

So, the overall analysis shows the potential and efficacy of the proposed QL-SSA optimal feature selection and classification. Due to the hybridization with QL, it adapts

the characteristics of exploration point insensitive and learns from the experience of the previous agent for reaching global optima. It has been observed during experimental analysis that in many cases, average and best values achieved by QL-SSA are almost similar or have a minor difference. It shows that the proposal of a hybrid optimizer is consistent and stable as compared to the baseline approach. Results show that it is capable of maintaining a good balance between intensification and diversification, which is an important characteristic of an efficient optimizer.

4.4.3.3 Comparison of SSA and QL-SSA for Optimal Gene Selection

Experimental analysis have also been performed on two large-scale genomic datasets [234]. The results obtained are reported in Table 4.5. It has been observed that QL-SSA is more stable than SSA on large-scale datasets, as previously reported with various benchmark datasets. QL-SSA outperformed SSA on GDS1615 with an average accuracy of 81.65% by using nearly 5794 features in 3665.29 seconds, while SSA achieves an average accuracy of 80.93% by using 6150 features in 6459.52 seconds. However, on another dataset (GDS531), QL-SSA achieved an average accuracy of 92.55% by selecting only 710 features in 4172 seconds, whereas SSA achieved an average accuracy of 95.18% by selecting 4276 features in 9293 seconds. Overall, the analysis demonstrates QL-SSA stability and adaptability. It has also been observed that QL-SSA is still computationally expensive, prompting us to propose another variant of modern hybrid with exceptional exploitation capability and adaptive characteristics with solution diversity.

Table 4.5: Comparative analysis of QL-SSA over SSA with SVM classifier

Method	Dataset	Avg Acc (%)	Best Acc (%)	Avg Features	Avg Time (sec)
SSA	GDS1615	80.93	83.25	6150.30	6459.52
	GDS531	95.18	97.92	4276.80	9293.41
QL-SSA	GDS1615	81.65	83.85	5794.50	3665.29
	GDS531	92.55	95.78	710.40	4172.32

4.5 Proposed Quantum-assisted Chaotic SSA

Before proceeding, the essential preliminaries of QCSSA (chaos theory and quantum computing) will be discussed in this section for clear understanding. Following that, chaotic and quantum chaotic variants will be briefly explained. Just after that, their application to optimal gene selection was investigated. Finally, extensive experimental analyses of benchmark functions and genomic datasets are being discussed.

4.5.1 Chaos Theory

Chaos is a nonlinear phenomenon characterized by unpredictable random behavior that often occurs in deterministic nonlinear dynamic systems under deterministic conditions. A chaotic map generated by a deterministic function is widely utilized in several application areas and algorithms to improve efficiency. Due to characteristics such as unpredictable random behavior, regularity, periodicity, dynamic nature, and sensitivity to initial values, chaos has been frequently used in the original evolutionary algorithms to improve performance. The dynamic characteristics and ergodicity of a chaotic system cause it to change randomly and also enable it to search different landscapes in the search space by moving through each state in the search space with enhanced searching speed. Premature convergence and local optima entrapment, particularly in the case of multi-modal and non-convex optimization problems, are the major challenges faced by the optimizer. To be efficient, optimizers must have a good balance of global exploration and local exploitation, which is a difficult task because optimizers with better exploration capability typically have poorer local exploitation and vice versa. To address this issue, chaos theory is best suited due to its inherent characteristics, which not only speed up the search process but also improve the variety of pattern movement. Various optimizers have embedded this concept to improve their performance [235, 236].

In general, single dimension chaotic maps are the simplest system capable of generating chaotic motion. The Lyapunov exponent is a significant factor that influences the

efficiency of global optimization. This exponent can determine whether the ergodicity of a chaotic sequence is better or worse. A chaotic sequence with a higher Lyapunov exponent has a higher chaotic degree, which implies a faster searching speed [237]. For a given chaotic one-dimensional map, $x_{n+1} = g(x_n)$, where x_n represents the chaotic variable and $n = 0, 1, 2, ., .,$ the Lyapunov exponent L_e can be expressed as:

$$L_e = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \ln |g^{'}(x_i)| \tag{4.7}$$

There are various chaotic maps; for this work, we used three chaotic maps, namely the sine map, the sinusoidal map, and the neuron map, to generate chaotic sequences [238]. It is further applied in place of random variables to improve efficiency. Here, we will introduce three well-known 1-D chaotic maps:

1. **Sine map:** The Sine map is part of a unimodal map, which is depicted below:

$$x_{r+1} = \frac{a}{4} \sin(\pi x_r) \; x \in (0, 1) \; 0 < a \leq 4 \tag{4.8}$$

For $a = 4$, the Lyapunov exponent L_e is computed by using Eq. (4.7) is 0.6882.

2. **Sinusoidal map:** The Sinusoidal map is presented as follows:

$$x_{r+1} = a x_r^2 \sin(\pi x_r) \;\; x \in (0, 1) \tag{4.9}$$

Here, $a = 2.3$ acts as a control parameter that estimates the chaotic behavior of the dynamic system.

3. **Neuron map** A chaotic map with nonlinear feedback is referred to as a neuron map. It employs the hyperbolic tangent function and the exponential function to create the chaotic map. It is expressed as:

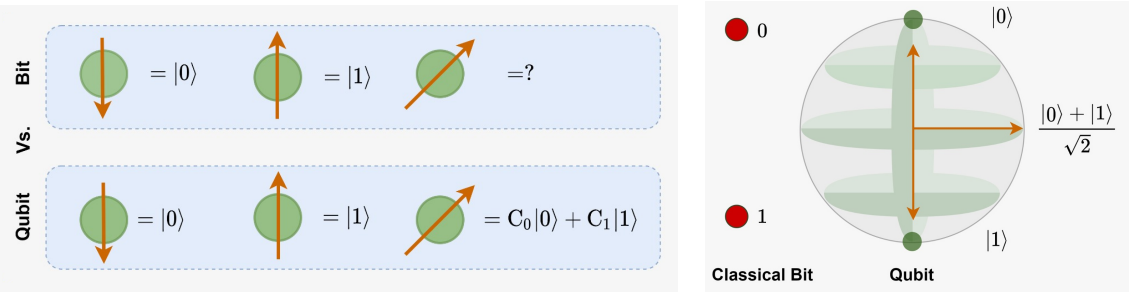
$$x_{r+1} = \zeta - 2 \tanh(\psi) e^{-3x_r^2} \tag{4.10}$$

where ζ represents the attenuation factor that lies in $(0 \leq \zeta \leq 1)$, and ψ denotes the proportionality factor. For further experiments, we have considered the values $\zeta = 0.5$ and $\psi = 5$. It generates the chaotic sequences x in the range of $(-1.5, 0.5)$.

4.5.2 Quantum Computation

Quantum computing emerged as a new area of computer science that is based on quantum mechanics principles. Richard Feynman introduced the concept of quantum computing in the early 1980's [239]. This was further formalized in the early 1990s with two remarkable achievements: a quantum algorithm for large number factorization by Peter Shor in 1994 [240], and a fast quantum search algorithm for searching unstructured databases by Lov Grover at Bell Labs in 1996 [241]. These works demonstrate that quantum computers are more powerful than classical computers when it comes to solving specialized problems. In addition, combining quantum computing and evolutionary algorithms to improve the behavior of traditional evolutionary algorithms was reported in late 1990 [242, 243].

Han and Kim[244] developed the first quantum genetic algorithm (QGA). After this proposal, researchers are attracted to this new approach due to its ability to process multiple states simultaneously in parallel. Following that, a variety of quantum-inspired NIA are proposed for classical computers and applied to various applications [245, 246]. Previously, it moved at a slow pace due to a lack of quantum literature, but nowadays, designing quantum-inspired algorithms has received a lot of attention from the research communities. Quantum computing has inherent probabilistic behavior, which allows NIA to gain better population diversity while also exploring the global search space effectively. The basic difference between a classical bit and a qubit is shown in Figure 4.3. Thus, in order to meet future demands, we have incorporated the concepts of a qubit, superposition theorem, and quantum gate into our proposal.



(a) (b)

Figure 4.3: Pictorial representation of (a) classical bit vs. qubit (b) bloch sphere of a qubit state

4.5.2.1 Qubit

The smallest information unit stored in a two-state quantum computer [247] is referred to as a quantum bit or qubit [244], which can be represented by a pair of complex numbers (α, β). It can be in state 0, 1, or the superposition of both states. The state of qubit Ψ is described by using the bracket notation as:

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{4.11}$$

The complex numbers α and β represent the probability amplitudes of the 0 and 1 states, respectively. The pair of complex numbers satisfy the conditions such that:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{4.12}$$

So, a qubit can be written as:

$$q = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{4.13}$$

NIA with Q-bit representation has superior population diversity features than other representations because it can express linear superposition of states probabilistically.

4.5.2.2 Quantum gate (Q-gate)

A Q-gate is a reversible gate and is implemented on Hilbert space by utilizing different linear and unitary operations. It is represented by a unitary operator ($\bigcup$) that acts on the qubit basis states. We have applied the Quantum rotation gate, which was originally proposed by [244] in our proposal due to its capability and effectiveness. It is designed in response to a practical problem and used to change the state of a Q-bit. It is usually written as:

$$U(\phi_i) = \begin{bmatrix} \cos(\phi_i) & -\sin(\phi_i) \\ \sin(\phi_i) & \cos(\phi_i) \end{bmatrix} \tag{4.14}$$

where ϕ_i denotes the rotation angle. Further, the qubit update process can be presented as follows:

$$\begin{bmatrix} \alpha^{i(t+1)} \\ \beta^{i(t+1)} \end{bmatrix} = U(\phi_i) \begin{bmatrix} \alpha^{i(t)} \\ \beta^{i(t)} \end{bmatrix} \tag{4.15}$$

where $i = 1, 2, .., n$, and $U(\phi_i)$ is a function of $\phi_i = s(\alpha^i, \beta^i).\Delta\phi_i$. $s(.)$ defines the sign of ϕ_i, which estimates the direction, and $\Delta\phi_i$ denotes the magnitude of the rotation angle. The angle parameter for rotation is selected based on adaptive search strategy [244].

4.5.3 Chaotic Squirrel Search algorithm (CSSA)

The proposed CSSA significantly improved the SSA algorithm by utilizing chaos theory. To solve the optimal convergence problem, we used three different well-known chaotic maps, as explained in Section 4.5.1, to generate random positions or new locations for squirrels in a specified condition. It follows the steps of the normal SSA.

The mathematical model for the chaotic squirrel search is explained as follows:

In the beginning, the location of all flying squirrels (FS) are initialized randomly with the help of k dimensional vector since they can glide from 1-dimensional to hyper-

dimensional search space and change their locations. We have considered the N_p number of "FS" in a forest, and the location of all flying squirrels can be represented by a matrix with an order of $N_p \times k$:

$$FS = [FS_{j,k}], \forall j \in 1, 2, \ldots\ldots N_p \, and \, \forall k \in 1, 2, \ldots\ldots, k \tag{4.16}$$

Here, FS_{jk} denotes the j^{th} squirrel's, and k^{th} dimension of its position vector. The initial location of each squirrel is generated by using uniform distribution $U(0, 1)$ as shown in the following equation:

$$FS_j = FS^{low} + U(0, 1) \times (FS^{up} – FS^{low}) \tag{4.17}$$

where FS^{low} and FS^{up} are the lower and upper bounds of k^{th} dimension of the j^{th} squirrel, respectively.

1. **Fitness evaluation and random selection:** The fitness value defines the quality of the food source sought by each flying squirrel with its corresponding location vector and is calculated by substituting the decision variables into a user-defined or standard objective (fitness) function. The fitness value fit can be expressed as:

$$fit = (f_1, f_2, f_3, \ldots\ldots\ldots f_{N_p}) \tag{4.18}$$

$$fit = (f_j) \; \forall j \in 1, 2, \ldots.., N_p \tag{4.19}$$

Here fitness value of jth flying squirrel can be computed as:

$$f_j = f_j(FS_{j1}, FS_{j2}, FS_{j3}, ..., FS_{jk}) \tag{4.20}$$

Then, the fitness value associated with each squirrel is stored and sorted in ascending order in an array comprised of fitness value and corresponding index.

