Spark-Enabled Binary Particle Swarm Feature Selection Algorithm Evaluated on Genomic Breast Cancer Data

Simone A. Ludwig
Department of Computer Science
North Dakota State University
Fargo, USA
simone.ludwig@ndsu.edu

***Abstract*—In large-scale machine learning applications, feature selection is essential for reducing dimensionality and computational overhead while maintaining or improving classification performance. Feature selection aims to identify an optimal subset of features to improve classification performance, with evaluation criteria and search strategies forming its two fundamental components. This paper implements a Spark-Enabled Binary Particle Swarm Optimization (BPSO-P) approach applied to classification of Genomic Breast Cancer Data. BPSO-P aims to enhance classification accuracy while efficiently exploring the feature search space. The BPSO-P algorithm is implemented using Spark, leveraging distributed computing to handle large-scale datasets. The performance of the proposed approach is evaluated across multiple dataset sizes using speedup and scaleup as the primary performance metrics. All datasets consist of 5,000 features and the sizes vary between 6.2 GB to 31 GB.**

***Index Terms*—Large-scale Data; Spark Framework; Gene Expression Data**

I. Introduction

Feature selection aims to identify an optimal subset of features to improve classification performance by exploring the search space of possible feature combinations. It involves two main components: an evaluation criterion and a search strategy. Based on the evaluation criterion, methods are typically classified as wrapper or filter approaches.

Wrapper methods use a learning algorithm to evaluate feature subsets, often achieving higher performance but at the cost of increased computational complexity and reduced generality [**?**]. In contrast, filter methods perform selection independently of any specific model, making them more efficient and generalizable. Their effectiveness depends on the chosen evaluation criterion, such as information-theoretic, dependency, consistency, or distance-based measures [**?**], [**?**], [**?**], [**?**], [**?**].

For a dataset with n features, the size of the search space is 2^n, making exhaustive search infeasible in most practical scenarios [**?**]. As a result, the choice of search strategy has a significant impact on the effectiveness of a feature selection algorithm. Numerous search techniques, such as greedy strategies, have been applied to feature selection; however, many of these methods are prone to becoming trapped in local optima and/or incur high computational costs [**?**], [**?**]. Therefore, the development of a computationally efficient global search technique is essential for designing effective feature selection algorithms.

Evolutionary feature selection methods emerged as wrappers or hybrid schemes that search the subset space of features while optimizing classification performance [**?**], [**?**]. Early work applied genetic algorithms (GA) and ant colony optimization (ACO) to select relevant features for tasks such as text and biomedical classification, showing that evolutionary and swarm principles can reduce dimensionality while preserving or improving accuracy [**?**]. Particle swarm optimization (PSO) was adapted to binary search spaces for feature selection, where each particle encodes a candidate subset and the swarm converges toward compact, high-accuracy feature sets [**?**], [**?**].

Several Binary PSO variants have been proposed to address feature selection under binary constraints. Stick Binary PSO redefines momentum as stickiness and velocity as flipping probability to enhance population diversity, though unconstrained flipping increases computational cost and slows convergence [**?**]. Evolutionary multitasking frameworks have applied PSO to multiple feature selection tasks simultaneously [**?**], while mutation-enhanced BPSO has been used for applications such as spam detection to reduce premature convergence and improve performance [**?**]. Competitive Swarm Optimization and its extensions introduce performance constraints, filter-based guidance, and surrogate models to improve diversity and evaluation efficiency; however, these methods may increase exploitation tendencies and susceptibility to local optima [**?**], [**?**].

II. Related Work

Parallelization techniques for nature-inspired algorithms are essential for handling large datasets and computationally intensive tasks, and this work therefore focuses on reviewing parallel methods with an emphasis on evolutionary and swarm intelligence approaches. MPI (Message Passing Interface) is one of the early frameworks used for this purpose. It provides a standard for communication between multiple nodes

in a distributed system, enabling efficient parallel computations. Several studies have explored using MPI to parallelize nature-inspired algorithms. For instance, a parallel genetic algorithm based on the MPI environment was developed, adapting the serial GA to a parallel implementation [**?**]. Another example is the MPI–OpenMP hybrid approach used to implement a multi-objective Particle Swarm Optimization (PSO) algorithm, combining distributed memory (MPI) and shared memory (OpenMP) parallelization [**?**]. Additionally, an MPI-based parallel GA was proposed for multiple geographical optimization problems, using a hybrid of fixed-position and sliding models [**?**].

