Extremely High-dimensional Optimization with MapReduce: Scaling Functions and Algorithm

Alberto Cano[a], Carlos García-Martínez[b], Sebastián Ventura[b,*]

[a]*Department of Computer Science, Virginia Commonwealth University, USA*
[b]*Department of Computer Science and Numerical Analysis, University of Cordoba, Spain*

Abstract

Large scale optimization is an active research area in which many algorithms, benchmark functions, and competitions have been proposed to date. However, extremely high-dimensional optimization problems comprising millions of variables demand new approaches to perform effectively in results quality and efficiently in time. Memetic algorithms are popular in continuous optimization but they are hampered on such extremely large dimensionality due to the limitations of computational and memory resources, and heuristics must tackle the immensity of the search space. This work shows how the MapReduce parallel programming model allows the addressing of problems with millions of variables, and presents an adaptation of the MA-SW-Chains algorithm to the MapReduce framework. Benchmark functions from the IEEE CEC 2010 and 2013 competitions are considered and results with 1, 3 and 10 million variables are presented. MapReduce shows to be an effective approach to scale optimization algorithms on extremely high-dimensional problems, taking advantage of the combined computational and memory resources distributed in a computer cluster.

Keywords: Real optimization, high-dimensional optimization, MapReduce

*Corresponding author
Email addresses: acano@vcu.edu (Alberto Cano), cgarcia@uco.es (Carlos García-Martínez), sventura@uco.es (Sebastián Ventura)

1. Introduction

Nowadays, researchers and engineers often face optimization problems with considerable amounts of decision variables. High-dimensional optimization has become an attracting field with associated special sessions in conferences [37, 58] and special issues in international journals [41], resulting in an increasing number of publications over the last decade.

Large scale global/continuous optimization (LSGO) [41, 50] is the research field aimed at developing effective and efficient approaches for large-scale optimization of real-parameter problems. The IEEE Congress on Evolutionary Computation (CEC) organizes dedicated competitions in a regular basis since 2008, where the state-of-the-art approaches are compared upon an agreed evaluation framework with standardized functions. Immense search spaces, complex function characteristics, and demanding computational requirements are some of the issues that approaches have to effectively address to get relatively good results. IEEE CEC competitions evaluate the performance of algorithms on a set of functions with up to 1,000 dimensions. Recently, researchers focused on increasing the dimensionality of the benchmark functions to up to 3 million dimension variables [32], which gives way to address the challenge of designing algorithms for extremely high-dimensional problems having more than 1 million variables. However, classical algorithms applied to extremely high-dimensional problems are limited by runtime, and computational and memory resources. Only very few attempts have been made by nature-inspired optimization algorithms on problems with millions of variables [13, 38, 54, 67].

MapReduce [11] is a powerful programming model for developing scalable and fault tolerant applications for computational and data intensive processing tasks. In contrast with other computational distributed frameworks, MapReduce scales well on commodity clusters with low economic costs. Nowadays, MapReduce is used in many applications areas, including searching, singular value decomposition, inverted index construction, machine learning, and problem optimization, among others [12]. However, to our knowledge, there are not studies regarding its application for LSGO.

In this work, we fill the gap, providing some results on MapReduce-based LSGO with extremely high-dimensional problems. Concretely, our main contributions are:

- We provide scalable MapReduced versions of CEC 2010 and 2013 benchmark functions, based on their decomposable properties. The dimen-

sionality of these functions can grow indefinitely according to the computational resources, and experiments are carried out with up to 10 million variables.

- MapReduce has been used for the first time, to our knowledge, for the parallel computation of the evaluation of single solutions in LSGO. To that end, the original objective function is written as the reduce-based aggregation of independent map-based subfunctions.

- We have designed a MapReduce version of a competitive model for LSGO. We have addressed the exploitation of available computational resources when the original model carried out sequential operations. Particularly, 1) the local optimization of a single solution is divided into multiple independent optimizations of variable subgroups, which share their progresses in a cooperative coevolution fashion; and 2) the steady-state evolution of a medium-sized population, in contrast with most MapReduce-based generational models with large population sizes, is partitioned into the independent evolution of several subpopulations.

The remainder of the article is structured as follows. Section 2 overviews the background of this study. Section 3 presents the MapReduce model for the benchmark functions and the algorithm proposal. Section 4 shows and discusses the experimental results. Finally, Section 5 draws the conclusions of this work.

2. Background

This section depicts the framework associated with our study. Section 2.1 offers a short look at the progresses on LSGO. Section 2.2 overviews the application of MapReduce technology in evolutionary computation. Section 2.3 describes the MA-SW-Chains algorithm [47], a popular method for LSGO.

2.1. Large Scale Global Optimization

High-dimensional problems are difficult to address due to numerous factors. First, the search space grows exponentially as the number of dimensions increases. The curse of dimensionality makes it difficult to lead effectively the optimization process. Second, the properties of the search space may change as the dimensionality grows up, e.g., multimodal functions. Moreover, the interaction between non-separable variables hardens the difficulty. Third,

the evaluation of solutions demands increasing computational resources and runtime. Eventually, the memory becomes full or the runtime is out of scale.

Approaches for LSGO come from different families and their combinations [42], differential evolution [20, 57], estimation distribution algorithms [40, 65], ant colony optimization [30], particle swarm optimization [10, 18], and memetic algorithms [34, 47], among others. Particularly, hybrid algorithms combining a population-based strategy with a local search operator have always reached most of the best positions in recent competitions:

- In CEC 2008 competition, the winner consisted in a multi-agent strategy, MTS [60], where each agent did an iterated local search using one out of three local search operators.

