Large Scale Distributed Optimization using Apache Spark: Distributed Scalable Shade-Bat (DistSSB)

Fahad Maqbool, Saad Razzaq, Asif Yar
Department of Computer Science & IT
University of Sargodha
Sargodha, Pakistan
{fahad.maqbool, saad.razzaq}@uos.edu.pk

Hajira Jabeen
Cluster of Excellence on Plant Sciences (CEPLAS)
Institute for Plant Sciences, University of Cologne
Cologne, Germany
hajira.jabeen@uni-koeln.de

***Abstract*—Large Scale Global Optimization (LSGO) is interesting for its applications in machine learning, e.g. in deep learning or knowledge graph embeddings. Evolutionary algorithms (EA) have been found efficient in solving complex optimization problems. However, the performance of conventional EAs degrades with the increasing number of decision variables due to the lack of scalability. This paper proposes a scalable, parallel, distributed and hybrid EA named Distributed Scalable Shade-BatScalable Shade Bat (DistSSB) to solve LSGO problems. DistSSB is inspired by the exploration capability of the SHADE algorithm and exploitation feature of the Bat algorithm (BA). To achieve scalability, DistSSB is implemented using the popular distributed in-memory framework, Apache Spark. DistSSB distributes its population into multiple sub-populations using the island model. Each sub-population is independently evolved using SHADE or BA. After the migration interval, the best solutions are broadcasted employing the mesh topology. We have compared the scalability, efficiency, and efficacy of DistSSB with SHADE-ILS (CEC-2018 Winner) and GL-SHADE algorithm on the CEC-2013 LSGO benchmark function suite.**

For most functions, DistSSB has obtained better optimization results in lesser execution time as compared to SHADE-ILS and GL-SHADE. We have tested and shown the scalability of DistSSB for up to *"one million"* dimensions, whereas SHADE-ILS and GLSHADE fail to scale up for larger problems.

***Index Terms*—LSGO, Apache Spark, Differential Evolution, SHADE-ILS, GL-SHADE, Evolutionary Computation, Distributed, Parallel, Scalable**

I. Introduction

Real-life optimization problems are becoming increasingly complex due to an increase in the number of decision variables as a result of digitalization. The traditional EA, i.e., Differential Evolution (DE) [1], Particle Swarm Optimization (PSO) [2], and Genetic algorithm (GA) [3] show promising results for relatively low dimensional problems. For high dimensions, EA suffers from performance bottleneck (slow convergence speed and convergence to local optima), termed as the curse of dimensionality [4]. Therefore it is important to develop distributed and scalable techniques for Large Scale Global Optimization (LSGO). LSGO problems have arisen in many fields, including but not limited to engineering [5], machine learning, resource scheduling, and vehicle routing in large scale traffic network [6], business intelligence, & data mining [7]. Knowledge Graph Embedding (KGE) modelling is a large scale optimization problem where large scale knowledge graphs like DBpedia (2 million entities and 10 million facts), Wikidata (1.5 million entities and 3 million facts), and Google KG (500 million entities and 3.5 billion facts) are transformed to multidimensional matrices. KGE is used to find missing, or new facts in the knowledge graphs [8]. Deep transfer learning is another application area to optimize Neural Network weights and then to use these pre-trained networks for new problems. Keras's VGG16, VGG19, NASNetLarge, and EfficientNetB7 are few large scale networks that contain 13.8 million, 14.3 million, 8.8 million, and 6.6 million decision variables respectively [9].

A problem is categorized as an LSGO problem [10], if the number of decision variables increases beyond 1000. The LSGO problems can range from fully separable problems to non-separable problems, where separability describes the extent to which a problem can be divided into sub-problems, such that a fitness function can evaluate them independently. A problem is fully separable if all its variables are independent of each other [11].

The established LSGO approaches have been categorized [12] as Cooperative Co-evolution (CC) and Non-Cooperative Co-evolution (NCC). In CC, a high dimensional problem is divided into low dimensional sub-problems, making them capable of being solved using conventional EA. Sub-problems maintain separate sub-populations to ensure diversity [13]. Although CC has been used in different meta-heuristics (e.g. Ant colony optimization (ACO) [14], GA [15], DE [16] and PSO [17]) but the performance of CC changes abruptly in case of complex and overlapping sub-problems. Additionally, CC is sensitive to the grouping strategy. In NCC, a problem in never decomposed and hence doesn't require a grouping strategy. The problem is solved without requiring detailed information about the decomposition. This makes NCC a suitable choice for many overlapping and complex optimization problems. NCC techniques include SHADE with Iterative Local Search (SHADE-ILS) [5], adaptive population

Acknowledgements This work is partially funded by the Deutsche Forschungsgemeinschaft (DFG) under Germany´s Excellence Strategy – EXC 2048/1 – project 390686111 (CEPLAS)

978-1-7281-8393-0/21/$31.00 ©2021 IEEE

differential evolution with dual control strategy [18], enhanced adaptive differential evolution algorithm [7], and SHADE-SPA memetic framework [19].

In this article, we have proposed a NCC technique, Distributed, Scalable Shade BAT (DistSSB). DistSSB is a parallel and scalable approach. DistSSB combines Success-History based parameter Adaptation for Differential Evolution (SHADE) [20] and Bat algorithm (BA) [21] to achieve adept exploration and exploitation, respectively. Using the Island model [22], we have divided the population into multiple sub-populations where each sub-population is evolved in parallel. SHADE is used for exploration in a few partitions, whereas BAT is used for exploitation in the remaining partitions. The performance of DistSSB is tested on the CEC-2013 benchmark function suite [10], and the results have been found better from SHADE-ILS and GL-SHADE for most functions with improved execution time for 1000 and 2000 decision variables. To verify the scalability, we have compared DistSSB, SHADE-ILS, and GL-SHADE on one million dimensions for $f12$ and found that SHADE-ILS, and GL-SHADE are unable to scale up to this problem size while DistSSB shows convincing results. To the best of our knowledge, DistSSB is the first, open source algorithm that addresses the problem of scalability.

The rest of the paper is organized as follows. In section II preliminary concepts are discussed. Section III covers the literature review. In section IV, design and implementation of DistSSB is covered. Experimental setup and results are discussed in section V and finally, the conclusion and future work is discussed in section VI.

II. Background

A. Apache Spark

The Apache Spark is an open-source, in-memory unified cluster computing framework and an analytic engine that has gained popularity in recent years due to its fast, in-memory data processing capability [23]. Spark is useful for iterative and interactive data processing. In Spark core API, high-level abstraction is provided using RDD to distribute data across the cluster for further processing. The driver node distributes data among available worker nodes and is responsible for the task scheduling and monitoring of worker nodes. Each worker node receives one or more partitions of the RDD.RDD has two types of operations. (i) Transformations (ii) Actions. Transformations perform a task on an existing RDD to create a new RDD. They are in fact logical execution plans that don't materialize the RDD. Actions materialise the execution by creating a new RDD. The actions result in network communication and may become a performance bottleneck if not used carefully.