Following that, the sorted fitness values are used to categorize three trees in such a way that the flying squirrel with the minimum fitness is treated as the best food source, which is the hickory tree (FS_{Ht}), the next three values can be considered as suboptimal solutions or food sources- acorn tree (FS_{At}), and the remaining squirrels are assumed to be on normal tree (FS_{Nt}). It can be expressed mathematically as:

$$sort(fit) = [sorted\_fit, sorted\_index] \tag{4.21}$$

Further, on the basis of a sorted list, trees are categorized as follows:

$$FS_{Ht} = FS_{sorted\_index(1)} \tag{4.22}$$

$$FS_{At}(1:3) = FS_{sorted\_index(2:4)} \tag{4.23}$$

$$FS_{Nt}(1:N_p-4) = FS_{sorted\_index(5:N_p)} \tag{4.24}$$

The dynamic foraging behavior of flying squirrels is modeled by assuming that squirrels on acorn trees have met their daily energy requirements and moved to the optimal food source; "hickory tree." However, squirrels that are on normal trees, on the other hand, are selected at random based on the assumption that some of these squirrels have met their daily energy needs and have moved towards the hickory tree, while the remaining squirrels proceed to fulfill their daily need towards acorn tree. To simulate the natural phenomenon where the presence of a predator affects the foraging process, we set the presence of predator probability (P_{pd}) to 0.1 for our problem. It is used to keep squirrels' locations updated throughout the process.

2. **New locations generation through gliding:** During the dynamic foraging process, FS searches for food sources efficiently in the absence of a predator while using a short random walk to search for a hiding position. There are three

possible movements: $At \Rightarrow Ht$, $Nt \Rightarrow At$, and $Nt \Rightarrow Ht$. The complete foraging behavior and individual locations generation can be written mathematically by formulating the following three possibilities:

Case 1: Here, FS on At move towards Ht, the new positions of squirrels are computed as:

$$(FS)_{At}^{gen+1} = \begin{cases} FS_{At}^{gen} + d_g \times G_c \times (FS_{Ht}^{gen} - FS_{At}^{gen}) & r_1 \geq P_{pd} \\ random\ location\, using\, chaotic\, map & otherwise \end{cases} \tag{4.25}$$

Case 2: FS on normal tree FS_{Nt} glide towards the acorn tree for daily energy needs $FS_{Nt} \rightarrow FS_{At}$. Therefore, new positions for squirrels can be obtained as:

$$(FS)_{Nt}^{gen+1} = \begin{cases} FS_{Nt}^{gen} + d_g \times G_c \times (FS_{At}^{gen} - FS_{Nt}^{gen}) & r_2 \geq P_{pd} \\ random\ location\, using\, chaotic\, map & otherwise \end{cases} \tag{4.26}$$

Case 3: Few FS on normal trees may move to hickory trees because they may fulfill their daily needs and are searching for the best food source, "hickory nuts ." So the movements are made from $FS_{Nt} \rightarrow FS_{Ht}$ and their new locations are updated accordingly as:

$$(FS)_{Nt}^{gen+1} = \begin{cases} FS_{Nt}^{gen} + d_g \times G_c \times (FS_{Ht}^{gen} - FS_{Nt}^{gen}) & r_3 \geq P_{pd} \\ random\ location\, using\, chaotic\, map & otherwise \end{cases} \tag{4.27}$$

The above formulations from equation (4.25) to equation (4.27) have three random numbers (r_1, r_2, r_3), lies in the range of [0,1]. d_g denotes gliding distance, which is to be considered as $9 - 20$ meters, while G_c is a gliding constant that balances the exploration and exploitation, which is an important factor for significant performance. We considered a value of 1.9 for our work, which is similar to the original work. The other variable, *gen*, represents the current generation or

iteration, and $gen + 1$ denotes the subsequent iteration or generation. To avoid the large perturbation introduced by $d_g \in [9, 20]m$ in equation (4.25) to equation (4.27), which may result in unsatisfactory performance, a scaling factor $(s_f) = 18$ is incorporated as a divisor of d_g and hence $d_g \in [0.5, 1.11]$, resulting in acceptable performance.

3. **Examine seasonal monitoring condition** Seasonal variations have a significant impact on the foraging behavior of flying squirrels. Due to the change in climate, they become less active in the winter. To incorporate the realistic condition, a term seasonal constant s^c is introduced in standard SSA to avoid local entrapment, and we also follow the same. The following equations first compute the seasonal constant s^c and its minimum value s^{min}:

$$(s^c)^{gen} = \sqrt{\sum_{i=1}^{k} (FS_{At,i}^{gen} - FS_{Ht,i})^2} \tag{4.28}$$

where $gen = 1, 2, 3$.

Further its minimum value s^{min} is computed as

$$s^{cmin} = \frac{10E - 6}{(365)^{\frac{gen}{maxgen/2.5}}} \tag{4.29}$$

Here, the higher the value of s^{cmin}, the more the exploration capability of the proposal, and the lower the value, the better the optimizer's exploitation capability. The gliding constant is used to find a balance between these two phases. Seasonal monitoring condition that implies $(s^c)^{gen} < s^{cmin}$ is checked; if the value is found to be true, it is assumed that winter is over, and FS that have lost their ability to search for optimal food sources, but still survived will relocate and move in a different direction for optimal food search.

4. **Relocate randomly and termination criterion** Their positions are randomly

relocated as follows:

$$FS_{Nt} = FS^{low} + Lévy(k) \times (FS^{up} - FS^{low}) \tag{4.30}$$

where *Lévy* flight is used to enhance the global exploration capability of the algorithm and assumed to be a powerful mathematical tool for improving different optimizers [248, 249]. The mathematical function for *Lévy* flight is shown as:

$$Lévy(u) = 0.01 \times \frac{r_a \times \sigma}{|r_b|^{\frac{1}{\beta}}} \tag{4.31}$$

where $\beta = 1.5$ and (r_a, r_b) are two random numbers generated from a normal distribution in the range of [0,1]. While σ is obtained from the following equation:

$$\sigma = \left(\frac{\Gamma(1+\beta) \times \sin(\pi.\beta/2)}{\Gamma(\frac{1+\beta}{2}) \times \beta \times 2^{\frac{(\beta-1)}{2}}} \right) \tag{4.32}$$

Here, $\Gamma(u) = (u-1)!$. Further, *maxgen* is used as a termination criterion. However, depending on the specific problem, we can also consider some threshold values as a stopping criterion. The entire process of checking seasonal monitoring conditions, updating locations, and creating new locations continues until the termination condition is satisfied.

4.5.4 Quantum-assisted Chaotic SSA (QCSSA)

Quantum variants have the same characteristics as their non-quantum counterparts. It also adheres to basic quantum computation concepts in population initialization, population update for the next generation, and so on. Hadamard gate (H-gate), one of the most popular and useful Q-gates, is used to generate quantum superposition vectors

at random. The following equation represents the H-gate:

$$H_{gate} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{4.33}$$

The steps we have followed in the proposal are described in Algorithm 4.3 and 4.4.

1. Population initialization: In contrast to chaotic SSA, here we initialized the quantum population Q. Quantum population Q is represented as

$$Q = \{Q_1, Q_2, ..., Q_{N_p}\} \tag{4.34}$$

Each individual of the population, Q_i of quantum superposition vectors, is defined as an array of j qubits that represent a quantum system $|\Psi\rangle_i$ with 2^j states. Similarly, quantum registers $|\Psi_n\rangle$ is a set of n qubits that can represent 2^n different states simultaneously. Each individual chromosome consists of a string of n qubits, which is presented by the following equation.

$$q = [q_1, q_2, ..., q_n] \Rightarrow q = \begin{bmatrix} \alpha^1 & \alpha^2 & \alpha^3 & .. & \alpha^n \\ \beta^1 & \beta^2 & \beta^3 & .. & \beta^n \end{bmatrix} \tag{4.35}$$

where each bit satisfies the given relation

$$|\alpha^j|^2 + |\beta^j|^2 = 1, \forall j \in \{1, 2, ...n\} \tag{4.36}$$

We assigned equal probabilities to the amplitudes of all qubits in the chromosomes. It is accomplished by multiplying the Hadamard matrix shown in equation (4.33) with $|0\rangle$ vector. The operation is denoted as:

Algorithm 4.3: QCSSA

Input : PopSize(N_p), Function, totalAcorntree(N_a), maxgen

Output: $F_{best} \equiv FS_{Ht}$ (F_{best} is the best fitness value)

Initialise Quantum Superposition Vector **Q**
Measure Every Single Individual **C**
Generate classical population and sort the population based on fitness value
while $F_{best} \neq$ *Optimal Value or Max-Gen* **do**
    H_t= bestfitness
    Acorn=nextbest N_a fitness
    $N \rightarrow H_t$ = half of remaining population
    $N \rightarrow A_t$ = rest population
    for i = *0 to* N_a **do**
        rotation(Q_i, H_t)
        F_{Q_i}=NEWLOCATION (F_{Q_i}, F_{Ht})
    end
    for i *in* $N - A_t$ **do**
        rotation($Q_i, Acorn$)
        F_{Q_i}=NEWLOCATION (F_{Q_i}, F_{Acorn_i})
    end
    for i *in* $N - H_i$ **do**
        rotation($Q_i, N - H_t$)
        F_{Q_i}=NEWLOCATION (F_{Q_i}, F_{N-H_i})
    end
    if *Seasoning monitoring condition gets satisfied* **then**
        Randomly relocate squirrels on acorn trees
    end
    Measure every individual
    Calculate the fitness of the population
    Sort the fitness
end
return F_{best} *i.e.* H_t

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{4.37}$$

By using these equations, we have initialized the population in the proposed

Algorithm 4.4: NEWLOCATION

Input : Current individual, best individual
Output: Updated location
if *random no. > 0.1* **then**
  Update with standard method
end
Update with chaotic function

quantum chaotic version.

2. Classical population generation: This step entails generating a classical population on the basis of quantum population Q. It means that we get classical chromosomes from a quantum superposition vector, which is presented as a binary value. It can be obtained as:

$$X_j = \begin{cases} 1 & if\ rand[0,1] < |\beta_j|^2 \\ 0 & otherwise \end{cases} \tag{4.38}$$

Further, the classical population $P(t)$ is generated using the aforementioned classical chromosomes or bits of an array obtained by measuring the qubit states in the quantum population $Q(t)$. Thus, $P(t)$ is a set of binary vectors or chromosomes.

3. Population update: This step mainly influences the convergence of the proposal. We use the rotation quantum gate $U(\phi)$ on the quantum superposition vectors to update the quantum population. This step is accomplished by using equations (4.14) and (4.15). We followed the similar steps for relocation explained in the CSSA by using equations (4.25) to (4.27) and equation (4.30).

We have used the same termination criterion as in the proposed chaotic version.

4.5.5 Application to Optimal Gene Selection and Classification

The application of feature selection is essential in genomic datasets where the number of features is typically very huge, but the number of samples is small. Reduction in input features will aid ML algorithms, as they suffer in high-dimension space. Apart from improved accuracy, the training time is also reduced when feature selection is used with ML algorithms. Hence, we apply QCSSA to nine different genomic datasets with varying input features and classes before the ML algorithm uses them.