MapReduce, introduced in [**?**], revolutionized the way large-scale data processing tasks are handled by abstracting the data processing into two fundamental operations: Map and Reduce. This framework has been effectively used to parallelize algorithms like Genetic Programming and Particle Swarm Optimization. For example, a MapReduce-based Particle Swarm Optimization algorithm was developed to handle large-scale clustering tasks, showing notable improvements in processing time and scalability [**?**].

Apache Spark, an advancement over MapReduce, offers in-memory computation, which significantly speeds up data processing tasks. For instance, [**?**] compared the performance of Glowworm Swarm Optimization (GSO) when implemented using Spark and MapReduce. Their findings indicated that Spark's in-memory computation capabilities provided faster and more efficient processing, making it more suitable for high-dimensional function optimization tasks. Another approach in [**?**] also implemented a parallel fuzzy clustering algorithm using Spark. This study demonstrated that the Spark-based algorithm outperformed traditional clustering methods, providing better scalability and handling of real-time data streams [**?**]. This research highlights Spark's capability to efficiently process and analyze large datasets, making it a preferred choice for implementing nature-inspired algorithms in a parallel computing environment.

[**?**] implemented a scalable design of an artificial bee colony for big data classification using Apache Spark. The performance results reveal that the proposed fitness algorithm can efficiently deal with unbalanced datasets as well as scale very well, achieving excellent speedup and scaleup results. Similar implementations were done for the Differential Evolution (DE) [**?**] and the PSO algorithms [**?**].

Another Spark-enabled approach implemented the Bison algorithm showing good scalability and speedup of the algorithm [**?**]. However, the number of features of the dataset was small being of size 80, and thus the need for the investigation of large datasets with a large feature set as proposed in this paper (5,000 features).

This paper consists of three parts. First, it presents a Spark-enabled implementation of a Binary Particle Swarm Optimization (BPSO-P) algorithm for feature selection [**?**], demonstrating how nature-inspired optimization can be effectively integrated with distributed data processing frameworks. Second, it provides a systematic performance evaluation of the proposed approach using speedup and scaleup metrics across multiple dataset sizes, offering insight into the scalability behavior and practical limitations of BPSO-P in a distributed environment. Third, through extensive experimental analysis, this work identifies the trade-offs between dataset size, parallel efficiency, and distributed overhead, thereby establishing practical guidelines for deploying Spark-based Binary PSO on moderate-scale clusters.

III. Spark-Enabled Binary PSO for Feature Selection

This section describes the Binary Particle Swarm Optimization (BPSO-P) algorithm used for feature selection and its parallel implementation using Apache Spark.

A. BPSO with Paired Evaluation for Feature Selection

Binary Particle Swarm Optimization with Paired Evaluation (BPSO-P) [**?**] is a filter-based feature selection approach that combines Binary Particle Swarm Optimization (BPSO) with information-theoretic measures to balance feature relevance and redundancy. In this method, each particle encodes a candidate feature subset as a binary vector, where a value of 1 indicates that the corresponding feature is selected.

The fitness function is designed to maximize the relevance of the selected features to the class label while simultaneously minimizing redundancy among the selected features. Relevance is quantified using mutual information between individual features and the class label, while redundancy is measured as the mutual information between all pairs of selected features. The fitness function is defined as:

$$\text{Fitness} = D - R,$$

where

$$D = \sum_{x \in X} I(x; c), \quad R = \sum_{x_i, x_j \in X} I(x_i; x_j).$$

Here, X denotes the selected feature subset, c is the class label, and $I(\cdot;\cdot)$ represents mutual information. D computes the mutual information between each feature and the class labels using pairwise calculations, thereby measuring the relevance of the selected feature subset. R evaluates the mutual information between pairs of selected features, capturing the redundancy within the subset. The fitness function is defined as a maximization objective that seeks to increase relevance (D) while simultaneously reducing redundancy (R) in the selected feature subset. This formulation encourages the selection of features that are highly informative with respect to the class, while discouraging the inclusion of redundant features.

BPSO-P employs the standard BPSO update mechanism, where particle velocities are updated using cognitive and social components, and positions are updated probabilistically via a sigmoid transformation of the velocity. The algorithm iteratively updates personal best and global best solutions based on the fitness value until a stopping criterion is met. The final selected feature subset corresponds to the global best

particle, which is then evaluated using a classifier on unseen test data.