B. Island Model

Island model is a population distribution paradigm [24] that divides the initial population into multiple sub-populations (islands). Each island executes an algorithm independently. Depending on the Migration Rate (M_r), islands share solutions with other islands after specified Migration Interval (M_i). Migrated solution(s) replace the worst solutions on each island. The island model enhances algorithms' global search ability by exploring search space in multiple trajectories and improving the execution (convergence) time.

C. BAT Algorithm

Yang and Xin-She proposed a novel meta-heuristic based Bat algorithm (BA) in 2010 [21]. BA uses the echolocation behavior of microbats to find the optimal solution. Following control parameters are used in BA.

- N: Population size
- P: Actual population comprising N bats.
- x_i: Position of Bat_i
- v_i : Velocity of Bat_i
- f_r^{max} & f_r^{min} : Upper and lower frequency bounds
- ϵ: Bandwidth for the calculation of a reasonable gradient.
- r: Pulse rate that determines the exploitation rate.
- it : Current iteration
- A_i : Loudness rate of Bat_i
- $\widehat{A}$: Mean loudness of all bats
- r_i^0 : Initial pulse rate of Bat_i
- α & γ : The two constant parameters of range [0,1] used to update A and r

The velocity v_i (Eq. 2) of Bat_i is influenced by a random frequency f_r (Eq. 1). New bat position x_i can be calculated using Eq. 3.

$$f_r = f_r^{min} + (f_r^{max} - f_r^{min}) * rand[0,1] \tag{1}$$

$$v_i = v_i + (x_i - x_{best}) * f_r \tag{2}$$

$$x_i = x_i + v_i \tag{3}$$

Local search is part of the exploitation process. The new bat position is calculated using a random walk around the x^{best} position using Eq. 4.

$$x_i^{new} = x^{best} + \epsilon\widehat{A} \tag{4}$$

x_i is replaced by x_i^{new} if fitness of x_i^{new} is better than the fitness of x_i and loudness is less than $U[0,1]$. x^{best} is replaced by x^{new} if fitness of x^{new} is better than x^{best}. x_i, r_i, and A_i are updated using Eq. 5 and Eq. 6 respectively. algorithm 1 presents the pseudo-code of the bat algorithm.

$$r_i = r_i^0 * (1 - e^{(-\gamma * it)}) \tag{5}$$

$$A_i = \alpha A_i \tag{6}$$

D. Differential Evolution (DE)

DE is a population-based EA, designed for solving continuous numerical optimization problems [1], [25]. The three control parameters of DE are population size (N), crossover rate (C_R), and scaling factor (F). Mutation, crossover, and selection are the three commonly used DE operators. A new candidate (donor) solution (v_d) is generated by applying the mutation operator. In the crossover phase, the donor solution interacts with the currently selected candidate solution x_i to generate a trail vector u_t. In the selection phase fitness of u_t is compared with x_i. u_t replaces x_i, if $f(u_t)$ is better than $f(x_i)$. $DE/rand/1/bin$ is the classical mutation strategy

Algorithm 1 Bat Algorithm

```
Initialize the population P
while stopping criteria do
    for x_i: 1 to N do
        Calculate f_r using Eq. 1
        Calculate v_i using Eq. 2
        Find x_i using Eq. 3
        if rand[0,1] ≥ r_i then
            x_i^new ← Perform local search using Eq. 4
        end if
        if rand[0,1] ≤ A_i & f(x_i^new) ≤ f(x_i) then
            x_i = x_i^new
            Update r_i using Eq. 5
            Update A_i using Eq. 6
        end if
    end for
end while
```

of DE. Here rand is base vector, 1 specifies the number of difference vector(s), and bin is the recombination type. In DE two recombination types (binomial (bin) and exponential (exp) are commonly used.

E. Success-History based parameter Adaptation for Differential Evolution (SHADE)

Among many variants of DE, SHADE has gained much popularity as it computes the control parameters (crossover CR and scaling factor F) dynamically [20], based on the history. It maintains a history data structure H that contains N entries for crossover (M_{CR}) and scaling factor (M_F). Initially, all values of H are set to a constant value, i.e., 0.5. A random index r_i is selected from $[1 : H]$ and is used to compute values of both control parameters F_i and CR_i in each iteration. F_i is calculated at line 8 in algorithm 2 using Eq. 7.

$$F_i = rand_c(M_{F,r_i}, 0.1) \tag{7}$$

Algorithm 2 SHADE

```
1:  Initialize Population (P), Memory (A') and History (H)
2:  while Termination Criteria do
3:      for x_i : 1 to N do
4:          Calculate parameters F_i and CR_i and by selecting index
            r_i randomly from [1 : H] using Eq. 7 and Eq. 8
5:          Generate trail vector u_t
6:          x_r1 ← rand(P)
7:          x_r2 ← rand(A')
8:          x^best ← [P, N * p]
9:          v_i^d = x_i + F_i * (x^best − x_i) + F_i * (x_r1 − x_r2) using Eq. 7
10:         Generate trail vector u_i^t = { v_i^d  if rand[0,1] < CR_i
                                          { x_i    otherwise
11:         if f(u_i^t) ≤ f(x_i) then
12:             x_i = u_i^t
13:             Update ((M_CR),(M_F)) based on F_i and CR_i
14:         end if
15:     end for
16: end while
```

$$CR_i = rand_n(M_{CR,r_i}, 0.1) \tag{8}$$

CR_i in algorithm 2 at line 4 is calculated using Eq. 8. F_i is selected randomly using Cauchy distributions, and CR_i is selected using uniform (normal) distribution. The upper limit for selection of F_i and CR_i is 0.1.

Instead of selecting the fittest individual as the best, SHADE considers $N * p$ fittest solutions and selects a random solution from them as the best solution x^{best} used at line 8 in algorithm 2. The value of p is calculated using Eq. 9.

$$p = rand[p_{min}, 0.2] \tag{9}$$

p is random number between p_{min} and 0.2. Value of p_{min} is set to 2/N, which guarantees that at least two fittest solutions are selected. 0.2 is the highest value for p as suggested in [26]. SHADE maintains an external archive called memory (A') that is used for the mutation process. A candidate vector x_i that is worse than the trial vector u_t is placed in A'. Pseudo code of SHADE is given in algorithm 2.