The feature selection process starts with preprocessing the datasets to ensure there are no redundant features and missing values are present in the datasets. The dataset will contain relatively fewer features than the original dataset due to preprocessing stage. The details about preprocessing are delineated in Section 4.5.6. Following preprocessing, the immediate step is to split data into three parts: training, validation, and testing. In Table 4.6, we present the statistics about the datasets before and after preprocessing stage. In the implementation of QCSSA, there are 'n' squirrels where the best squirrel is on the hickory tree, and the next three are on acorn trees, and the rest of the squirrels are on normal trees. Here, a tree represents a squirrel's position and is represented as a vector of size equal to the number of features of the input dataset. Initially, vectors for all the squirrels are initialized randomly with data values within a bounded set [0.0, 1.0]. In feature selection problems, the method's accuracy is essential. Still, the number of features selected by the method is critical as the run time complexity of ML algorithms is positively correlated with the number of input features. Thus, we design the fitness function shown in equation (4.39), where it takes into account both accuracy as well as the number of features selected by the optimization algorithm.

$$Fitness = \lambda * (1 - valacc) + (1 - \lambda) * \frac{n_s}{N_T} \tag{4.39}$$

Here, n_s and N_T represent a count of selected genes and the total number of genes,

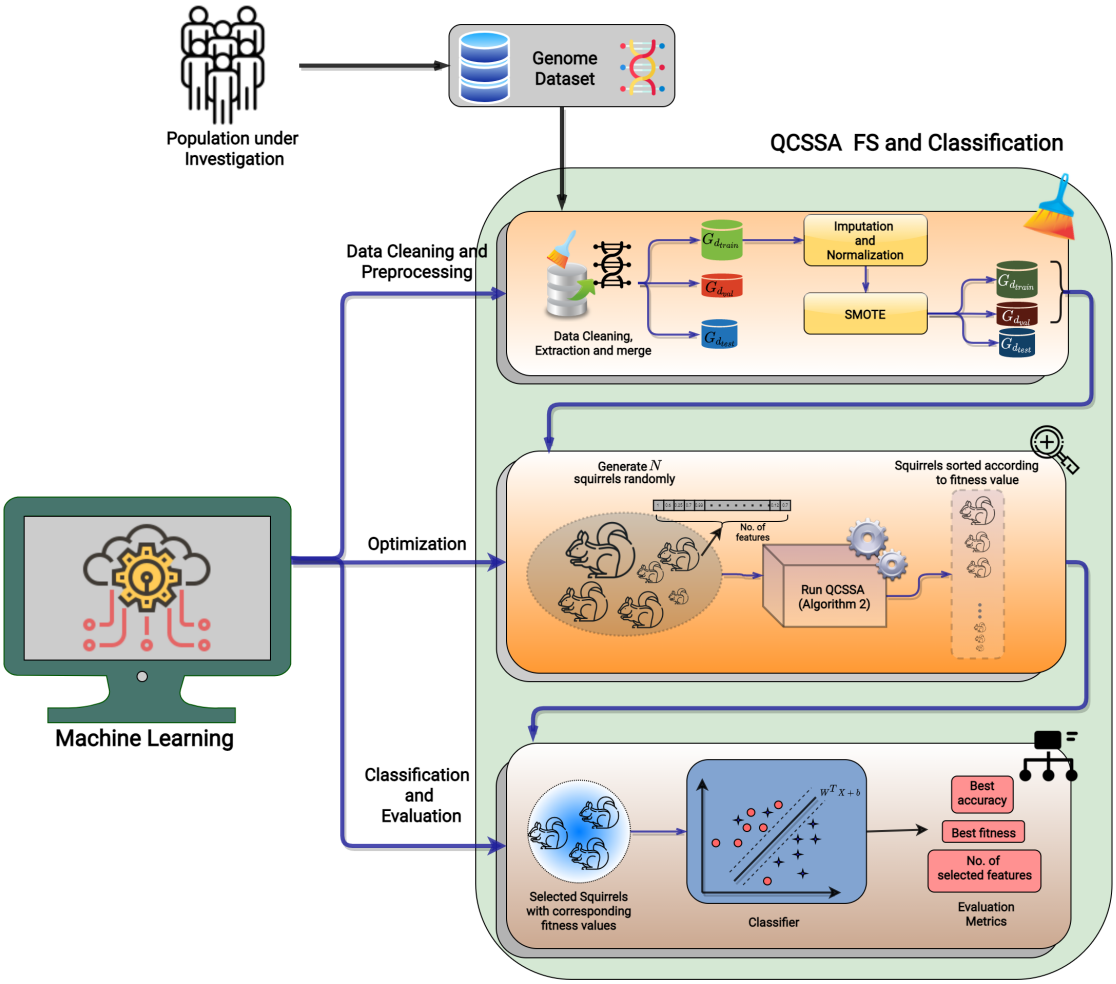


Figure 4.4: Overall framework for QCSSA feature selection and classification

respectively.

The squirrels are arranged in ascending order based on their fitness values. The squirrel with the lowest fitness value is better as per equation (4.22). Further, the execution of QCSSA is depicted in Figure 4.4 and detailed in Algorithm 4.5. At the end of the optimization process, the positions of squirrels are converted to binary vectors as we need to select only the necessary features. We use the standard binarization method used in feature selection literature, which involves using threshold $Th = 0.5$. If the value in the vector is above Th, then the feature corresponding to that column is retained, while columns with a value below Th are dropped.

Finally, the selected set of features is utilized by an ML algorithm, which in our case

Algorithm 4.5: QCSSA for genome feature selection + classification

Input: Genome dataset (G_d), α, Max no. of generations(g_{max}), No. of squirrels (N), No. of acorn trees (nt_{acorn}), classifier (C)

Output: Best fitness (Fit_{best}), best accuracy (Acc_{best}), No. of selected features ($n_{selected}$), best AUC (Auc_{best})

1 Process G_d to extract genes, chromosomes, merge repetitive genes

2 Split G_d into $G_{d_{train}}$, $G_{d_{val}}$, $G_{d_{test}}$

3 Using only G_{dtrain} impute missing values, normalize and handle data imbalance using SMOTE for $G_{d_{val}}$, $G_{d_{test}}$

/* Start optimisation algorithm */

4 Run the QCSSA algorithm and update FS

5 **for** *squirrel* $\in$ **FS do**

6 fitness$_{squirrel}$, test acc, no. of features $\leftarrow$ Evaluate($G_{d_{train}}$, $G_{d_{val}}$,$G_{d_{test}}$, C, squirrel)

7 **end**

8 **return** Best fitness (Fit_{best}), best test accuracy ($TestAcc_{best}$), No. of selected features ($n_{selected}$), best AUC ($TestAuc_{best}$)

is SVM with RBF kernel. The hyperparameters C and γ of the classification model are also optimized in wrapper-based feature selection.

4.5.6 Experimental Results and Discussion

In this section, we present the exhaustive experimental results of the proposed chaotic variants, quantum variants, and quantum chaotic variants of the SSA on twenty-six classical standard benchmark functions [250]. These benchmarks include problems of different modality and complexity ranging from low dimension to high dimension, uni-modal to multi-modal, separable to non-separable, continuous to discontinuous, linear to nonlinear, and convex to non-convex problems. In addition, we presented a comparative analysis with the original SSA and other existing state-of-the-art optimizers.

Furthermore, we addressed one of the most challenging real-world problems for optimal gene selection in high-dimensional genomic datasets. The analyses presented here are based on rigorous experiments conducted during this study. We further showcase

the potency of QCSSA by comparing it to other standard algorithms used in feature selection literature. For a fair comparison, all of the experiments are carried out with identical experimental settings.

The lack of consistency is a significant weakness of these feature selection algorithms. Hence, we repeated the experiments ten times and reported the average values as well as the standard deviation (SD) for comparison. We have also performed the statistical test for the significance analysis of the proposal. All experiments were run on a CPU compute node equipped with a 2.4 GHz Intel-Xeon Skylake 6148 processor and 192 GB of RAM.

4.5.6.1 Dataset description and preprocessing

To demonstrate the efficacy of QCSSA, we select a wide variety of datasets from Gene Expression Omnibus (GEO) [234]. The selected dataset have m features and n examples where $m >> n$. The high dimensionality of these datasets, as well as the cost of gathering new examples for training, make the problem harder. The train, test, and validation split distributions of the selected datasets are shown in Table 4.6. The publicly available GEO data are preprocessed using the steps outlined in work [251].

In Table 4.6, we report the number of features before and after preprocessing. In Figure 4.4, we show the sequence of steps that the dataset goes through before being used by the ML algorithm. One of the crucial steps is the data cleaning stage, where we ensure that the given datasets have no obvious errors and missing values. As the utility of the datasets by ML algorithms depend on the number of observations per class, our preliminary analysis clearly demonstrated the importance of using data augmentation techniques such as Synthetic Minority Oversampling Technique (SMOTE) [252]. Hence, we augment the dataset using SMOTE. Prior to the data standardization step, we split the main dataset G_d into three disjoint datasets: train ($G_{d_{train}}$), val ($G_{d_{val}}$), test ($G_{d_{test}}$). We only use the data statistics from $G_{d_{train}}$ for standardization.

4.5.6.2 Comparison of proposals on benchmark functions

We conducted extensive experimentation to evaluate the accuracy and effectiveness of the proposals for solving benchmark functions containing a wide range of optimization problems. For unbiased analysis, we considered $N_P = 50$ number of squirrels, $P_{pd} = 0.1$ as specified in the original SSA. We set *maxgen* to 1000 due to the level of complexity of test functions. Table 4.7 details the parameters used by the optimizers for comparative analysis. The experiments for each optimizer are carried out with 30 independent trials, and the analysis of the statistical results are presented in Table 4.8 and Table 4.9.

We proposed and implemented three chaotic variants, CSSA1 (with sinusoidal map), CSSA2 (with sine map), and CSSA3 (with neuron map), as well as its quantum variants, QCSSA1, QCSSA2, and QCSSA3, along with QSSA. Thus, the obtained results from these proposals are compared with the original SSA proposed in 2019 to check its efficacy, although the efficacy of SSA is already compared with other existing optimizers in the original SSA work. For comparative analysis, we also considered three popular

Table 4.6: Dataset specifications

Dataset	Title	Original Features	Cleaned Features	Samples	Classes	Class Distribution			
						0	1	2	3
GDS1615	Ulcerative colitis and Crohn's disease comparison: peripheral blood mononuclear cells	22282	13650	127	3	42	26	58	-
GDS531	Colon epithelial biopsies of ulcerative colitis patients	12625	9391	173	2	36	136	-	-
GDS968	Radiation therapy toxicity association with abnormal transcriptional response to DNA damage	12625	9117	171	4	45	45	42	38
GDS1962	Multiple myeloma and bone lesions	54675	29185	180	4	23	26	80	50
GDS2545	Metastatic prostate cancer (HG-U95A)	12625	9391	171	4	18	62	65	25
GDS2546	Glioma-derived stem cell factor effect on angiogenesis in the brain	12620	9583	167	4	17	59	65	25
GDS2547	Tobacco smoke effect on maternal and fetal cells	12646	9370	164	4	17	58	63	25
GDS3268	Metastatic prostate cancer (HG-U95B)	44290	29916	202	2	72	129	-	-
GDS3929	Metastatic prostate cancer (HG-U95C)	24526	19335	183	2	55	127	-	-

Table 4.7: Parameters used with various algorithms

Approach	Parameters
PSO	$c_1 = 2$, $c_2 = 2$,$Vmax = 6$ and inertia weight=0.9
GWO	a=2
WOA	The parameter of spiral's shape is $b = 1$, the boundaries are $l_b = 1$ and $l_u = -1$
SSA	The boundaries are $l_b = -1$ and $u_b = 1$
CSSA (Sinusoid)	$N_p = 50$, $N_{fs} = 4$,$P_{pd} = 0.1, G_c = 1.9$
CSSA (Sin Map)	$N_p = 50$, $N_{fs} = 4$,$P_{pd} = 0.1, G_c = 1.9$
CSSA (Neuron)	$N_p = 50$, $N_{fs} = 4$,$P_{pd} = 0.1, G_c = 1.9$
QCSSA (Sinusoid)	$N_p = 50$, $N_{fs} = 4$,$P_{pd} = 0.1, G_c = 1.9$, $\phi = 0.025\pi$
QCSSA (Sin Map)	$N_p = 50$, $N_{fs} = 4$,$P_{pd} = 0.1, G_c = 1.9$, $\phi = 0.025\pi$
QCSSA (Neuron)	$N_p = 50$, $N_{fs} = 4$,$P_{pd} = 0.1, G_c = 1.9$, $\phi = 0.025\pi$

and highly potential optimizers: PSO [253], WOA [254], and GWO [255].

- **Unimodal benchmark functions:** First, we present fitness values of test functions as a statistic value by considering unimodal functions consisting of separable and non-separable functions with different complexities and dimensions ranging from 2 to 30. Table 4.8 shows the best, average, and worst fitness values, as well as the standard deviation, for unimodal functions, where TF_1 to TF_4 are unimodal separable benchmark functions with 30 dimensions and global optima at 0. Besides that, TF_5 to TF_12 are unimodal and non-separable benchmarks with dimensions ranging from 2 to 30. The global optima for TF_6 is -1, while the remaining have 0 as global optima. The results clearly show that either of the proposals outperforms or is on par with the original SSA. On TF_6, however, they all perform equally well. Figure 4.5 depicts a 3-d surface plot, contour plot of QCSSA (neuron map), and corresponding convergence plot at log scale of fitness function vs. generations of different optimizers on four representative test functions TF_1, TF_5, TF_6 and TF_12. It is clear that chaotic versions and quantum versions exhibit good convergence when compared to the original SSA and other state-of-the-art methods.