By explicitly modeling pairwise feature interactions, BPSO-P is effective at significantly reducing the dimensionality of the feature space while maintaining comparable classification performance, making it well suited for high-dimensional datasets where computational efficiency and generality are important.

B. Spark-Enabled BPSO-P Feature Selection Algorithm

Apache Spark is a distributed computing framework that enables parallel data processing across clusters. Data is partitioned and processed concurrently by worker nodes, coordinated by a driver program. This architecture allows scalable and efficient evaluation of candidate solutions during optimization.

Spark-Enabled BPSO-P is implemented as an iterative parallel process in Spark. After loading and preprocessing the dataset (feature scaling and label encoding), the data are split into training and testing sets and distributed as RDDs.

At each iteration, the swarm of particles is broadcast to all worker nodes. Each executor evaluates particle fitness on its local data partition. Fitness values are aggregated via Spark accumulators, after which $pbest$ and $gbest$ are updated centrally. Particle velocities and positions are then updated, and the process repeats until convergence. The final $gbest$ particle defines the selected feature subset. Algorithm **??** shows the algorithm.

Algorithm 1 Spark-Enabled BPSO-P for Feature Selection

```
Initialize particle swarm with random binary positions and velocities
Initialize personal bests (pbest) and global best (gbest)
Load dataset into Spark RDD
for t = 1 to T do
    Broadcast particle swarm to all executors
    Initialize accumulator for fitness values
    for each data partition (in parallel) do
        Evaluate particle fitness (classification accuracy)
        Update accumulator
    end for
    Aggregate fitness and update pbest and gbest
    Update particle velocities
    Update particle positions using sigmoid transfer function
end for
return gbest as the optimal feature subset
```

IV. Experimental Setup

This section starts off with a description of the dataset, followed by the parameters used for the experiments, as well as the performance metrics. The infrastructure for the running of the experiments is outlined in this section as well.

Table I: Datasets Used For Scaling Experiments

Folds / Dataset Name	Size
200	6.20 GB
400	12.40 GB
600	12.40 GB
800	18.60 GB
1,000	30.99 GB

A. Dataset

The TCGA-BRCA dataset has been used for the experiments, which has been extracted from the Cancer Genome Atlas (TCGA) [**?**]. TCGA is a pioneering cancer genomics initiative that has molecularly characterized more than 20,000 primary cancer samples and corresponding normal tissues across 33 cancer types. This collaborative project, launched in 2006 by the National Cancer Institute (NCI) and the National Human Genome Research Institute, brought together experts from various fields and institutions. Over the course of twelve years, TCGA produced more than 2.5 petabytes of data, encompassing genomic, epigenomic, transcriptomic, and proteomic information. This vast dataset, which has already contributed to advancements in cancer diagnosis, treatment, and prevention, continues to be publicly accessible to the research community. The dataset used is BRCA (Breast Cancer) containing of two classes - cancer or not cancer. It consists of 5,000 columns or features and 1,224 rows/samples, and is of size 30.99 MB.

Since Spark processing is evaluated with respect to speedup and scaleup, the dataset is expanded by duplicating it in increments of $200\times$, from $200\times$ up to $1,000\times$. The resulting dataset sizes are presented in Table **??**. It is important to note that this expansion applies only to the number of samples, while the number of features remains constant at 5,000.

B. Algorithm Parameters

The Spark-Enabled BPSO-P implementation used the following parameters: 1) Number of iterations = 100; 2) Number of particles = 100; 3) w = 0.5; 4) c_1 = 2.0; 5) c_2 = 2.0; and 6) Classifier: K-Nearest Neighbor (KNN). An 80/20 training–testing split was used, with a fixed random seed of 42 to ensure reproducibility. Furthermore, feature selection was performed using only the training data.

C. Performance Metrics

Even though the focus of this paper is on the scalability aspect of the parallel Spark implementation of the BPSO-P algorithm, however, the classical measure for classification are also provided using KNN after the important features have been selected:

- Accuracy: The proportion of correct predictions among the total number of examples.
- Precision: The proportion of true positive predictions among the total positive predictions.
- Recall: The proportion of true positive predictions among the total actual positives.

- F1-score: The harmonic mean of precision and recall, providing a single metric that balances both concerns.