III. Related Work

Evolutionary algorithms have been used for optimization from last few decades and several parallel and distributed models have been proposed. With the advancements in distributed and scalable tools to handle big data, like Apache Hadoop and Apache Spark, researchers proposed implementations using these cloud computing frameworks. In this paper, we only cover the implementations covering the big data frameworks, and winners of the LSGO competitions.

Spark based SparkPSODE [27] combined the dominating features of both PSO and DE where individual's position and velocity was updated using PSO while mutation, crossover, and selections operators were adopted from DE. Authors did not address the complex functions with rotations or overlapping factors. Increasing partitions in Spark results in more computational time. This could be improved by utilizing proper computational power of Spark framework. Fahad et al., [28] proposed a scalable GA using Apache Spark named (S-GA) for high dimensional problems. They have used the Island model to split the population into sub-populations. Results have shown that an increase in the number of islands increased the number of migrants causing higher network traffic, and increased execution time of S-GA. However, this resulted in faster convergence in terms of iterations. They have used five simple functions for evaluations and did not cover the complex optimization problems. AlJame et al., [29] proposed an Apache Spark implementation of Whale Optimization algorithm (Spark-WOA). Spark-WOA materialized RDD containing evolved solutions after each iteration. Spark-WOA broadcasted the best solution after each iteration to all the partitions. Frequent materialization of RDD is a bottleneck that resulted in network communication and overall increased execution time. Authors have evaluated their technique on four simple benchmark functions for upto seventy (70) iterations only, whereas the number of decision variables was missing. Spark-based DE with grouping topology (SgtDE) was proposed by He et al., [30] to solve LSGO. The initial population was divided into fifteen sub-populations and three

equal groups. Ring topology was used for migration among the same group's islands, while mesh topology was used for communication between the groups. SgtDE used a population size of 300. Although Apache Spark is a cluster computing framework, SgtDE was tested on a single machine only.

SHADE [31], a variant of DE, maintained history through the memory of Crossover Rate (M_{CR}) and scaling Factor (M_F). Additionally, it preserved a collection of Crossover Rate (CR) and scaling Factor (F) values that have performed well in the past. These values helped producing new (CR, F) pairs by sampling the domain of the parameters adjacent to one of these preserved pairs. SHADE showed a limited evaluation on 100 population size, 30 decision variables, and $3.0 * 10^5$ fitness function evaluations. Also, the execution time for convergence has not been discussed. Oscar and Carlos proposed a Global and Local search using SHADE (GL-SHADE) [32] that divided the population into two parts. SHADE algorithm was executed on one part for exploration, while eSHADEls was executed on other part for exploitation. After certain iterations, both sub-populations shared their best individuals. The GL-SHADE was evaluated on the CEC-2013 benchmark function suite, and it achieved better results. GL-SHADE was tested with 100 population size and the execution time was not reported. SHADE-ILS [5] combined the exploration power of SHADE with the exploitation power of local search strategies for optimization. SHADE-ILS adaptively adjusted its parameters, and the mutation operator selected the best variable among p best solutions. With a population size of 100, SHADE-ILS obtained good results for CEC-2013 benchmarks functions and was selected as the winner of CEC-2018. An Efficient Recursive Differential Grouping for Large-Scale Continuous Problems (ERDG), a CC-based approach, was proposed by Yang et al. [33]. Their focus was to reduce the computation cost by examining historical information used for interrelationship examination and reducing the function evaluation. They have used CEC 2013 benchmark functions for evaluation. SHADE-ILS performed better than ERDG on half of the functions. Cooperative Co-evolution with Recursive Differential Grouping (CC-RDG3) [34] has comparable results to SHADE-ILS. CC-RDG3 is a CC-based technique where the focus is on division of dimensions.

In summary, most of the examples discussed above do not target the scalability and handle problems with large dimensions, even though some of them used big data frameworks. For the comparison, we have selected SHADE-ILS, a NCC algorithm, as it is the winner of CEC-2018 [5] and GL-SHADE [32] as it has obtained better results for some of the functions compared to SHADE-ILS using CEC-2013 benchmark function suite [10]. Although CC-RDG3 is the winner of CEC-2019, we have not selected it for comparison as it is a CC based technique that divides the dimensions and decomposes the overlapping problems, whereas in NCC based technique the whole population is decomposed into multiple groups and hence these methods are not directly comparable.

IV. Distributed Scalable Shade-Bat (DistSSB)

In this section, we detail the proposed Scalable Shade Bat (DistSSB) algorithm. DistSSB uses island model by dividing the population among islands. This helps in achieving speedup and avoiding stagnation through solution sharing among islands. The SHADE algorithm has a strong exploration capability and helps finding new solutions in unexplored search space, whereas, BA algorithms offers an intensive exploitation strategy. Therefore DistSSB combines SHADE (for exploration) and BA (for exploitation) for value-added optimization. Inspired from center-based sampling strategy [35], we select the m fittest solutions and calculate their centroid as the best solution, as given in Eq. 10 for islands utilizing BA. It has been statistically proven that there is a higher chance of finding an unknown optimal solution when initial solutions are closer to the center of the search space and this chance increases with increasing number of dimensions [36].

$$x^{best} = \frac{x_1^{best} + x_2^{best} + ... + x_m^{best}}{m} \tag{10}$$

Additionally, we have proposed a local search operator as explained in algorithm 1 for BA that helps in fast convergence by maintaining diversity as given in Eq. 11. Eq. 4 is replaced by Eq. 11 in algorithm 1 to avoid premature convergence.

$$x_{new} = x^{best} + \epsilon A(x^{best,n} - x_{k,n}) \tag{11}$$

Where k is a random number between $[0, N-1]$ and n is number between $[0, d-1]$ such that $k \neq n$ where d represents dimensions. The main steps of the DistSSB are given in algorithm 3. DistSSB creates RDD from the population of random solutions.