Table 4.8: Statistical results analysis on unimodal benchmarks

Function name			Normal and its Proposed Chaotic Variants				Proposed Quantum and its Chaotic Variants			
			SSA	**CSSA1**	**CSSA2**	**CSSA3**	**QSSA**	**QCSSA1**	**QCSSA2**	**QCSSA3**
Step	TF_1	best	9.03E-09	2.76E-10	4.39E-10	7.60E-11	2.29E-08	2.67E-10	6.18E-10	**7.50E-11**
		worst	1.60E-06	5.60E-02	8.27E-04	4.11E-04	9.42E-01	8.83E-01	5.26E-01	6.02E-03
		avg	1.67E-07	3.73E-03	2.76E-05	8.24E-05	2.93E-01	6.09E-02	6.80E-02	9.69E-04
		std	2.94E-07	1.42E-02	1.51E-04	1.28E-04	3.18E-01	1.69E-01	1.63E-01	1.86E-03
Sphere	TF_2	best	3.16E-06	**1.83E-23**	2.00E-23	-1.11	7.62E-05	2.11E-23	3.22E-23	2.88E-09
		worst	1.55E-04	1.19E-22	1.95E-22	-1.11	8.39E+02	7.78E-03	1.75E-22	7.29E-01
		avg	4.19E-05	5.38E-23	9.07E-23	-1.11	2.19E+02	2.91E-04	9.60E-23	4.95E-02
		std	3.89E-05	2.34E-23	4.06E-23	-1.11	2.64E+02	1.41E-03	4.11E-23	1.43E-01
Sum Squares	TF_3	best	1.09E-06	**1.93E-22**	2.59E-22	2.29E-08	4.28E-06	3.16E-22	3.99E-22	1.15E-07
		worst	4.87E-05	9.84E-22	1.73E-21	3.99E-02	7.76E+01	5.64E-02	2.34E-21	2.24E-06
		avg	1.05E-05	4.12E-22	8.58E-22	1.36E-03	1.38E+01	2.53E-03	1.16E-21	1.10E-06
		std	1.08E-05	2.19E-22	4.05E-22	7.28E-03	2.01E+01	1.03E-02	5.95E-22	6.85E-07
Quartic	TF_4	best	5.31E-10	2.14E-23	3.17E-23	2.76E-09	1.44E-09	**1.61E-23**	3.60E-23	9.98E-09
		worst	4.07E-08	9.25E-23	1.63E-22	2.36E-01	4.83E-01	1.86E-08	2.24E-22	3.97E-01
		avg	6.28E-09	4.96E-23	7.75E-23	7.86E-03	5.22E-02	1.13E-09	1.04E-22	4.73E-02
		std	8.15E-09	1.90E-23	3.55E-23	4.30E-02	1.26E-01	4.084E-09	4.02E-23	1.05E-01
Beale	TF_5	best	3.92E-24	9.03E-23	2.11E-23	3.61E-23	1.53E-23	1.01E-23	1.36E-23	**2.22E-24**
		worst	6.76E-19	9.79E-18	5.78E-19	4.22E-18	8.79E-19	2.41E-19	1.32E-19	7.50E-19
		avg	6.44E-20	4.46E-19	4.12E-20	2.10E-19	1.32E-19	5.05E-20	2.17E-20	6.15E-20
		std	1.33E-19	1.83E-18	1.13E-19	7.76E-19	2.68E-19	8.83E-20	3.69E-20	1.56E-19
Easom	TF_6	best	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**
		worst	-1	-1	-1	-1	-1	-1	-1	-1
		avg	-1	-1	-1	-1	-1	-1	-1	-1
		std	0	0	0	0	0	0	0	0
Matyas	TF_7	best	3.39E-26	3.58E-27	1.08E-27	3.78E-25	2.14E-24	4.89E-28	5.67E-28	**5.10E-28**
		worst	4.24E-20	3.63E-25	2.40E-24	2.69E-20	1.10E-19	2.24E-25	4.58E-25	4.03E-21
		avg	3.89E-21	9.76E-26	8.84E-25	2.12E-21	7.05E-21	6.74E-26	8.00E-26	1.94E-22
		std	8.88E-21	9.24E-26	7.94E-25	6.29E-21	2.15E-20	6.04E-26	9.39E-26	7.40E-22
Colville	TF_8	best	7.19E-04	2.11E-05	5.53E-04	**3.61E-06**	4.41E-04	9.28E-06	3.40E-05	1.81E-04
		worst	3.02E-02	2.90E-02	2.62E-01	1.16E-02	4.96E-01	3.88E-02	6.89E-01	2.31E-02
		avg	9.99E-03	6.39E-03	1.96E-02	2.38E-03	4.94E-02	7.71E-03	3.14E-02	3.63E-03
		std	6.68E-03	7.75E-03	4.76E-02	2.26E-03	1.16E-01	8.21E-03	1.24E-01	5.59E-03
Zakharov	TF_9	best	5.64E-11	1.61E-23	**4.58E-24**	5.14E-11	4.11E-11	9.17E-24	1.58E-23	3.56E-11
		worst	3.85E-08	1.87E-22	2.02E-22	1.34E-08	1.82E-08	1.62E-22	2.86E-22	7.22E-09
		avg	4.79E-09	6.37E-23	8.02E-23	2.59E-09	9.01E-09	6.43E-23	9.47E-23	1.45E-09
		std	7.53E-09	4.00E-23	3.87E-23	3.14E-09	8.01E-09	3.91E-23	6.23E-23	1.68E-09
Schwefel 2.22	TF_10	best	2.37E-04	**9.97E-12**	1.58E-11	4.45E-05	6.46E-04	7.80E+00	3.52E-01	5.20E+00
		worst	6.14E-03	2.89E-11	3.93E-11	4.92E-01	1.31E+07	9.86E+00	2.28E+00	1.19E+01
		avg	1.52E-03	1.99E-11	3.04E-11	4.92E-02	8.04E+05	8.81E+00	1.27E+00	1.01E+01
		std	1.31E-03	5.05E-12	5.41E-12	1.48E-01	2.77E+06	7.40E-01	4.71E-01	1.43E+00
Schwefel 1.2	TF_11	best	1.11E+01	**5.43E-23**	1.11E-21	4.44E-02	3.24E+04	5.12E+00	7.45E-01	7.09E+00
		worst	1.25E+02	2.37E-21	4.02E-19	2.78E-01	6.72E+04	8.61E+00	2.11E+00	1.51E+01
		avg	4.60E+01	6.83E-22	2.96E-20	1.37E-01	5.30E+04	7.03E+00	1.45E+00	1.06E+01
		std	2.66E+01	5.93E-22	7.56E-20	6.71E-02	8.15E+03	9.48E-01	3.55E-01	2.27E+00
Dixon-Price	TF_12	best	6.67E-01	6.67E-01	6.67E-01	2.50E-01	3.57E+05	1.17E+02	6.10E+00	**1.94E-04**
		worst	4.67E+00	6.67E-01	6.67E-01	2.96E-01	1.10E+06	2.85E+02	2.06E+01	4.36E-03
		avg	1.58E+00	6.67E-01	6.67E-01	2.52E-01	7.64E+05	2.65E+02	1.43E+01	1.46E-03
		std	1.37E+00	1.76E-06	5.60E-07	8.44E-03	1.51E+05	4.11E+01	3.56E+00	1.20E-03

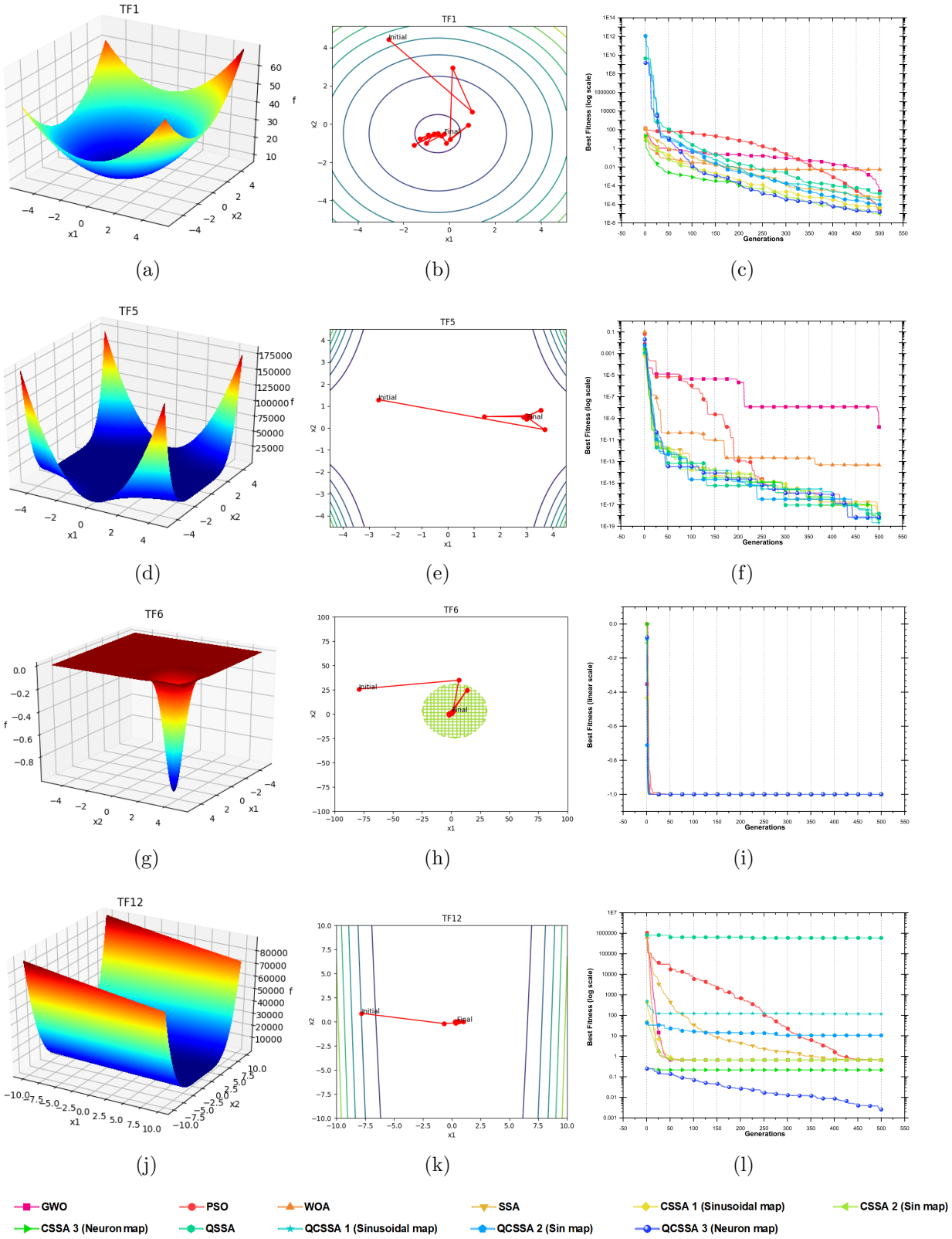


Figure 4.5: Surface plot, contour plot (QCSSA 3) and its corresponding comparative convergence plot on unimodal benchmark functions