The main focus, however, was the performance of the algorithm that was assessed using the speedup to measure the parallelization ability of the algorithm by taking the ratio of the running time on a single node to the running time on parallel nodes n. In addition, the speedup is calculated as follows, where the dataset size is fixed while the number of nodes is increased in a certain ratio.

$$Speedup = \frac{T_1}{T_n}, \tag{1}$$

where T_1 is the running time using a single node, and T_n is the running time using n nodes.

The scaleup measures how the cluster of nodes are utilized efficiently by the parallel algorithm. The scaleup is calculated as follows, where the dataset size and the number of nodes are increased by the same ratio.

$$Scaleup = \frac{T_{sn}}{T_{Rsn}}, \tag{2}$$

where T_{sn} is the running time for the dataset with size s using n nodes, R is the increasing ratio, and T_{Rsn} is the running time for the dataset with size Rs using Rn nodes.

D. Infrastructure for Running Experiments

The nodes used for the experiments were the Pittsburgh Supercomputing Center's Bridges-2 system, which is equipped with two AMD EPYC 7742 CPUs, each featuring 64 cores (128 threads total per node) with a clock speed between 2.25 GHz and 3.40 GHz. These nodes have either 512 GB of RAM and 3.84 TB of local NVMe SSD storage. The nodes are connected via Mellanox ConnectX-6 HDR InfiniBand with 200 Gbps bandwidth, providing efficient communication across the system, which is key for high-performance computing tasks such as used in this study. Spark version 3.0.1 with Hadoop 2.7 was used for the algorithm implementation.

V. Results

A series of preliminary experiments was conducted to systematically explore the impact of both feature dimensionality and dataset size on the performance and feasibility of the proposed approach. These experiments considered multiple configurations with varying numbers of selected features as well as different sample sizes drawn from the available datasets. The goal was to understand the computational behavior of the algorithm under increasing problem complexity and to identify practical limits imposed by the computing environment. Given a strict wall-time constraint of 72 hours, particular attention was paid to determining the largest dataset that could be processed within this time limit when executing the Spark-Enabled BPSO-P algorithm on a single compute node. The results of these exploratory runs informed the final experimental design by establishing a balance between maximizing dataset size and feature dimensionality while maintaining computational feasibility. This process ensured that the selected experimental configuration fully leveraged the available resources without exceeding time constraints, thereby enabling a fair and reproducible evaluation of the proposed method.

Table II: Execution Times in seconds versus Number of Nodes for All Datasets

Nodes	200 Fold	400 Fold	600 Fold	800 Fold	1,000 Fold
1	42940.36	104783.62	159580.01	211177.02	239580.01
2	26378.01	52672.30	77950.96	105747.69	119817.83
4	13624.34	26614.01	39774.14	52892.52	60635.25
8	7048.28	13853.73	20715.91	27225.67	30167.79
16	3700.84	7329.11	10814.42	14394.78	15984.25
32	2696.17	4339.64	6790.40	8438.63	10903.66
64	1791.44	3807.60	5307.64	6225.71	7910.57

The TCGA-BRCA dataset achieved the following results in terms of classification measures (this dataset contains 5,000 features and 1,224 samples):

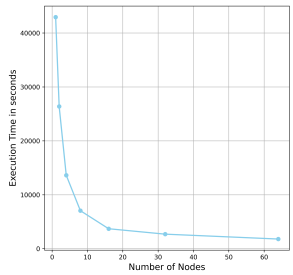
- Number of features: 1,609
- Accuracy = 0.921249
- Precision = 0.914800
- Recall = 0.921249
- F1-score = 0.897548

Figure **??** and Tables **??** and **??** show the execution times and speedup results across increasing numbers of compute nodes for multiple dataset sizes. The results of the dataset of fold 200, 400, 600, 800, and 1,000 are shown. The dashed line represents ideal linear speedup, while the solid line shows the measured performance. Across all dataset sizes, the implementation exhibits near-linear speedup at smaller node counts, indicating effective parallel utilization and low overhead in this setup. As the number of nodes increases, the observed speedup gradually deviates from the ideal case, with the gap becoming more pronounced at higher node counts.