Algorithm 3 Distributed Scalable SHADE Bat

```
1:  Initialize Population (P), Migration Rate (M_r), Migration Interval (M_i)
2:  population = sc.parallelize (P)
3:  while stoppingCriteria do
4:      bestSolutions = population.mapPartitionsWithIndex { ( index, iterator ) {
5:          popData = loadData()
6:          popData = popData.eliminateWeakSolutions()
7:          popData = broadcastedSol.union(popData)
8:          for t : 1 to M_i do
9:              if index % 2 == 0 then
10:                 SHADE(popData)
11:                 best = selectRandom(popData.take(M_r))
12:             end if
13:             if index % 2 == 1 then
14:                 BA(popData)
15:                 best = findCentroid(popData.take(M_r))
16:             end if
17:         end for
18:         save.popData.iterator
19:     best.iterator
20:     } } . collect()
21:     broadcastedSol = sc.broadcast(bestSolutions)
22: end while
```

The code within mapPartitionsWithIndex (line 5-19) is evolved in a parallel fashion on multiple partitions. It is an

internal feature of Spark that returns a new RDD by applying a function to each partition in distributed manner. Within the parallel execution, half of the sub-populations (partitions) are evolved using BA (line 14-15) while others are evolved using SHADE (line 10-11). After each migration interval, the best solution(s) from each partition (line 19) are collected and placed in bestSolutions (line 4). bestSolutions are then broadcasted to other partitions (line 21). Each partition at the start of the next migration interval picks all the broadcasted solutions and uses them to replace its weakest solutions (line 6-7). This creates a mesh topology between the partitions. The evolved RDD (popData) with replaced and evolved solutions is not materialized throughout the DistSSB execution. This RDD remains in-memory until the next broadcast. In-memory execution reduces the network communication between worker and master nodes and decreases the overall execution time. The evolved population at each partition is saved temporarily on the local (partition) file system after every migration interval (line 18). The saved popData and bestSolutions are retrieved again at the start of the next migration interval (line 5). The algorithm terminates when stopping criteria is met.

V. Experimental Setup

To test and evaluate the performance of the DistSSB, we have used the Large Scale Global Optimization benchmark functions suite and the experimental conditions used in CEC-2013 [10]. CEC benchmark contains 15 functions with 1000 decision variables except for $F13$ and $F14$ where decision variables are 905 due to overlapping factors. These benchmark functions are divided into fully separable, partially separable, overlapping, and non-separable functions. We briefly discuss these functions below, and additional details can be found in [10], [26].

1) Fully separable functions: $(f1 - f3)$
2) Partially separable functions
   a) with a separable subcomponent: $(f4 - f7)$
   b) with no separable subcomponent: $(f8 - f11)$
3) Overlapping Functions: $(f12 - f14)$
4) Non-separable Functions: $(f15)$

We have tested all CEC-2013 benchmark functions upto 2,000 dimensions (2000D) while tested a few functions for $5.0 * 10^4$ dimensions. To verify the scalability of DistSSB we have evaluated $f12$ for 10^6 dimensions. Table I shows the parameter settings of DistSSB. Maximum number of function evaluations are $3.0 * 10^6$ and VTR is 0.0. All the results are reported as average of 25 executions against each configuration. In Table I parameters for BA (ϵ, r_i^0, α and γ) are initialized as in [22]

TABLE I
Parameter settings for DistSSB

Parameter	Value	Parameter	Value
Population size (NP)	10^3	ϵ	0.003
Number of Islands (I_n)	10	r_i^0	0.1
Migration Interval (M_i)	$2 * 10^4$	α , γ	0.95
Migration Rate (M_r)	1	CR, F	0.95, 0.55