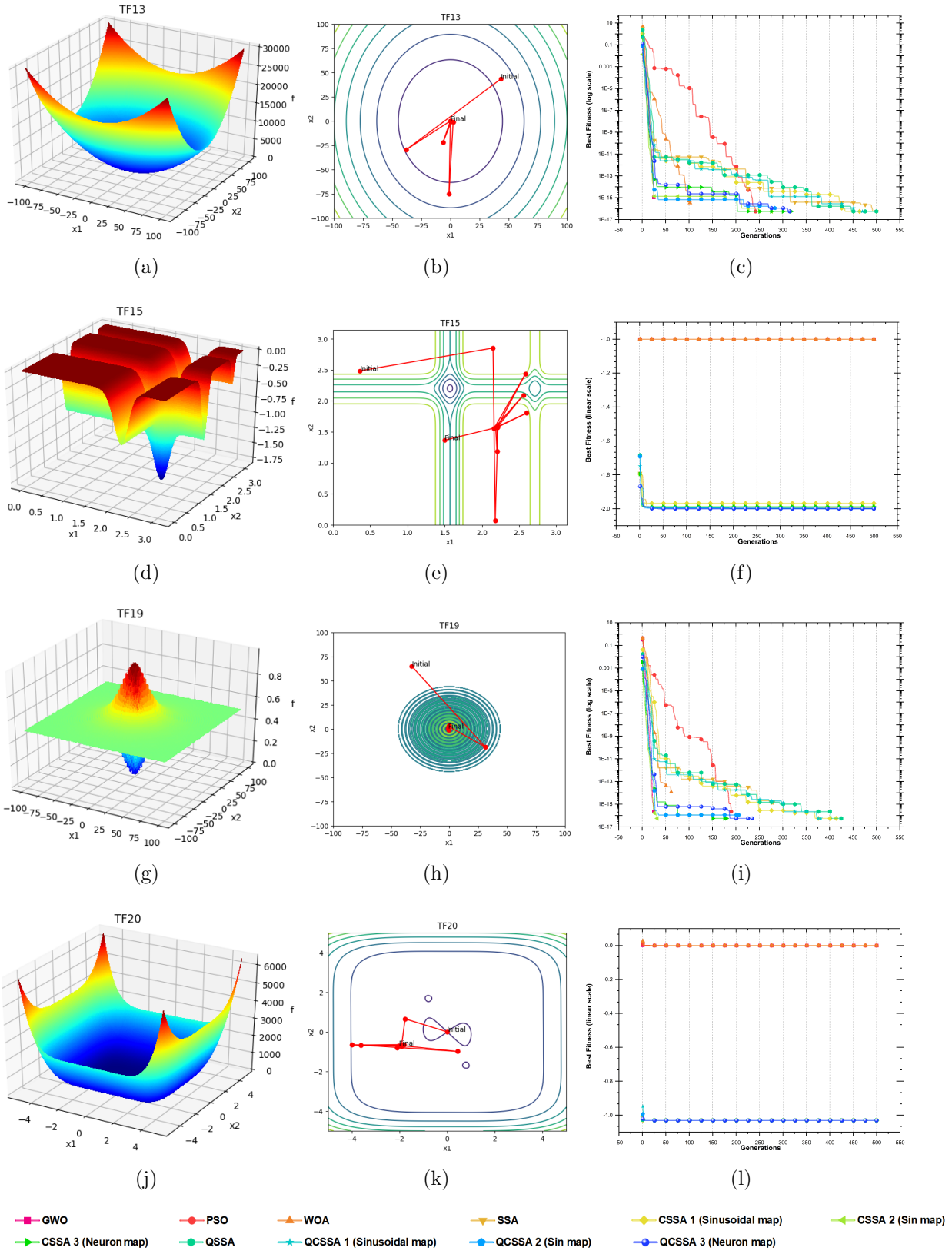
- **Multi-modal benchmark functions:** The purpose of performing the experimental analysis over multi-modal benchmarks is to evaluate the exploration capabilities of the proposals. For this, we have considered the test functions TF_13 to TF_18, which have dimensions ranging from 2 to 30 and belong to the category of multi-modal separable benchmarks. Further, to analyze the efficacy of the proposed quantum-assisted chaotic version and chaotic versions of SSA, we have performed experiments on more challenging benchmark functions with higher complexity that fall into the category of multi-modal non-separable benchmarks. This category includes TF_19 to TF_26, with low to high dimensions. The statistics values obtained are shown in Table 4.9. In Figure 4.6, the convergence rate compared with other well-known optimizers for a few representatives of multi-modal benchmarks is also presented, along with corresponding 3-D surface plots and contour plots of QCSSA (neuron map). It's clear from the observations that either the chaotic variant or its quantum variants perform well as compared to the original SSA in most of the functions. But there are some cases like TF_13 where all optimizers are entrapped in local optima, while on TF_24, chaotic variants perform equally, but quantum variants are not performing well, which can be acceptable according to NFL theorem that states-no optimizer is a universal optimizer. Further, on TF_20, all reached to optima except PSO, GWO, and WOA. There are several cases where the proposals beat the existing optimizers PSO, GWO, WOA, and original SSA.

4.5.6.3 Comparison of convergence of QCSSA, and CSSA with other optimization algorithms on genomic datasets

The convergence analysis takes into account a single weighted metric - fitness, as defined by equation (4.39). The fitness function takes into account two critical metrics: test accuracy and the number of features selected by the algorithm. The fitness function

Table 4.9: Statistical results analysis on multi-modal benchmarks

Function name			Normal and its Proposed Chaotic Variants				Quantum and its Chaotic Variants			
			SSA	**CSSA1**	**CSSA2**	**CSSA3**	**QSSA**	**QCSSA1**	**QCSSA2**	**QCSSA3**
Bohachevsky1	TF_13	best	**-5.55E-17**	**-5.55E-17**	**-5.55E-17**	**-5.55E-17**	**-5.55E-17**	**-5.55E-17**	**-5.55E-17**	**-5.55E-17**
		worst	-5.55E-17	-5.55E-17	-5.55E-17	-5.55E-17	7.21E-04	-5.55E-17	-5.55E-17	-5.55E-17
		avg	-5.55E-17	-5.55E-17	-5.55E-17	-5.55E-17	2.61E-05	-5.55E-17	-5.55E-17	-5.55E-17
		std	1.88E-32	1.88E-32	1.88E-32	1.88E-32	1.32E-04	1.88E-32	1.88E-32	1.88E-32
Booth	TF_14	best	3.93E-23	9.76E-24	1.80E-23	1.85E-23	3.41E-23	1.56E-23	3.10E-23	**1.19E-24**
		worst	1.66E-19	3.49E-19	2.95E-20	1.18E-19	1.54E-19	7.12E-20	1.47E-22	3.55E-20
		avg	1.19E-20	1.54E-20	2.20E-21	9.11E-21	8.37E-21	6.15E-21	1.10E-22	3.58E-21
		std	3.27E-20	6.40E-20	5.47E-21	2.66E-20	2.84E-20	1.45E-20	4.04E-21	9.41E-21
Michalewicz2	TF_15	best	**-1.8013**	**-1.8013**	**-1.8013**	**-1.8013**	**-1.8013**	**-1.8013**	**-1.8013**	**-1.8013**
		worst	-1.99E+00	-1.99E+00	-1.00E+00	-1.99E+00	-2.00E+00	-1.99E+00	-2.00E+00	-1.99E+00
		avg	-1.81E+00	-1.75E+00	-1.82E+00	-1.81E+00	-1.87E+00	-1.88E+00	-1.87E+00	-1.84E+00
		std	3.51E-02	2.08E-01	5.52E-02	4.74E-02	9.47E-02	9.30E-02	9.15E-02	6.96E-02
Michalewicz5	TF_16	best	**-4.687**	**-4.687**	**-4.687**	**-4.687**	-4.681	-4.685	**-4.687**	**-4.687**
		worst	-4.53E+00	-3.69E+00	-3.69E+00	-3.70E+00	-3.60E+00	-3.47E+00	-3.25E+00	-4.89E+00
		avg	-4.68E+00	-4.41E+00	-4.39E+00	-4.61E+00	-4.42E+00	-4.27E+00	-4.30E+00	-4.41E+00
		std	5.90E-02	3.93E-01	4.02E-01	2.36E-01	3.63E-01	4.69E-01	4.34E-01	3.52E-01
Michalewicz10	TF_17	best	**-9.59E+00**	-9.27E+00	-9.16E+00	-9.51E+00	-8.59E+00	-7.97E+00	-8.87E+00	-8.15E+00
		worst	-8.06E+00	-6.41E+00	-5.84E+00	-7.27E+00	-4.63E+00	-3.70E+00	-3.44E+00	-4.52E+00
		avg	-9.03E+00	-8.34E+00	-7.78E+00	-8.70E+00	-6.62E+00	-5.59E+00	-5.58E+00	-6.14E+00
		std	3.53E-01	7.07E-01	8.05E-01	6.54E-01	1.10E+00	1.19E+00	1.36E+00	9.55E-01
Rastrigin	TF_18	best	1.39E+01	**0**	**0**	4.97E+00	2.61E+01	**0**	**0**	1.39E+01
		worst	8.06E+01	0	0	5.29E+01	1.18E+02	0	0	4.88E+01
		avg	3.75E+01	0	0	2.79E+01	4.68E+01	0	0	2.55E+01
		std	1.23E+01	0	0	1.11E+01	1.67E+01	0	0	7.97E+00
Schaffer	TF_19	best	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
		worst	9.72E-03	0	0	9.72E-03	9.72E-03	9.72E-03	0	9.72E-03
		avg	1.94E-03	0	0	1.30E-03	2.33E-03	3.52E-04	0	3.24E-04
		std	3.95E-03	0	0	3.36E-03	4.15E-03	1.77E-03	0	1.77E-03
Six hump camel back	TF_20	best	**-1.0316**	**-1.0316**	**-1.0316**	**-1.0316**	**-1.0316**	**-1.0316**	**-1.0316**	**-1.0316**
		worst	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316
		avg	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316
		std	6.78E-16	6.78E-16	6.78E-16	6.78E-16	3.07E-04	6.75E-16	4.01E-05	3.84E-05
Boachevsky2	TF_21	best	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
		worst	0	0	0	0	0	0	0	0
		avg	0	0	0	0	0	0	0	0
		std	0	0	0	0	0	0	0	0
Boachevsky3	TF_22	best	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
		worst	5.00E-16	0	0	5.55E-17	2.22E-16	0	0	0
		avg	5.55E-17	0	0	1.85E-18	2.41E-17	0	0	0
		std	1.29E-16	0	0	1.01E-17	5.19E-17	0	0	0
Shubert	TF_23	best	**-186.73**	**-186.73**	**-186.73**	**-186.73**	**-186.73**	**-186.73**	**-186.73**	**-186.73**
		worst	-186.73	-186.73	-186.73	-186.73	-186.73	-186.73	-186.73	-186.73
		avg	-186.73	-186.73	-186.73	-186.73	-186.73	-186.73	-186.73	-186.73
		std	3.88E-13	3.89E-13	2.60E-13	3.54E-13	8.07E-03	0	2.99E-03	3.63E-13
Rosenbrock	TF_24	best	4.18E+00	2.31E+01	2.28E+01	**1.83E-01**	4.27E+02	2.37E+01	2.33E+01	2.80E+01
		worst	2.23E+03	2.85E+01	2.70E+01	7.71E+01	5.21E+35	7.75E+01	8.86E+36	8.86E+36
		avg	1.48E+02	2.41E+01	2.39E+01	1.63E+01	5.21E+34	3.00E+01	1.43E+36	3.07E+35
		std	3.98E+02	1.16E+00	1.00E+00	1.71E+01	1.59E+35	9.46E+00	3.23E+36	1.64E+36
Griewank	TF_25	best	3.17E-05	8.20E-05	3.79E-05	**7.49E-06**	7.81E-02	1.07E+01	1.60E+01	3.08E-02
		worst	1.15E-01	1.07E+01	1.33E+01	5.16E+00	2.45E+00	3.66E+01	4.60E+01	1.29E+01
		avg	2.87E-02	4.77E+00	5.49E+00	1.84E+00	1.09E+00	2.07E+01	2.83E+01	4.14E+00
		std	3.06E-02	2.84E+00	4.23E+00	1.80E+00	5.09E-01	6.95E+00	6.79E+00	3.07E+00
Ackley	TF_26	best	3.54E-04	**3.19E-12**	4.54E-12	3.49E-05	2.04E+01	2.04E+01	2.02E+01	2.03E+01
		worst	4.63E-02	7.28E-12	1.16E-11	5.94E-04	2.11E+01	2.07E+01	2.06E+01	2.11E+01
		avg	7.83E-03	5.04E-12	6.95E-12	1.23E-04	2.09E+01	2.06E+01	2.04E+01	2.06E+01
		std	1.17E-02	1.02E-12	1.70E-12	1.01E-04	1.22E-01	8.46E-02	1.16E-01	2.54E-01



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

Figure 4.6: Surface plot, contour plot (QCSSA 3) and corresponding comparative convergence plot on multi-modal benchmarks functions

is designed in such a way that users can prioritize one metric over another in required settings. The trade-off between test accuracy and the number of features selected will depend purely on the area of application and dataset used. The results of QCSSA are compared to those of some well-known algorithms, including WOA, GWO, PSO, as well as original SSA, and shown in Table 4.10. The results for various chaotic functions used by CSSA are shown in columns 7, 8, and 9, and the average fitness for quantum versions for the same set of chaotic functions is shown in columns 11, 12, and 13. The average fitness value is calculated by running the algorithm ten times, and the standard deviations are shown in Table 4.10. As we can infer from the table, QCSSA, especially the one quantum version with neuron map chaotic function, outperforms other algorithms and their chaotic variants significantly on most datasets. For seven out of nine datasets, QCSSA is superior over all the other algorithms and has a lower standard deviation in most cases, indicating our technique's benefits over others. In Table 4.11, we show the accuracy achieved by all algorithms and their chaotic variants on $G_{d_{test}}$. Though, test accuracy seems to be higher for other algorithms compared to QCSSA because of the huge number of features selected by these algorithms. Our goal is to achieve very high test accuracy while reducing the number of features used for training. Thus, comparing these algorithms solely on test accuracy is unfair. In Figure 4.7, we show the plot for fitness vs. generations, and we can see that the QCSSA with neuron map function outperforms all the other optimizers by a significant margin.