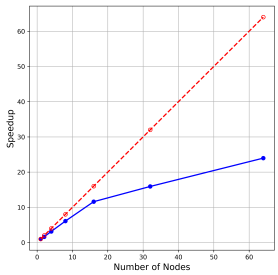
For smaller datasets, scalability saturates earlier, as the computational workload per node becomes insufficient to amortize the communication and scheduling overhead introduced by Spark. In contrast, larger datasets demonstrate improved scalability, achieving higher absolute speedup values and maintaining closer proximity to ideal performance up to moderate node counts. However, even for the largest datasets, diminishing returns are observed beyond approximately 32 nodes, where additional resources yield progressively smaller performance gains. This behavior highlights the combined impact of distributed coordination overhead, data shuffling costs, and synchronization within the Spark execution model.

Overall, the results indicate that the proposed BPSO-P implementation benefits substantially from parallel execution, particularly for larger datasets, while also revealing practical scalability limits when deployed at larger scales on the evaluated cluster configuration.

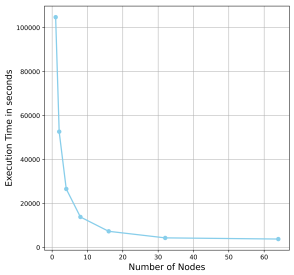
Figure **??** shows the scaleup behavior of the Spark-enabled BPSO-P implementation as the number of compute nodes increases while proportionally scaling the dataset size. Ideally, scaleup performance would exhibit approximately constant execution time as resources and workload grow. However, the results show that execution time remains relatively stable only



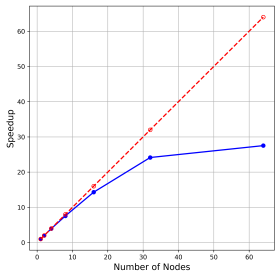
(a) Exec. Time (200)



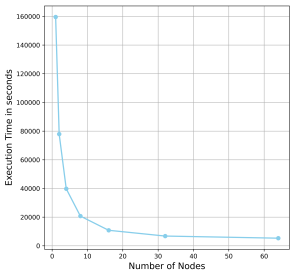
(b) Speedup (200)



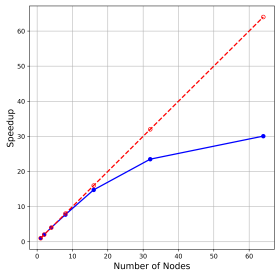
(c) Exec. Time (400)



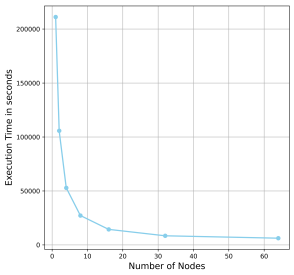
(d) Speedup (400)



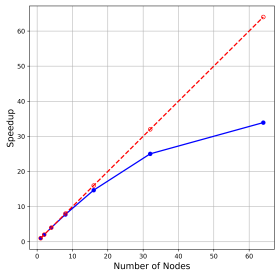
(e) Exec. Time (600)



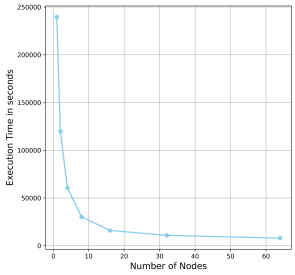
(f) Speedup (600)



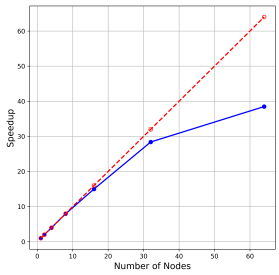
(g) Exec. Time (800)



(h) Speedup (800)



(i) Exec. Time (1,000)



(j) Speedup (1,000)

Figure 1: Execution time and speedup across datasets of sizes 200, 400, 600, 800, and 1,000.

Table III: Speedup versus Number of Nodes for all Datasets

Nodes	200 Fold	400 Fold	600 Fold	800 Fold	1,000 Fold
1	1.00	1.00	1.00	1.00	1.00
2	1.63	1.99	2.05	2.00	2.00
4	3.15	3.94	4.01	3.99	3.95
8	6.09	7.56	7.70	7.76	7.94
16	11.60	14.29	14.76	14.67	14.99
32	15.92	24.15	23.50	25.03	28.39
64	23.97	27.52	30.06	33.92	38.48

at smaller scales and gradually increases as the number of nodes grows. While the execution time increases modestly when scaling from 2 to 16 nodes, a more pronounced rise is observed at larger node counts, particularly beyond 32 nodes. This trend indicates that the overhead associated with distributed execution, including communication, synchronization, and Spark scheduling costs, begins to outweigh the benefits of additional parallelism at higher scales. These results suggest that although the proposed implementation demonstrates reasonable scaleup behavior for moderate cluster sizes, scalability becomes limited at larger scales when executed on a single Spark application configuration.