TABLE II
DistSSB, GL-SHADE, and SHADE-ILS fitness comparison

f	Details	1000D			2000D		
		DistSSB	GL-SHADE	SHADE-ILS	DistSSB	GL-SHADE	SHADE-ILS
$f1$	Best	1.83E-05	3.07E-10	0.00E+00	4.48E-08	1.17E+00	4.76E-27
	Median	2.19E-02	5.38E-06	0.00E+00	4.06E-03	1.92E+00	6.66E-25
	Worst	4.79E-01	8.71E-05	8.81E-03	1.20E-01	4.68E+00	2.73E-23
	Mean	1.52E-01	1.32E-05	1.48E-03	3.00E-02	2.32E+00	5.85E-24
	StDev	2.08E-01	2.65E-05	2.94E-03	5.15E-02	1.43E+00	1.20E-23
	P-Value		2.12E-04$^-$	1.50E-03$^-$		**1.57E-04$^+$**	1.57E-04$^-$
$f2$	Best	4.69E+02	0.00E+00	9.08E+02	7.26E+02	1.49E-01	9.18E+02
	Median	5.07E+02	1.05E+00	1.09E+03	1.63E+03	2.83E-01	4.92E+03
	Worst	6.03E+02	4.97E+00	1.30E+03	8.53E+03	9.24E-01	6.77E+05
	Mean	5.18E+02	1.89E+00	1.11E+03	3.32E+03	4.13E-01	1.41E+05
	StDev	4.29E+01	2.15E+00	1.89E+02	3.34E+03	3.03E-01	3.00E+05
	P-Value		1.57E-04$^-$	**1.570E-04$^+$**		1.57E-04$^-$	**2.33E-02$^+$**
$f3$	Best	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.02E+01	2.03E+01
	Median	2.00E+01	2.01E+01	2.01E+01	2.00E+01	2.05E+01	2.07E+01
	Worst	2.02E+01	2.02E+01	2.02E+01	2.00E+01	2.05E+01	2.07E+01
	Mean	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.04E+01	2.06E+01
	StDev	7.04E-02	8.38E-02	6.07E-02	0.00E+00	1.64E-01	2.23E-01
	P-Value		**2.46E-04$^+$**	**4.59E-03$^+$**		**1.57E-04$^+$**	**3.41E-03$^+$**
$f4$	Best	2.54E+08	1.72E+08	8.90E+09	2.58E+09	1.50E+10	3.97E+10
	Median	4.09E+08	7.30E+08	1.73E+10	3.84E+09	2.42E+10	9.07E+10
	Worst	5.42E+08	1.06E+09	2.16E+10	6.26E+09	3.78E+11	1.68E+11
	Mean	3.99E+08	6.84E+08	1.62E+10	4.17E+09	9.36E+10	9.57E+10
	StDev	1.05E+08	2.74E+08	5.40E+09	1.50E+09	1.59E+11	4.60E+10
	P-Value		**1.56E-02$^+$**	**1.570E-04$^+$**		**1.57E-04$^+$**	**1.57E-04$^+$**
$f5$	Best	1.58E+06	1.04E+06	1.72E+06	2.49E+06	1.42E+06	1.71E+06
	Median	1.80E+06	1.17E+06	1.99E+06	5.23E+06	3.07E+06	4.18E+06
	Worst	2.86E+06	1.48E+06	2.23E+06	2.73E+07	1.04E+07	6.20E+06
	Mean	2.02E+06	1.18E+06	2.02E+06	9.19E+06	4.19E+06	4.22E+06
	StDev	3.73E+05	1.14E+05	2.25E+05	1.03E+07	3.57E+06	1.68E+06
	P-Value		1.57E-04$^-$	4.96E-01$^=$		1.51E-01$^-$	2.84E-02$^-$
$f6$	Best	1.00E+06	1.04E+06	1.02E+06	1.02E+06	1.01E+06	1.02E+06
	Median	1.01E+06	1.05E+06	1.04E+06	1.04E+06	3.58E+06	3.45E+06
	Worst	1.02E+06	1.06E+06	1.04E+06	1.07E+06	7.24E+06	6.54E+06
	Mean	1.01E+06	1.05E+06	1.04E+06	1.04E+06	3.48E+06	3.35E+06
	StDev	7.89E+03	7.38E+03	8.58E+03	1.552E+04	2.45E+06	2.02E+06
	P-Value		**1.57E-04$^+$**	**5.07E-04$^+$**		**3.61E-03$^+$**	**1.40E-02$^+$**
$f7$	Best	1.43E+04	4.90E+05	9.27E+06	2.40E+07	2.29E+08	8.24E+07
	Median	3.87E+04	5.49E+05	9.79E+06	3.77E+07	4.54E+08	2.92E+08
	Worst	6.78E+04	6.25E+05	2.49E+07	6.27E+07	7.06E+08	5.42E+08
	Mean	4.16E+04	5.47E+05	1.17E+07	4.07E+07	4.83E+08	3.08E+08
	StDev	1.95E+04	4.20E+04	4.78E+06	1.33E+07	1.94E+08	1.73E+08
	P-Value		**1.57E-04$^+$**	**1.57E-04$^+$**		**1.57E-04$^+$**	**1.57E-04$^+$**
$f8$	Best	2.84E+10	6.11E+12	4.04E+13	1.42E+13	5.27E+13	1.83E+14
	Median	5.16E+10	7.10E+12	3.39E+14	2.57E+13	1.93E+15	2.35E+15
	Worst	6.26E+10	7.40E+12	3.28E+14	3.85E+13	8.92E+15	6.71E+15
	Mean	5.26E+10	7.40E+12	3.28E+14	2.57E+13	1.93E+15	2.35E+15
	StDev	2.65E+10	9.65E+11	1.83E+14	8.65E+12	3.23E+15	2.42E+15
	P-Value		**1.57E-04$^+$**	**1.57E-04$^+$**		**1.57E-04$^+$**	**1.57E-04$^+$**
$f9$	Best	2.25E+08	2.65E+08	1.83E+08	5.71E+08	1.02E+09	4.65E+08
	Median	2.55E+08	3.55E+08	1.94E+08	6.77E+08	1.58E+09	6.79E+08
	Worst	2.70E+08	5.25E+08	2.20E+08	8.01E+08	4.26E+09	9.01E+08
	Mean	2.54E+08	3.65E+08	1.98E+08	9.13E+07	9.31E+07	9.27E+07
	StDev	1.44E+07	9.68E+07	1.55E+07	8.51E+07	1.23E+09	1.49E+08
	P-Value		**1.50E-03$^+$**	1.57E-04$^-$		**1.57E-04$^+$**	**6.70E-04$^+$**
$f10$	Best	9.05E+07	9.18E+07	9.15E+07	9.02E+07	8.60E+07	8.81E+07
	Median	9.06E+07	9.29E+07	9.24E+07	9.15E+07	9.29E+07	9.16E+07
	Worst	9.19E+07	9.08E+07	9.30E+07	9.20E+07	9.83E+07	9.98E+07
	Mean	9.07E+07	9.28E+07	9.24E+07	9.13E+07	9.31E+07	9.27E+07
	StDev	2.35E+05	5.68E+05	5.71E+05	6.33E+05	4.29E+06	4.01E+06
	P-Value		**1.57E-04$^+$**	**1.57E-04$^+$**		**5.21E-01$^+$**	**7.91E-01$^+$**
$f11$	Best	5.71E+06	9.16E+11	2.17E+08	9.49E+07	2.37E+12	1.08E+10
	Median	9.17E+06	9.32E+11	3.03E+08	4.58E+08	4.78E+12	3.04E+10
	Worst	1.70E+07	9.44E+11	4.13E+08	6.84E+08	7.86E+12	6.84E+10
	Mean	9.99E+06	9.31E+11	2.92E+08	4.07E+08	4.95E+12	3.40E+10
	StDev	3.95E+06	1.15E+10	5.29E+07	2.00E+08	1.79E+12	1.80E+10
	P-Value		**1.57E-04$^+$**	**1.57E-04$^+$**		**1.57E-04$^+$**	**1.57E-04$^+$**
$f12$	Best	1.81E+03	2.90E+01	2.18E+02	3.10E+03	9.27E+01	1.01E+03
	Median	2.06E+03	1.20E+02	5.95E+02	4.63E+03	3.53E+03	1.20E+03
	Worst	2.46E+03	2.52E+02	6.86E+02	5.82E+03	9.79E+03	4.65E+03
	Mean	2.09E+03	1.25E+02	4.96E+02	4.48E+03	4.31E+03	1.97E+03
	StDev	2.05E+02	7.78E+01	1.67E+02	9.98E+02	3.32E+03	1.35E+03
	P-Value		1.57E-04$^-$	1.57E-04$^-$		4.06E-01$^=$	1.94E-03$^-$
$f13$	Best	4.05E+06	4.13E+08	4.47E+08	1.91E+09	1.63E+10	1.64E+09
	Median	6.61E+06	4.44E+08	5.20E+09	3.19E+09	3.18E+10	8.45E+09
	Worst	8.98E+06	4.69E+08	5.50E+09	4.70E+09	5.38E+10	9.98E+09
	Mean	6.63E+06	4.42E+08	3.82E+09	3.22E+09	3.29E+10	7.37E+09
	StDev	1.4E+06	1.62E+07	2.12E+09	8.27E+08	1.32E+10	2.95E+09
	P-Value		**1.57E-04$^+$**	**1.57E-04$^+$**		**5.06E-03$^+$**	**3.64E-01$^+$**
$f14$	Best	6.63E+06	5.58E+07	1.13E+10	2.58E+07	2.52E+10	2.76E+10
	Median	7.58E+06	8.78E+07	1.68E+10	3.08E+08	4.35E+10	4.97E+10
	Worst	8.56E+06	9.03E+07	2.35E+10	4.26E+08	7.29E+10	8.31E+10
	Mean	7.60E+06	7.94E+07	1.66E+10	2.27E+08	4.57E+10	5.34E+10
	StDev	6.96E+05	1.45E+07	5.24E+09	1.76E+08	1.74E+10	2.03E+10
	P-Value		**1.57E-04$^+$**	**1.57E-04$^+$**		**1.57E-04$^+$**	**1.57E-04$^+$**
$f15$	Best	2.89E+06	1.01E+07	7.28E+06	1.53E+07	1.93E+08	8.54E+06
	Median	3.09E+06	1.37E+07	8.85E+06	3.29E+07	3.19E+08	7.02E+07
	Worst	3.38E+06	1.59E+07	1.67E+07	7.17E+07	5.38E+08	9.81E+07
	Mean	3.09E+06	1.35E+07	1.04E+07	3.81E+07	3.58E+08	6.15E+07
	StDev	1.46E+05	1.99E+06	3.40E+06	2.05E+07	1.32E+08	3.14E+07
	P-Value		**1.57E-04$^+$**	**1.57E-04$^+$**		**1.57E-04$^+$**	**3.26E-01$^+$**
+ / = / -			11 / 0 / 4	11 / 1 / 3		12 / 1 / 2	12 / 0 / 3