In most cases, QCSSA with neuron map converges faster than other algorithms, reducing the computation required for the feature selection process and achieving the best fitness value. On very few datasets, the SSA with a chaotic neuron map function dominates over others. This behavior can be attributed to the impact of the neuron map, as the same function when used with the quantum version, outperforms others. In the following subsection, we present more information on the feature selection ability of the algorithm. Further, the average execution time taken by optimizers on genomic

Table 4.10: Fitness value (average over 10 runs) (lower the value better) - SVM

Dataset	Measure	PSO	GWO	WOA	SSA	CSSA			QSSA	QCSSA		
						Sinusoid	Sin Map	Neuron		Sinusoid	Sin Map	Neuron
GDS1615	Avg	0.2029	0.2029	0.2029	0.0441	0.0403	0.0161	0.0079	0.0442	0.0404	0.0319	**0.0000**
	SD	0.0001	0.0017	0.0012	0.0001	0.0001	0.0194	0.0158	0.0001	0.0002	0.0158	0.0000
GDS531	Avg	0.0687	0.0687	0.0662	0.0122	0.0010	0.0002	0.0001	0.0197	0.0008	0.0002	**0.0000**
	SD	0.0012	0.0112	0.0111	0.0116	0.0003	0.0000	0.0000	0.0124	0.0003	0.0001	0.0000
GDS968	Avg	0.1511	0.0252	0.1523	0.0997	0.0624	**0.0158**	0.0154	0.0845	0.0548	0.0235	0.0230
	SD	0.0017	0.0016	0.0018	0.0191	0.0188	0.0186	0.0187	0.0115	0.0184	0.0186	0.0186
GDS1962	Avg	0.1421	0.1699	0.1524	0.0692	0.0355	0.0220	0.0216	0.0693	0.0444	0.0306	**0.0216**
	SD	0.0008	0.0023	0.0001	0.0001	0.0105	0.0001	0.0000	0.0001	0.0005	0.0105	0.0000
GDS2545	Avg	0.0732	0.0731	0.0731	0.1863	0.1192	0.0647	0.0643	0.1917	0.1297	0.0807	**0.0536**
	SD	0.0001	0.0003	0.0024	0.0107	0.0216	0.0131	0.0273	0.0001	0.0264	0.0170	0.0240
GDS2546	Avg	0.1369	0.1363	0.1368	0.1116	0.0866	0.0645	**0.0375**	0.1116	0.0918	0.0646	0.0429
	SD	0.0002	0.0001	0.0153	0.0001	0.0106	0.0131	0.0214	0.0001	0.0130	0.0131	0.0273
GDS2547	Avg	0.1073	0.1074	0.1073	0.0870	0.0615	0.0224	0.0166	0.0897	0.0450	0.0114	**0.0056**
	SD	0.0000	0.0003	0.0045	0.0002	0.0109	0.0110	0.0135	0.0082	0.0135	0.0135	0.0110
GDS3268	Avg	0.1750	0.2097	0.1756	0.1698	0.1387	0.1270	0.0991	0.1697	0.1333	0.1105	**0.0771**
	SD	0.0002	0.0015	0.0088	0.0001	0.0174	0.0220	0.0220	0.0000	0.0208	0.0001	0.0321
GDS3929	Avg	0.3128	0.3121	0.3121	0.1917	0.1616	0.1273	0.1104	0.1944	0.1665	0.1327	**0.0771**
	SD	0.0015	0.0022	0.0115	0.0110	0.0107	0.0135	0.0174	0.0082	0.0003	0.0207	0.0441

Table 4.11: Test accuracy (average over 10 runs) - SVM

Dataset	Measure	PSO	GWO	WOA	SSA	CSSA			QSSA	QCSSA		
						Sinusoid	Sin Map	Neuron		Sinusoid	Sin Map	Neuron
GDS1615	Avg	92.56%	92.59%	92.59%	80.93%	82.96%	84.44%	80.00%	81.48%	84.07%	85.56%	77.41%
	SD	0.21%	0.20%	0.15%	2.04%	4.29%	2.51%	5.02%	2.34%	3.01%	1.81%	2.16%
GDS531	Avg	92.77%	92.77%	93.97%	95.18%	92.53%	91.81%	90.60%	95.06%	91.81%	95.53%	83.61%
	SD	0.69%	0.86%	0.68%	1.20%	2.07%	0.90%	2.79%	1.26%	1.41%	2.34%	4.61%
GDS968	Avg	62.93%	48.15%	64.81%	60.74%	73.33%	73.33%	76.67%	63.52%	70.37%	74.44%	73.33%
	SD	0.88%	0.35%	2.10%	5.48%	3.01%	3.63%	4.48%	4.31%	2.03%	2.72%	6.15%
GDS1962	Avg	84.56%	85.71%	85.35%	87.24%	87.35%	88.37%	87.76%	86.84%	88.57%	88.37%	87.55%
	SD	1.56%	0.36%	1.87%	1.46%	2.29%	1.19%	1.71%	2.16%	2.45%	1.04%	1.50%
GDS2545	Avg	95.24%	85.89%	85.32%	85.13%	86.15%	84.10%	80.00%	84.87%	84.87%	81.28%	77.95%
	SD	1.44%	0.19%	1.65%	0.63%	2.05%	1.74%	4.18%	0.96%	2.21%	4.18%	3.84%
GDS2546	Avg	87.50%	87.50%	87.50%	80.13%	79.50%	79.75%	77.00%	80.13%	79.75%	81.50%	77.50%
	SD	0.33%	0.12%	1.80%	0.37%	1.70%	2.00%	3.59%	0.88%	1.46%	1.22%	3.79%
GDS2547	Avg	79.22%	77.92%	76.66%	80.00%	81.56%	80.26%	78.96%	81.04%	79.74%	82.86%	78.70%
	SD	0.15%	0.08%	1.45%	1.76%	1.51%	2.23%	3.22%	1.66%	2.67%	2.08%	2.54%
GDS3268	Avg	80.76%	76.92%	79.74%	85.38%	84.62%	83.85%	83.08%	82.77%	84.36%	86.12%	82.56%
	SD	1.10%	0.23%	1.22%	1.18%	0.81%	0.63%	4.01%	1.06%	0.51%	1.31%	2.38%
GDS3929	Avg	76.60%	74.02%	75.32%	85.97%	84.42%	84.16%	80.52%	86.23%	83.90%	83.90%	74.55%
	SD	0.98%	0.34%	1.33%	1.82%	2.60%	2.52%	3.67%	2.48%	4.69%	1.32%	5.73%

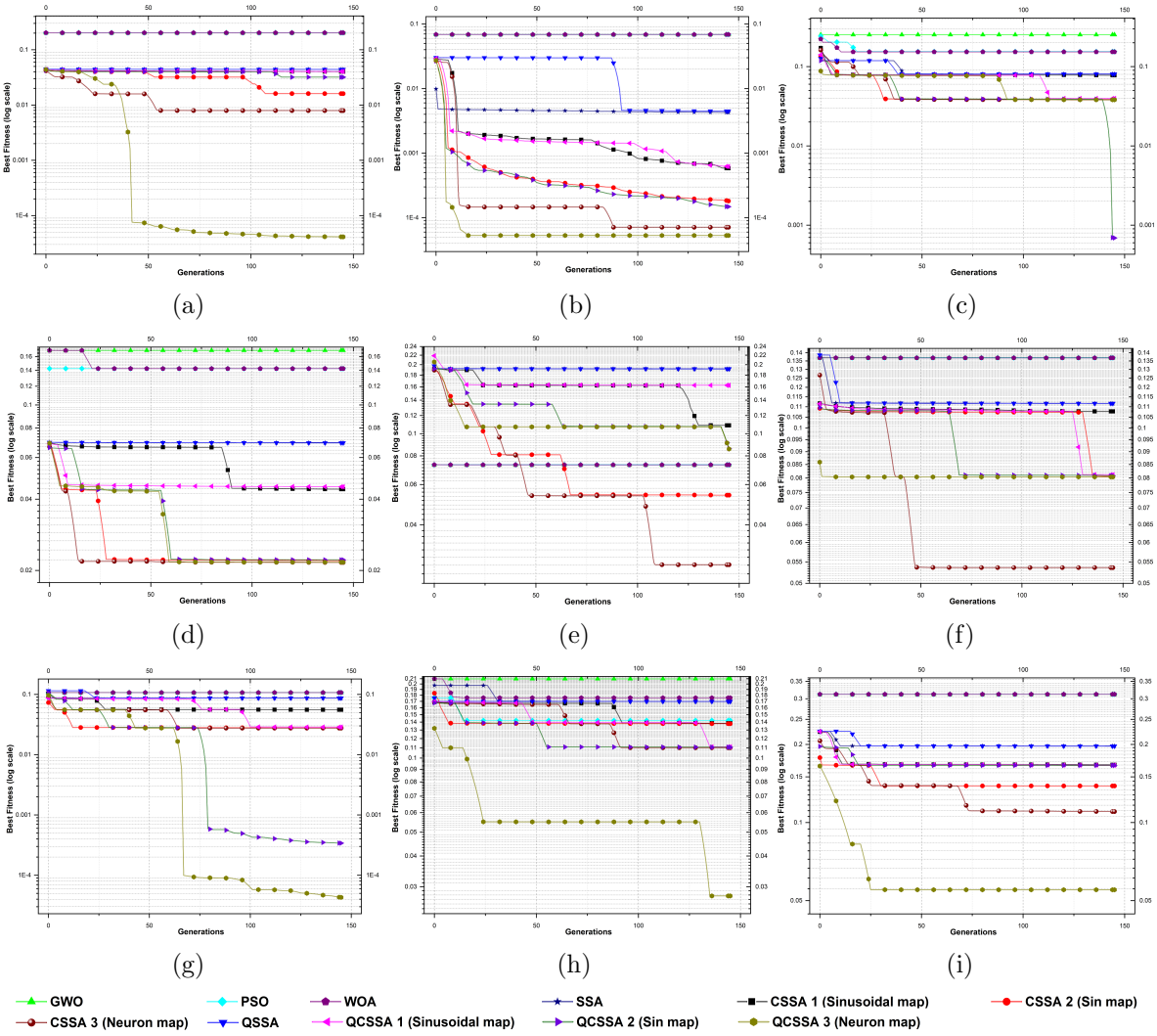
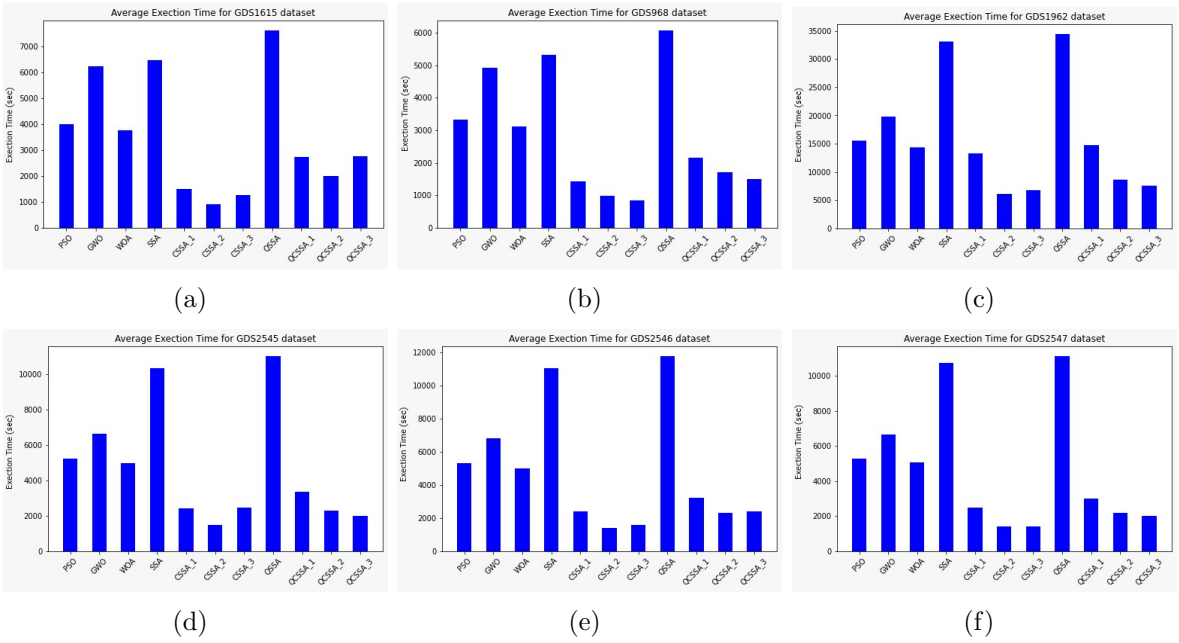


Figure 4.7: Best fitness(log scale) v/s generations plots for experiments on all genomic datasets for QCSSA, CSSA, SSA, and other algorithms (a) GDS1615 (b) GDS531 (c) GDS968 (d) GDS1962 (e) GDS2545 (f) GDS2546 (g) GDS2547 (h) GDS3268 (i) GDS3929

datasets has been shown in Figure 4.8. It is clearly visible that in almost all cases, CSSA and QCSSA are able to reduce the execution time compared to others.