VI. Conclusions

This study demonstrated the effectiveness of a Spark-enabled Binary Particle Swarm Optimization (BPSO-P) approach for feature selection in large-scale classification tasks. By integrating feature selection with distributed computing, the proposed method effectively reduced feature dimensionality while maintaining competitive classification performance on the Breast Cancer dataset. The results show that BPSO-P can efficiently explore the feature search space and scale to larger dataset sizes when implemented using Spark. Moreover, the experimental evaluation using speedup and scaleup metrics highlights the potential of the proposed approach to leverage

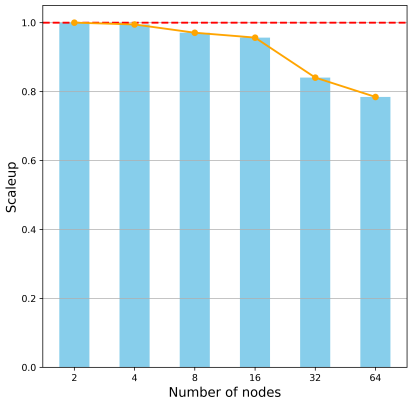


Figure 2: Scaleup

parallelism for improved computational efficiency, making it suitable for data-intensive machine learning applications.

The experimental results demonstrate that the Spark-enabled BPSO-P approach achieves meaningful performance gains through parallel execution, particularly for larger dataset sizes. Across all evaluated configurations, the algorithm exhibits near-linear speedup at smaller node counts, confirming effective utilization of distributed resources. As the number of nodes increases, scalability gradually departs from ideal behavior, with diminishing returns observed at higher node counts due to communication, synchronization, and scheduling overhead inherent to the Spark execution model. Larger datasets consistently achieve higher speedup and better scalability compared to smaller datasets, indicating that increased computational workload per node helps amortize distributed overhead. Overall, these results highlight both the benefits and practical limits of applying Spark-Enabled BPSO-P for feature selection, demonstrating its suitability for moderate-scale deployments while identifying scalability constraints at larger system scales.

ACKNOWLEDGMENT

This work used Bridges-2 at Pittsburgh Supercomputing Center through allocation CIS260044 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by National Science Foundation grants 2138259, 2138286, 2138307, 2137603, and 2138296.

REFERENCES

[1] M. Mafarja, I. Aljarah, A. A. Heidari, A. I. Hammouri, H. Faris, Z. W. Geem, and S. Mirjalili, "Hybrid nature-inspired algorithms for feature selection in high-dimensional classification," Scientific Reports, vol. 15, no. 1, p. 2, 105, 2025.

[2] S. Arora and P. Agrawal, "Ensemble feature selection method based on recently developed nature-inspired algorithms," IEEE Transactions on Emerging Topics in Computational Intelligence, vol. 7, no. 3, pp. 689–701, 2023.

[3] R. K. Sahoo, B. K. Panigrahi, and S. Das, "A hybrid bio-inspired neural model based on Ropalidia marginata for medical classification," Scientific Reports, vol. 15, no. 1, p. 3 240, 2025.

[4] A. K. Das, D. P. Mishra, and S. K. Patra, "Assessment of nature-inspired algorithms for text feature selection," Computer Science, vol. 23, no. 2, pp. 179–198, 2022.

[5] J.-J. Zhang, W.-J. Liu, G.-Y. Liu, "Parallel Genetic Algorithm Based on the MPI Environment," Journal of Telkomnika, vol. 10, no. 7, pp. 1708-1715, 2012.

[6] A. Smith, B. Lee, and C. Wang, "MPI-Based Particle Swarm Optimization for High-Dimensional Problems," IEEE Transactions on Evolutionary Computation, vol. 21, no. 2, pp. 157–169, 2017.

[7] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2004.

[8] A. K. Sangaiah, A. Thangavelu, and S. Sundararaman, "Nature-inspired computing for feature selection and classification," in *Handbook of Research on Big Data Clustering and Machine Learning*, 2023, pp. 320–340.