while two control parameters (CR, F) of SHADE are set as suggested in [5]. All the DistSSB experiments are performed on a three-node cluster: DELL PowerEdge R815, 2x AMD Opteron 6376 (64 Cores), 256 GB. RAM, 3 TB SATA RAID-5 with spark-2.1.0 and Scala 2.11.8. GL-SHADE and SHADE-ILS experiments have been performed on Google Cloud (12GB RAM) that is GPU enabled.
The source code[1] of DistSSB using Scala and Apache Spark framework is available on github.

A. Optimization

The optimization results of DistSSB are compared with GL-SHADE and SHADE-ILS. For 1000D, DistSSB has obtained better convergence as compared to SHADE-ILS for eleven functions and performed equivalent on one function while SHADE-ILS performed better for $f1, f9, \& \ f12$. When compared to GL-SHADE, DistSSB performed better for eleven functions, while GL-SHADE performed better on $f1, f2, f5, \& \ f12$.

For 2000D, DistSSB's performance improved and it converged better than SHADE-ILS on twelve functions, while SHADE-ILS is better for $f1, f5, \& \ f12$. In comparison to GL-SHADE, DistSSB performed better on twelve functions, equal on one function while GL-SHADE performed better for $f2, \& \ f5$. Table II shows a comparison of DistSSB, GL-SHADE and SHADE-ILS on 1000D and 2000D with a maximum of $3 * 10^6$ function evaluations.

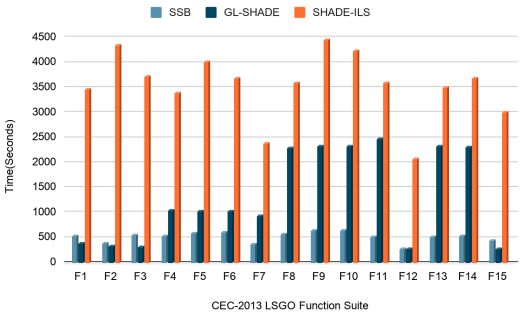

Fig. 1. Execution time of DistSSB, GL-SHADE, and SHADE-ILS on 1000D

This can be observed that DistSSB has performed better for majority of functions on high dimensions. We have used Wilcoxon rank sum test to analyze the accuracy of results using significance level of 0.05. The last row of Table II indicates the number of functions for which DistSSB is better $+$, worse $-$, or equal $=$ to the compared algorithms. Table III, shows the average ranking of algorithms based on friedman statistical ranking test. DistSSB has the smallest mean rank that reflects DistSSB is better than GL-SHADE and SHADE-ILS.

[1]https://github.com/HajiraJabeen/SparkOP

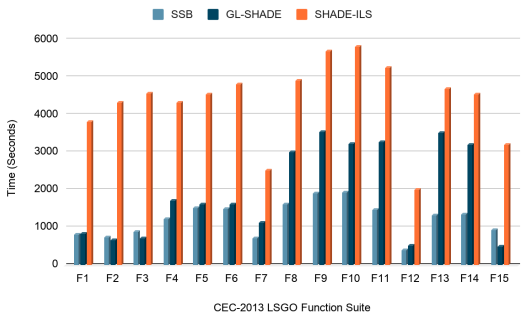

Fig. 2. Execution time of DistSSB, GL-SHADE, and SHADE-ILS on 2000D

B. Execution Time of DistSSB

The execution time of SHADE-ILS, GL-SHADE, and DistSSB is measured using system clock time. Time comparison is shown in Figure. 1 and Figure. 2 for 1000D, and 2000D respectively. DistSSB outperformed SHADE-ILS on all functions for both dimensions. While comparing with GL-SHADE for 1000D, DistSSB performed better on ten functions, equal for one function while GL-SHADE performed better on $f1$, $f2$, $f3$ & $f15$, while equal for $f12$. In case of 2000D DistSSB performed better than GL-SHADE on twelve functions while GL-SHADE performed better for $f2, \ f3, \ \& \ f15$. Here it is pertinent to mention that by increasing dimensions from 1000 to 2000, DistSSB improves in execution time for $f1, \ \& \ f12$ over GL-SHADE.

C. Scalability of DistSSB

We have tested the scalability of DistSSB on higher dimensions. On 50,000D results show that DistSSB converged faster for $f2$, $f3$, & $f15$ functions, whereas the execution time of GL-SHADE was better for these functions on lower dimensions. Figure. 3 shows the execution time of SHADE-ILS, GL-SHADE and DistSSB on $f2, \ f3, \& f15$ for 50000D. To check the scalabilty of DistSSB experiments are performed on one million dimensions for overlapping function $f12$, which is rather simple and does not involve a rotation matrix (to reduce the overall complexity). We also tested the scalability over one million dimensions. The results in Table IV show that Shade-ILS and GL-SHADE resulted in memory-outage, while DistSSB and proves to be scalable and suitable for high dimensional problems.

D. Effect of Islands

We have tested the behaviour of DistSSB using 10, 15, and 20 islands for $f3$, against 50000D. The results are shown in Figure. 4. Increasing the number of islands improves data parallelism across partitions but this decreases diversity at each partition and decreases the chances of finding optima during a migration interval. This is the reason that $f3$ has got better optimization value on 10 islands. On the other hand, execution time decreases with the increase in the number of islands due to the increased parallelism. Execution time of DistSSB for $f1$,$f2$,$f3$,$f12$, and $f15$ against 50000D and varying islands

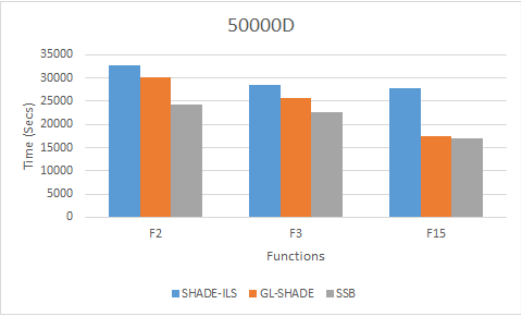

Fig. 3. Time comparison of $f2$, $f3$, & $f15$ at 50000D

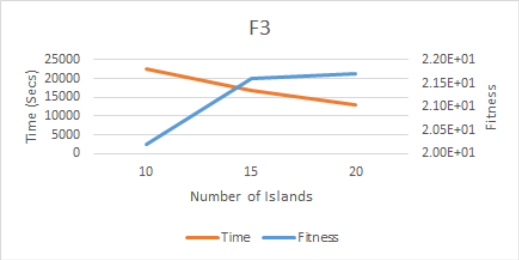

Fig. 4. Behaviour of DistSSB, on 10,15 and 20 islands on $f3$ against 50000D

also support the argument as shown in Figure. 5. The number of islands may be carefully decided keeping in view the trade off between execution time and optimization value.