4.5.6.4 Feature selection ability of QCSSA and CSSA

The primary goal of our algorithm is to select as few features as possible while maintaining good accuracy. Another essential characteristic is reproducibility, which is crucial for real-world applications. We expect the algorithm to choose fewer features yet con-



(a) (b) (c)

(d) (e) (f)

Figure 4.8: Average execution time in seconds for experiments on all genomic datasets for QCSSA, CSSA, SSA, and other algorithms (a) GDS1615; (b) GDS968 (c) GDS1962 (d) GDS2545 (e) GDS2546 (f) GDS2547

Table 4.12: Selected no. of features (average over 10 runs) (lower the value better) - SVM

Dataset	Measure	PSO	GWO	WOA	SSA	CSSA			QSSA	QCSSA		
						Sinusoid	Sin Map	Neuron		Sinusoid	Sin Map	Neuron
GDS1615	Avg	8123.32	6693.35	6623.85	6150.30	908.20	327.00	**30.40**	6225.50	1109.80	299.00	**56.00**
	SD	3.18	5.36	8.25	153.04	96.61	69.90	16.09	126.32	205.69	62.24	64.88
GDS531	Avg	5003.23	4590.00	4853.23	4276.80	907.00	206.80	54.60	4235.20	781.80	212.40	**28.60**
	SD	23.13	10.25	17.56	128.55	242.52	43.17	16.08	87.81	264.09	76.11	13.60
GDS968	Avg	6026.33	4545.35	4536.17	4133.40	1322.20	**542.60**	160.00	4117.90	1402.80	572.20	**123.00**
	SD	12.32	8.25	6.23	107.01	338.23	217.24	50.20	97.58	287.69	183.12	142.09
GDS1962	Avg	16536.14	14425.32	14662.78	13656.20	3120.80	1513.60	**261.20**	13744.80	4038.60	1238.20	**326.20**
	SD	18.48	3.25	3.22	179.03	236.47	210.88	141.53	254.10	1456.07	360.34	95.63
GDS2545	Avg	5236.23	4533.56	4511.22	4085.80	1407.20	419.00	**36.60**	4155.80	1228.60	388.20	**59.00**
	SD	20.14	12.22	16.45	109.87	260.94	99.69	9.97	86.08	355.79	70.11	31.50
GDS2546	Avg	5123.65	4719.36	4612.33	4348.80	941.00	303.20	**75.20**	4338.20	811.00	404.80	**58.40**
	SD	43.13	5.33	12.56	66.12	317.41	63.43	28.92	82.29	145.31	103.04	35.70
GDS2547	Avg	4736.21	4647.56	4356.78	4211.40	906.60	381.20	95.40	4193.80	972.60	396.20	**65.20**
	SD	33.12	11.22	18.46	158.87	313.77	51.25	62.99	103.99	122.31	79.68	16.92
GDS3268	Avg	14939.11	14813.22	14762.12	14214.20	3662.00	1469.80	264.00	14154.20	3969.60	1516.00	**159.60**
	SD	52.63	18.23	21.33	166.43	808.94	360.09	171.84	148.27	1198.28	270.80	131.24
GDS3929	Avg	12277.23	9863.23	9633.78	9040.60	4041.00	1488.20	719.60	9018.10	2845.00	1423.80	**234.40**
	SD	64.52	28.22	22.12	148.41	833.77	533.97	363.30	145.54	630.67	292.83	256.71

sistently achieve the same or higher accuracy at each run. In Table 4.12, we show the average number of features selected by QCSSA, CSSA, and other standard algorithms. The average is taken over ten runs, and respective fitness values can be found in Table 4.10. It is evident from the data presented in Table 4.12 that the QCSSA neuron map variant selects the least number of features for most of the datasets when compared to all other algorithms. As an example, consider the result for the GDS531 dataset (Table 4.12 second row); the number of features selected by the QCSSA Neuron map is more than 100 times lower compared to PSO, GWO, WOA, and SSA. In a few datasets, the

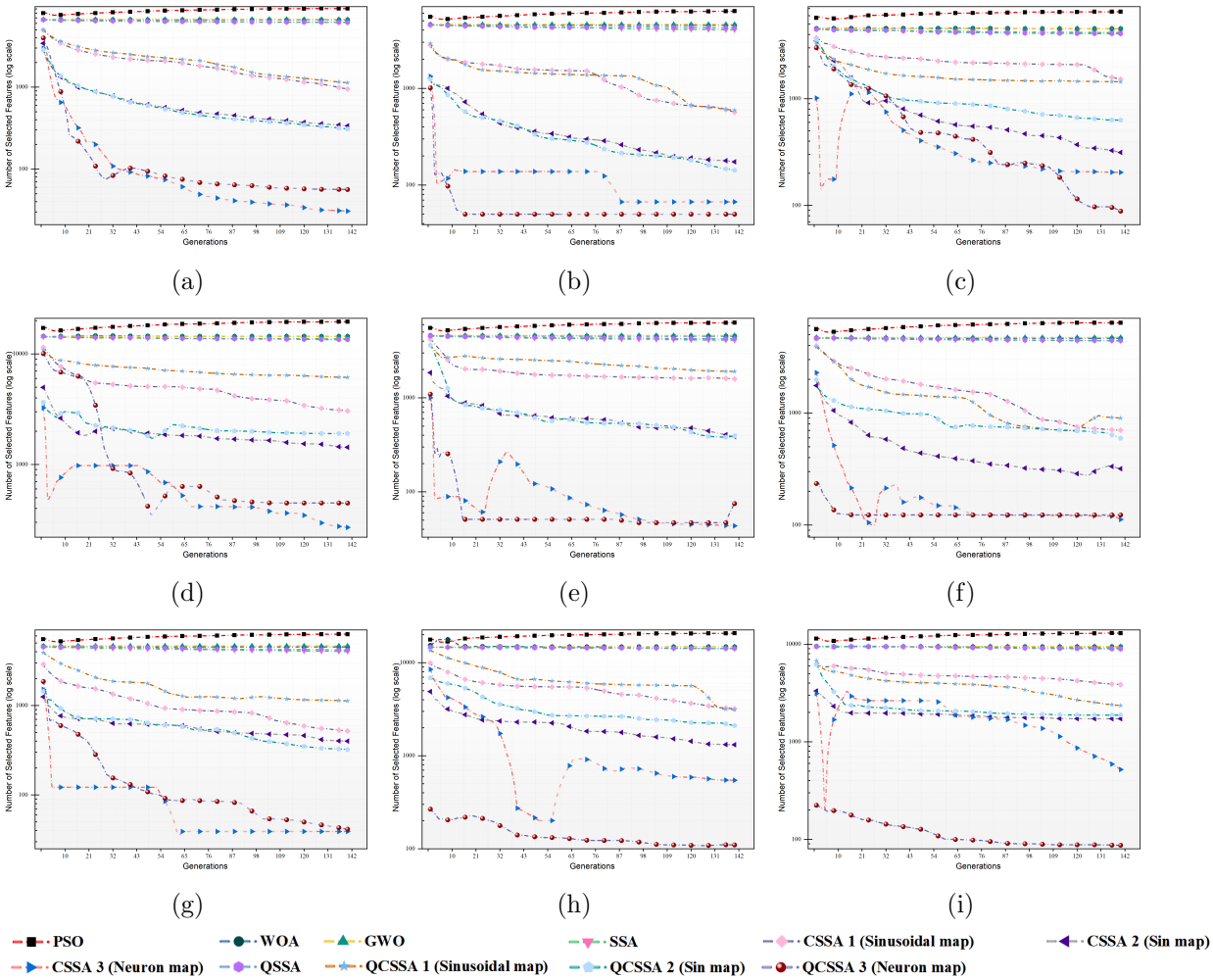


Figure 4.9: Number of selected features(log scale) v/s generations plots for experiments on all genomic datasets for QCSSA, CSSA, SSA, and other algorithms (a) GDS1615 (b) GDS531 (c) GDS968 (d) GDS1962 (e) GDS2545 (f) GDS2546 (g) GDS2547 (h) GDS3268 (i) GDS3929

CSSA neuron map variant selects fewer features, which reiterates the neuron map's utility for feature selection problems. Another variant of QCSSA with sine map gives even higher accuracy by utilizing a few more features as compared to the quantum neuron variant (QCSSA3) and a chaotic variant of SSA with neuron map (CSSA3) on datasets like GDS531, GDS2547, and GDS3268 in terms of accuracy. While on GDS1615 and GDS2545, PSO attained higher accuracy, but with 8123 and 4736 features, respectively, which is very high in comparison to our proposals with slightly lower accuracy. Further,

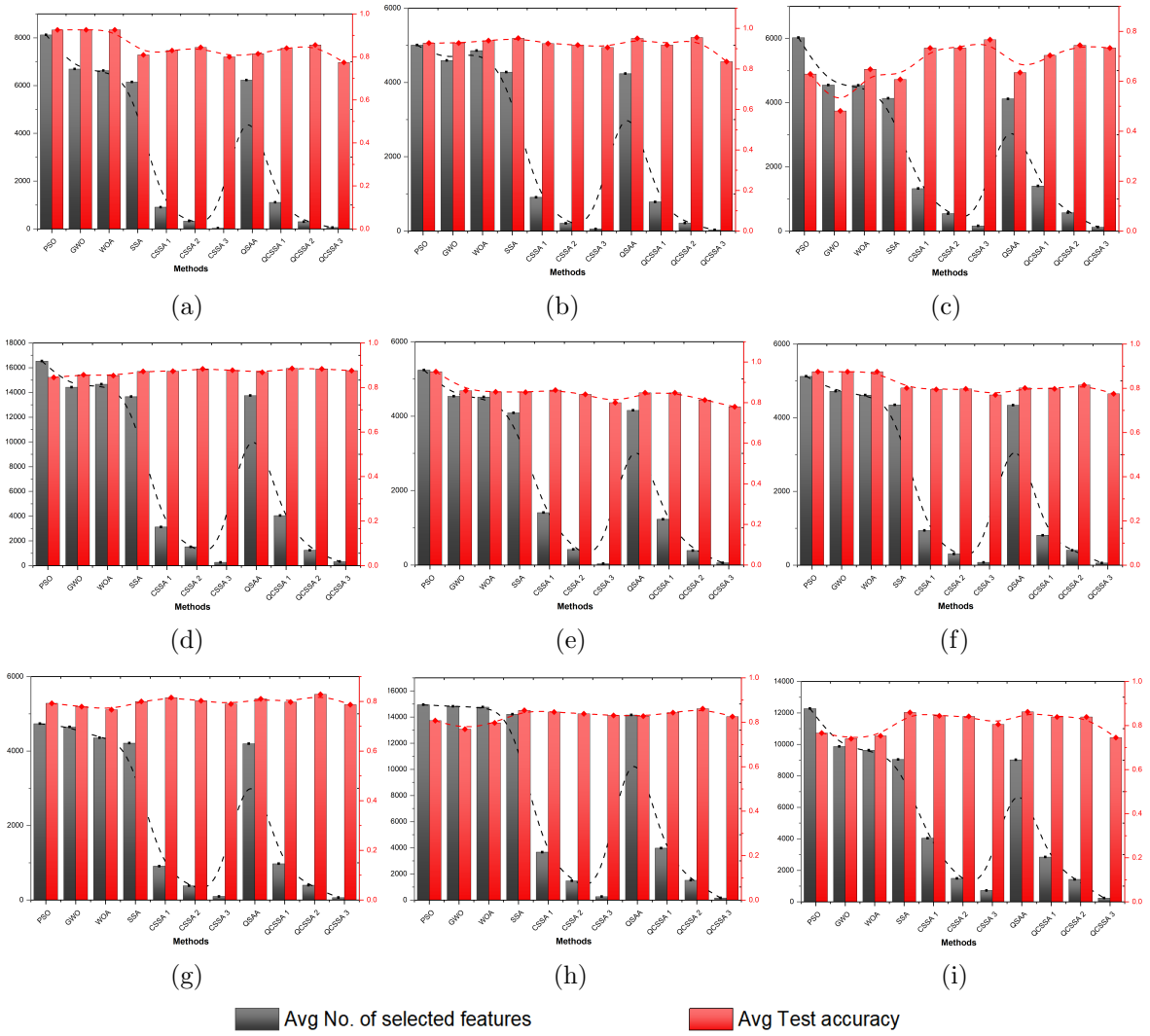
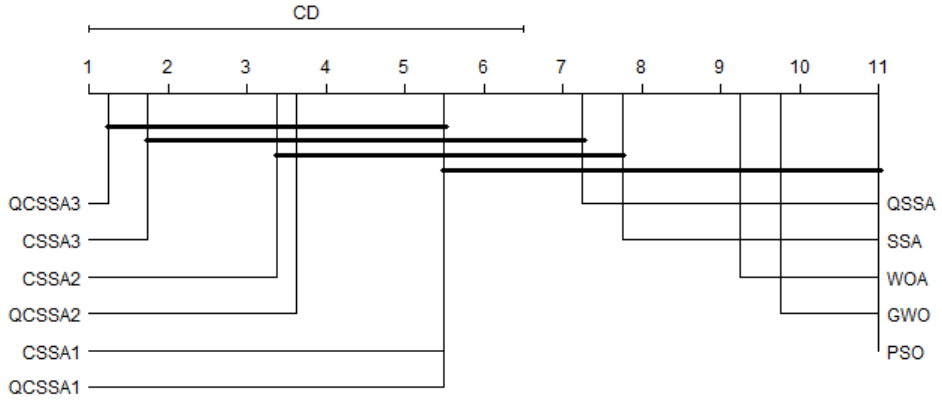