[9] M. A. Elaziz, D. Oliva, and S. Xiong, "A systematic review of emerging feature selection optimization methods based on nature-inspired algorithms," *IEEE Access*, vol. 9, pp. 108 654–108 677, 2021.

[10] M. Mafarja, I. Aljarah, A. A. Heidari, A. I. Hammouri, H. Faris, Z. W. Geem, and S. Mirjalili, "Hybrid nature-inspired algorithms for feature selection in high-dimensional classification," *Scientific Reports*, vol. 15, no. 1, p. 2, 105, 2025.

[11] B.H. Nguyen, B. Xue, P. Andreae, M. Zhang, A new binary particle swarm optimization approach: Momentum and dynamic balance between exploration and exploitation, IEEE Trans. Cybern. 51 (2) (2021) 589–603, http://dx.doi.org/10.1109/TCYB.2019.2944141.

[12] K. Chen, B. Xue, M. Zhang, F. Zhou, An evolutionary multitasking-based feature selection 454 method for high-dimensional classification, IEEE Trans. Cybern. 52 (7)(2022) 7172–7186, http://dx.doi.org/10.1109/TCYB.2020.3042243.

[13] Y. Zhang, S. Wang, P. Phillips, G. Ji, Binary PSO with mutation operator for feature selection using decision tree applied to spam detection, Knowl.-BasedS yst. 64 (2014) 22–31, http://dx.doi.org/10.1016/j.knosys.2014.03.015.

[14] R. Cheng, Y. Jin, A competitive swarm optimizer for large scale optimization, IEEE Trans. Cybern. 45 (2) (2015) 191–204, http://dx.doi.org/10.1109/TCYB.2014.2322602.

[15] B. H. Nguyen, B. Xue, M. Zhang, A constrained competitive swarm optimizer with an SVM- 468 based surrogate model for feature selection, IEEE Trans. Evol. Comput.28 (1) (2024) 2–16, 469 http://dx.doi.org/10.1109/TEVC.2022.3197427.

[16] S. A. Ludwig and I. Aljarah, "MapReduce Intrusion Detection System Based on a Particle Swarm Optimization Clustering Algorithm," in *IEEE Congress on Evolutionary Computation*, 2013.

[17] G. Miryala and S. A. Ludwig, "Comparing Spark with MapReduce: Glowworm Swarm Optimization Applied to Multimodal Functions," *International Journal of Swarm Intelligence Research*, vol. 9, no. 3, pp. 1–22, 2018.

[18] S. A. Ludwig, "MapReduce-based Fuzzy C-Means Clustering Algorithm: Implementation and Scalability," *International Journal of Machine Learning and Cybernetics*, vol. 6, no. 6, pp. 923–934, 2015.

[19] J. Al-Sawwa and M. Almseidin, "A Spark-Based Artificial Bee Colony Algorithm for Unbalanced Large Data Classification," *Information*, vol. 13, no. 11, p. 530, 2022. [Online]. Available: https://doi.org/10.3390/info13110530

[20] J. Al-Sawwa and S. A. Ludwig, "Performance Evaluation of a Cost-Sensitive Differential Evolution Classifier using Spark – Imbalanced Binary Classification," *Journal of Computational Science*, vol. 40, 2020.

[21] J. Al-Sawwa and S. A. Ludwig, "Parallel Particle Swarm Optimization Classification Algorithm Variant implemented with Apache Spark," *Journal of Concurrency and Computation: Practice and Experience*, vol. 32, no. 2, 2020.

[22] S. A. Ludwig, J. Al-Sawwa, and A. M. Misquith, "Parallelization of the Bison Algorithm Applied to Data Classification," *Algorithms*, vol. 17, no. 11, p. 501, 2024. [Online]. Available: https://doi.org/10.3390/a17110501

[23] L. Cervante, Bing Xue, M. Zhang and Lin Shang, "Binary particle swarm optimisation for feature selection: A filter based approach," 2012 IEEE Congress on Evolutionary Computation, Brisbane, QLD, Australia, 2012, pp. 1-8, doi: 10.1109/CEC.2012.6256452.

[24] The Cancer Genome Atlas Research Network. (2013). The Cancer Genome Atlas Pan-Cancer 482 analysis project. Nature Genetics, 45(10), 1113-1120. https://doi.org/10.1038/ng.2764.