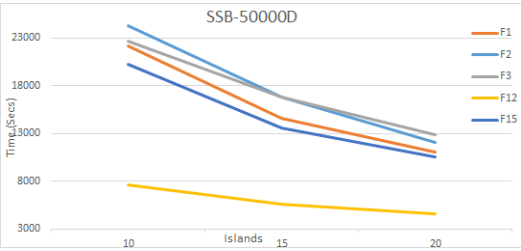

Fig. 5. Execution time of DistSSB for 50000D

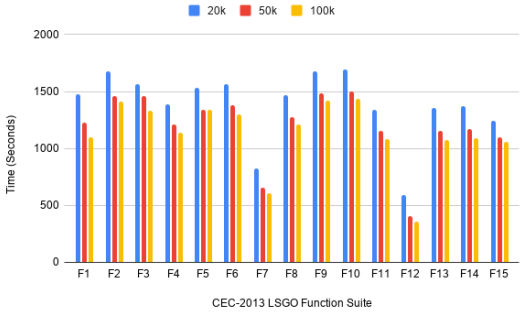

Fig. 6. Execution time of DistSSB for 20k, 50k and 100k migration interval on 1000D

TABLE III
ALGORITHMS RANKING USING FRIEDMAN STATISTICAL RANKING

Dimensions	DistSSB	GL-SHADE	SHADE-ILS
1000	1.50	1.93	2.56
2000	1.46	2.26	2.26

E. *Effect of Migration Interval*

Migration interval of 20K, 50K, and 100K has been used to study the impact on optimization value and execution time. Table V shows that decreasing migration interval results in improved optimization value in most of the cases. Once solutions at a partition are stuck in local optima, then more iterations will be wasted in case of higher migration intervals, hence effecting the overall convergence towards global optima. On the other hand higher migration interval improves the execution time as shown in Figure. 6 and Figure. 7. This is due to the reason that higher migration interval results in reduced network overhead for constant island size.

TABLE IV
FITNESS VALUE OF $f12$ ON 10^6 DIMENSIONS

DistSSB	GL-SHADE	SHADE-ILS
2.73E+01	Out of memory exception	Unable to allocate memory

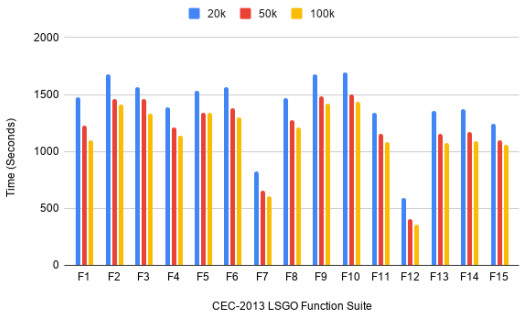

Fig. 7. Execution time of DistSSB for 20, 50 & 100k migration interval on 2000D

TABLE V
AVERAGE FITNESS COMPARISON OVER 25 RUNS USING 10^7 ITERATIONS, 1 MIGRATION RATE, AND 10 ISLANDS

f	1000D			2000D		
	M_i: 20k	M_i: 50k	M_i: 100k	M_i: 20k	M_i: 50k	M_i: 100k
$f1$	4.13E-18	1.21E-20	**1.58E-25**	1.03E+00	1.34E-08	**6.77E-18**
$f2$	1.25E+03	6.74E+02	**3.67E+02**	1.15E+04	1.96E+03	**1.40E+03**
$f3$	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01
$f4$	1.18E+08	2.19E+08	**7.68E+07**	**1.83E+09**	3.18E+09	3.67E+09
$f5$	3.66E+06	**2.95E+06**	3.39E+06	**8.24E+06**	7.76E+06	9.15E+06
$f6$	**9.98E+05**	9.99E+05	1.03E+06	1.04E+06	1.05E+06	1.05E+06
$f7$	**4.16E-05**	1.76E-03	1.01E-01	**1.70E+05**	5.52E+05	3.48E+06
$f8$	**1.47E+09**	3.76E+09	2.63E+09	**1.26E+12**	2.12E+12	1.41E+12
$f9$	2.64E+08	2.70E+08	**2.43E+08**	**6.07E+08**	7.07E+08	6.63E+08
$f10$	**9.05E+07**	9.08E+07	9.26E+07	9.18E+07	**9.05E+07**	9.06E+07
$f11$	9.25E+05	**1.70E+06**	2.77E+06	**1.57E+08**	2.11E+08	4.10E+08
$f12$	2.16E+03	2.15E+03	**2.05E+03**	7.94E+04	3.45E+05	**7.27E+04**
$f13$	**3.40E+05**	6.66E+05	1.79E+06	**1.35E+08**	1.41E+08	1.06E+09
$f14$	**7.06E+06**	8.29E+06	8.37E+06	**1.64E+08**	1.74E+08	2.99E+08
$f15$	**4.90E+05**	1.04E+06	1.40E+06	**6.83E+06**	1.42E+07	1.91E+07

VI. Conclusion and Future Work

Various evolutionary techniques have been proposed to solve LSGO problems. However, the evaluations of these methods have remained limited. While the problem sizes are increasing with time, scalable and distributed techniques are required to solve ever-growing LSGO problems. In this paper, we have proposed DistSSB, a scalable and distributed technique to solve LSGO problems. DistSSB uses Apache Spark for scalability and offers efficient exploration and exploitation using SHADE and BAT algorithms. We have compared the performance of DistSSB for varying number of dimensions over functions of different complexity with two state-of-the-art algorithms GL-SHADE and SHADE-ILS. DistSSB outperformed both algorithms in scalability while delivering comparable results. We have tested DistSSB for up to one million dimensions, while the other two algorithms failed to scale. It should be noted that the execution time of DistSSB remains reasonable for a million dimensions. In the future, we want to investigate local search methodologies to augment the performance of DistSSB and search for the optimal island size in comparison to the problem size. We also want to apply DistSSB to machine learning parameter optimization problems in the future.

References

[1] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[2] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4. IEEE, 1995, pp. 1942–1948.