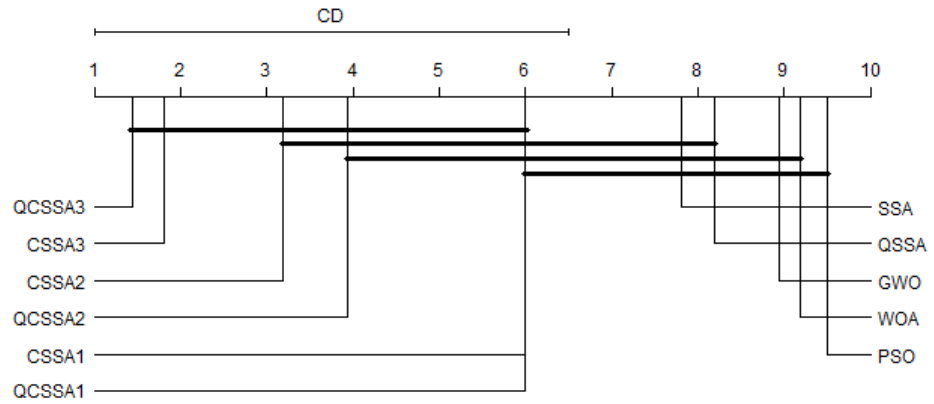
Figure 4.10: Twin axis bar plots for comparative study of the number of selected features and test accuracy for experiments on all genomic datasets for QCSSA, CSSA, SSA and other algorithms (a) GDS1615 (b) GDS531 (c) GDS968 (d) GDS1962 (e) GDS2545 (f) GDS2546 (g) GDS2547 (h) GDS3268 (i) GDS3929

we also would like to highlight that QCSSA selects not only less number of features but also has faster convergence according to plots in Figure 4.9.

To outline the advantage of our method over others, we show a side-by-side comparison of test accuracy and the number of features selected by the algorithms as bar plots in Figure 4.10. Figure 4.10 red bars indicate the average test accuracy over multiple runs, and grey bars indicate the number of features selected. Our algorithm consistently picks the fewest number of features with the least amount of loss in accuracy, which is acceptable. In some cases, accuracy is critical over the number of selected features;



(a)



(b)

Figure 4.11: Critical difference diagram for (a) average selected features and (b) average fitness function. The goodness of the model is ranked from best to worst (left to right), and the bold lines indicate methods that do not produce statistically significant differences.

Table 4.13: WTL ACC-Feature

Method	PSO	GWO	WOA	SSA
CSSA 1	[5, 4, 0]	[6, 3, 0]	[6, 3, 0]	[5, 4, 0]
CSSA 2	[5, 4, 0]	[5, 4, 0]	[5, 4, 0]	[4, 5, 0]
CSSA 3	[4, 5, 0]	[5, 4, 0]	[5, 4, 0]	[2, 7, 0]
QSSA	[6, 3, 0]	[6, 3, 0]	[5, 4, 0]	[4, 3, 2]
QCSSA 1	[5, 4, 0]	[5, 4, 0]	[5, 4, 0]	[3, 6, 0]
QCSSA 2	[6, 3, 0]	[6, 3, 0]	[6, 3, 0]	[7, 2, 0]
QCSSA 3	[3, 6, 0]	[5, 4, 0]	[4, 5, 0]	[2, 7, 0]

one can choose the CSSA neuron map variant or a quantum variant with a sine map (QCSSA2). In summary, our algorithm benefits are multiple folds, especially in terms of convergence and the number of features selected by the algorithm, and it is a good fit for genomic datasets where the number of features is too high.

The comparison of these methods is further analyzed using a critical difference diagram, as shown in Figure 4.11. These graphs show the average ranking as well as pairwise differences between optimizers. The Nemenyi post-hoc statistical test [256] is performed with a significance level of 0.05 to draw the critical difference diagram. Additionally, experimental statistics for Win-Tie-Lose (WTL) for comparing proposals against pre-existing and baseline approaches have been presented in Table 4.13 and Table 4.14. Accuracy with the number of selected features by the optimizer is chosen as a criterion in Table 4.13. In contrast, Table 4.14 presents the comparison based on fitness value with the number of selected features by the optimizers.

4.5.6.5 Comparison among various chaotic functions of CSSA and QCSSA

It is evident from previous subsections that QCSSA outperforms algorithms such as PSO, GWO, SSA, and WOA. However, among the chaotic version and quantum chaotic versions of SSA, some variants perform significantly better than others. Our experi-

Table 4.14: WTL Fitnees-Feature

Method	PSO	GWO	WOA	SSA
CSSA1	[8, 1, 0]	[7, 2, 0]	[8, 1, 0]	[9, 0, 0]
CSSA2	[9, 0, 0]	[9, 0, 0]	[9, 0, 0]	[9, 0, 0]
CSSA3	[9, 0, 0]	[9, 0, 0]	[9, 0, 0]	[9, 0, 0]
QSSA	[8, 1, 0]	[7, 2, 0]	[8, 1, 0]	[3, 3, 3]
QCSSA1	[8, 1, 0]	[7, 2, 0]	[8, 1, 0]	[9, 0, 0]
QCSSA2	[8, 1, 0]	[8, 1, 0]	[8, 1, 0]	[9, 0, 0]
QCSSA3	[9, 0, 0]	[9, 0, 0]	[9, 0, 0]	[9, 0, 0]

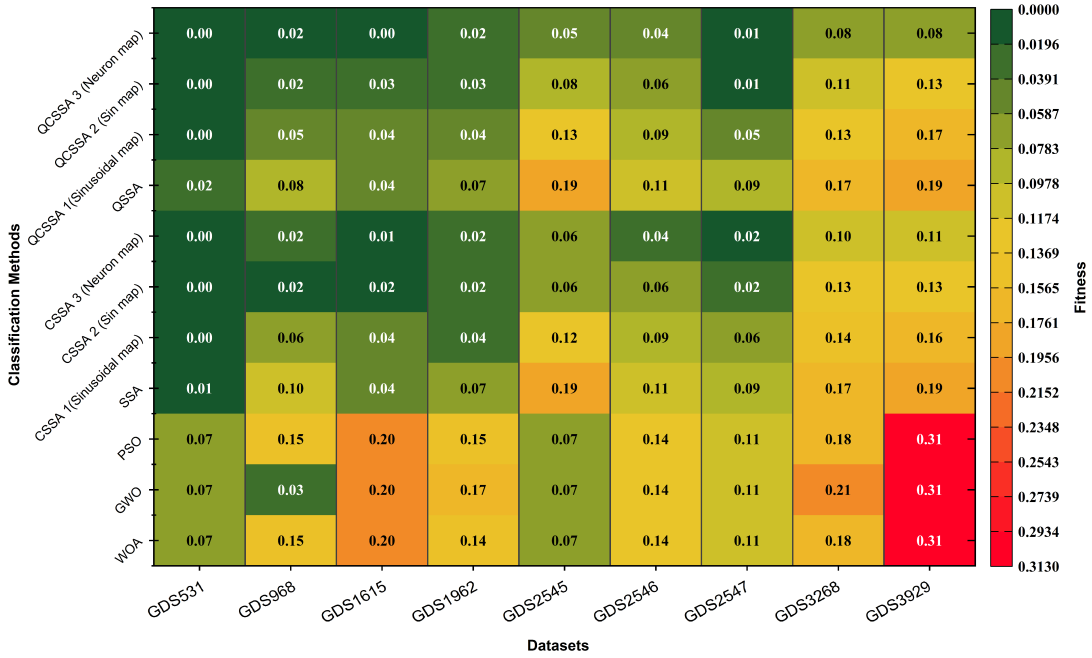


Figure 4.12: Heatmap comparing fitness values across all methods for all datasets.

ments show that the neuron map function works well compared to sinusoidal and sin map chaotic functions. This behavior can be consistently observed in Table 4.10 and Table 4.12 for both the quantum version and non-quantum versions of SSA. The heatmap in Figure 4.12 presents a comparison of fitness values across all algorithms for all the datasets used by our method. In Figure 4.12, QCSSA with neuron map chaotic function is much better than other quantum variants of SSA, and SSA with neuron map

chaotic performs better than the quantum version only for a few datasets. For some real-world problems, SSA with neuron map chaotic function can be beneficial over the quantum version, but QCSSA (neuron map) does accomplish better results in our set of experiments. In summary, our extensive experiments reveal QCSSA to be a clear winner over other algorithms.

4.6 Summary

Recent technological advancements are propelling the world toward automated applications. Real-world applications of these automated technologies necessitate complex optimization processes, which must be solved using an efficient optimizer. Among these, feature selection is one of the most complex optimization problems that affect the performance of ML-based applications, particularly when dealing with high-dimensional gene expression datasets. Gene expression profiling plays an important role in the identification and prediction of cancer tumors. However, only a few gene expression features are strongly linked to cancerous tumors. These high-dimensional data contain irrelevant, redundant, and noisy gene features, which adversely affect the accuracy, computation cost, and memory of ML models. A similar effect can be seen in other real-world data utilizing ML techniques for classification. Therefore, optimal feature set selection is critical for overcoming the challenges associated with high-dimensional dataset classification using ML.

In the first part of this chapter, we propose a novel QL-SSA algorithm that combines the strength of QL and SSA to achieve superior results for a combinatorial optimization problem. In contrast to basic QL, we attempted to identify effective solutions employing cooperative agents (squirrels) in a multi-agent context. Since QL is insensitive to exploration point but not much efficient in the exploitation phase, we use QL to enhance the diversity of solutions and exploit the previous solution's evaluation to achieve superior results in our proposed approach. By combining both concepts, we tried to

overcome local optima stagnation limitations and enhance the diversity of solutions. Notably, feature selection is a crucial combinatorial problem in the current scenario due to the advancement in technology that leads to Big data analysis. Big Data analysis is done with the help of ML, and the performance of the predictive model lies on the quality of selected features. Therefore, we provided the hybrid wrapper-based method to solve the optimal feature subset selection problem for data classification. Results are analyzed using two classifiers on 20 real-world benchmark datasets with a varied range of complexity. Two large-scale genomic data were also considered to validate the effectiveness of the proposal. The obtained statistics for classification show the effectiveness and capability of maintaining a good balance between exploration and exploitation of proposed QL-SSA for feature subset selection and classification.

Later, we presented a QCSSA algorithm for optimal gene feature selection in high-dimensional genomic datasets. The efficacy of the proposal is validated using twenty-six mathematical standard benchmark functions as well as real-world applications for high-dimension genomic expression feature selection and classification on nine genomic datasets. We demonstrated that QCSSA with neuron map performs well on these datasets compared to state-of-the-art feature selection algorithms. The proposed algorithm is able to achieve good accuracy while reducing the number of features to around 10,000 times, which is much lower than other contemporary algorithms.