[3] J. Holland, "Adaptation in natural and artificial systems: an introductory analysis with application to biology," *Control and artificial intelligence*, 1975.

[4] S. Chen, J. Montgomery, and A. Bolufé-Röhler, "Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution," *Applied Intelligence*, vol. 42, no. 3, pp. 514–526, 2015.

[5] D. Molina, A. LaTorre, and F. Herrera, "Shade with iterative local search for large-scale global optimization," in *2018 IEEE congress on evolutionary computation (CEC)*. IEEE, 2018, pp. 1–8.

[6] S. Mahdavi, S. Rahnamayan, and M. E. Shiri, "Multilevel framework for large-scale global optimization," *Soft Computing*, vol. 21, no. 14, pp. 4111–4140, 2017.

[7] A. W. Mohamed, "Solving large-scale global optimization problems using enhanced adaptive differential evolution algorithm," *Complex & Intelligent Systems*, vol. 3, no. 4, pp. 205–231, 2017.

[8] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A survey on knowledge graphs: Representation, acquisition and applications," *arXiv preprint arXiv:2002.00388*, 2020.

[9] "Keras," https://keras.io/api/applications/, accessed: 2021-04-22.

[10] X. Li, K. Tang, M. N. Omidvar, Z. Yang, K. Qin, and H. China, "Benchmark functions for the cec 2013 special session and competition on large-scale global optimization," *gene*, vol. 7, no. 33, p. 8, 2013.

[11] J. Blanchard, C. Beauthier, and T. Carletti, "A surrogate-assisted cooperative co-evolutionary algorithm using recursive differential grouping as decomposition strategy," in *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2019, pp. 689–696.

[12] N. R. Sabar, A. Turky, and A. Song, "Adaptive multi-optimiser cooperative co-evolution for large-scale optimisation," in *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2019, pp. 705–712.

[13] Y.-J. Gong, W.-N. Chen, Z.-H. Zhan, J. Zhang, Y. Li, Q. Zhang, and J.-J. Li, "Distributed evolutionary algorithms and their models: A survey of the state-of-the-art," *Applied Soft Computing*, vol. 34, pp. 286–300, 2015.

[14] K. Doerner, R. F. Hartl, and M. Reimann, "Cooperative ant colonies for optimizing resource allocation in transportation," in *Workshops on Applications of Evolutionary Computation*. Springer, 2001, pp. 70–79.

[15] Y.-H. Chang, "Adopting co-evolution and constraint-satisfaction concept on genetic algorithms to solve supply chain network design problems," *Expert Systems with Applications*, vol. 37, no. 10, pp. 6919–6930, 2010.

[16] L. M. Antonio and C. A. C. Coello, "Indicator-based cooperative coevolution for multi-objective optimization," in *2016 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2016, pp. 991–998.

[17] C. Tan, C. K. Goh, K. C. Tan, and A. Tay, "A cooperative coevolutionary algorithm for multiobjective particle swarm optimization," in *2007 IEEE Congress on Evolutionary Computation*. IEEE, 2007, pp. 3180–3186.

[18] J. Zhang and A. C. Sanderson, "Jade: adaptive differential evolution with optional external archive," *IEEE Transactions on evolutionary computation*, vol. 13, no. 5, pp. 945–958, 2009.

[19] A. A. Hadi, A. W. Mohamed, and K. M. Jambi, "Lshade-spa memetic framework for solving large-scale optimization problems," *Complex & Intelligent Systems*, vol. 5, no. 1, pp. 25–40, 2019.

[20] R. Tanabe and A. Fukunaga, "Evaluating the performance of shade on cec 2013 benchmark problems," in *2013 IEEE Congress on evolutionary computation*. IEEE, 2013, pp. 1952–1959.

[21] X.-S. Yang, "A new metaheuristic bat-inspired algorithm," in *Nature inspired cooperative strategies for optimization (NICSO 2010)*. Springer, 2010, pp. 65–74.

[22] M. A. Al-Betar and M. A. Awadallah, "Island bat algorithm for optimization," *Expert Systems with Applications*, vol. 107, pp. 126–145, 2018.

[23] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[24] A. L. Corcoran and R. L. Wainwright, "A parallel island model genetic algorithm for the multiprocessor scheduling problem," in *Proceedings of the 1994 ACM symposium on applied computing*, 1994, pp. 483–487.

[25] H. Wang, S. Rahnamayan, and Z. Wu, "Parallel differential evolution with self-adapting control parameters and generalized opposition-based learning for solving high-dimensional optimization problems," *Journal of Parallel and Distributed Computing*, vol. 73, no. 1, pp. 62–73, 2013.

[26] X. Zhang, Z.-H. Zhan, and J. Zhang, "Adaptive population differential evolution with dual control strategy for large-scale global optimization problems," in *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2020, pp. 1–7.

[27] D. Fan and J. Lee, "A hybrid mechanism of particle swarm optimization and differential evolution algorithms based on spark," *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 13, no. 12, pp. 5972–5989, 2019.

[28] F. Maqbool, S. Razzaq, J. Lehmann, and H. Jabeen, "Scalable distributed genetic algorithm using apache spark (s-ga)," in *International Conference on Intelligent Computing*. Springer, 2019, pp. 424–435.

[29] M. AlJame, I. Ahmad, and M. Alfailakawi, "Apache spark implementation of whale optimization algorithm," *Cluster Computing*, vol. 23, no. 3, pp. 2021–2034, 2020.

[30] Z. He, H. Peng, J. Chen, C. Deng, and Z. Wu, "A spark-based differential evolution with grouping topology model for large-scale global optimization," *Cluster Computing*, pp. 1–21, 2020.

[31] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for differential evolution," in *2013 IEEE congress on evolutionary computation*. IEEE, 2013, pp. 71–78.

[32] O. Pacheco-Del-Moral and C. A. C. Coello, "A shade-based algorithm for large scale global optimization," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2020, pp. 650–663.

[33] M. Yang, A. Zhou, C. Li, and X. Yao, "An efficient recursive differential grouping for large-scale continuous problems," *IEEE Transactions on Evolutionary Computation*, 2020.

[34] Y. Sun, X. Li, A. Ernst, and M. N. Omidvar, "Decomposition for large-scale optimization problems with overlapping components," in *2019 IEEE congress on evolutionary computation (CEC)*. IEEE, 2019, pp. 326–333.

[35] S. Rahnamayan and G. G. Wang, "Center-based sampling for population-based algorithms," in *2009 IEEE Congress on Evolutionary Computation*. IEEE, 2009, pp. 933–938.

[36] S. Rahnamayan. and G. G. Wang, "Toward effective initialization for large-scale search spaces," *Trans Syst*, vol. 8, no. 3, pp. 355–367, 2009.