- In CEC 2010 competition, the general best approach was MA-SW-Chains [47], a memetic algorithm incorporating local search chains with the Solis and Wets' method; and the third one, DMS-PSO-SHS [70], combined particle swarm optimization, harmony search, and a modified MTS local search operator among others.

- In the special issue of the Soft Computing Journal 2011 [41], the best approach combined differential evolution and the MTS-LS1 local search into a multiple offspring framework, MOS-SOCO2011 [33].

- In CEC 2012 competition, the second best approach applied a differential evolution technique and a local search procedure on the best solution, jDEsps [7].

- In CEC 2013 competition, the winner was a MOS variant with a genetic algorithm and two local search operators, MOS-CEC2013 [34].

- In CEC 2015 competition, MOS was the winner again and the second best algorithm was based in an iterative hybridization of differential evolution with local search [45].

Recently, LaTorre et al. [35] compared the best algorithms of the mentioned competitions. Their results show that MOS-CEC2013 generally obtains the best results on a testbed including CEC 2010, Soft Computing special issue 2011, and CEC 2013 benchmarks, and MA-SW-Chains ranks second for the CEC 2013 functions.

Cooperative coevolution [53] is another strategy that has gained some relevance on *large-scale optimization*, not necessarily LSGO. The main idea is to decompose the optimization of high-dimensional problems into the decomposed optimization of smaller subcomponents. Afterwards, the solutions of subcomponents are combined to construct a complete solution for the high-dimensional problem. Its assumption is that, in numerous problems with abundant variables, there exist variables subgroups with strong intradependencies, while subgroups interdependencies are weaker. This group is known as *partially separable problems* [6, 36], which widely appear in real-world large-scale applications [2, 3, 9, 22, 29, 44, 51, 66, 71]. In contrast, fully-separable problems contain subgroups with just one variable and no interdependencies, and there is only one group comprising all the variables with complex dependencies among each other in fully-non-separable ones (the rank of the Hessian matrix of the problem is equal to its dimensionality). The goal of cooperative coevolution, especially in partially separable problems, is to detect the subgroups of dependent variables precisely, so the aggregation of the subsequent results for the subgroups produces good global solutions.

Most cooperative coevolution strategies for large scale optimization are divided into static and dynamic grouping methods, and this latter into random and learning-based ones [42]. Given that detecting independent groups of variables may be difficult in some cases and impossible in non-separable problems, dynamic grouping has received greater attention. Mahdavi et al. [42] provide a comprehensive survey on cooperative coevolution algorithms for large scale optimization, with instances of differential evolution [49, 68] and particle swarm optimization [4, 39], among others. Several examples worth to be mentioned are MLCC [69], ranked fourth in CEC 2008 competition, 2S-Ensemble [65], second best method in CEC 2010 competition, and DECC-G [68] and CC-CMA-ES [40], second and third best algorithms in CEC 2013 competition, respectively.

2.2. MapReduce and Evolutionary Computation

MapReduce [11, 12] is a relatively new programming model that facilitates the parallelization of large-scale computational and data intense applications over a large number of computers, mainly characterized by being scalable and fault-tolerant. Although Hadoop [56] is the most popular disk-based MapReduce implementation (written in Java), the MapReduce model may be implemented in any other distributed platforms and languages such as

MPI [23]. The programmer has to provide *map* and *reduce* functions according to the following specification,

$$map(k_1, v_1) \rightarrow list(k_2, v_2)$$
$$reduce(k_2, list(v_2)) \rightarrow list(v_3)$$

and the framework manages the parallel execution of *map* and *reduce* jobs. Map jobs or *mappers*[1] receive a key/value pair and produce a list of intermediate key/value pairs. Each map job is independent from the others, so they can be distributed across the available computers. Once mappers have finished, reducers are given a list with all the values associated with the same given key k_2 and return normally a smaller list of values. Reduce jobs are also independent from the others and can be executed in parallel.

Given the success on data-intensive processing applications, MapReduce is being explored for other contexts, such as artificial intelligence [25, 28]. Particularly, several researchers have proposed evolutionary computation and swarm optimization models benefiting from this paradigm. To our knowledge, these proposals lie in one of the following groups:

Evaluation-concurrent algorithms. Most population-based algorithms generate a moderate or large set of new solutions per iteration. Given that the evaluation of each of these solutions is independent from those of the others and that the evaluation function has always been considered computationally expensive, it becomes an excellent candidate for parallelization in the MapReduce framework [1, 17, 27, 72]. In this kind of algorithms, each invocation of the map function receives a solution to be evaluated, usually the key is an identifier of the solution and the value contains the encoded solution; it computes the objective value of the solution, and outputs the solution along with its evaluation. The rest of the computation, such as crossover operation and selection in evolutionary algorithms, is carried out sequentially by a coordinator or a single reducer.

Solution-generation-concurrent algorithms. Approaches in this group parallelize the generation of new candidate solutions, besides their evaluation. Each iteration of the algorithm is implemented as a MapReduce job where

[1]In this work, we refer with mapper (reducer) to a virtual process that executes the map (reduce) function, not to be confounded with the real computation resources where this process is executed on.

evolution tasks are distributed appropriately to mappers and reducers. The most common approach is to implement the evaluation in the map function and make reducers to apply the selection, crossover and mutation operations to generate a new solution each [19, 26, 63]. This is a natural task distribution because selection operator needs that previous solutions have been evaluated, i.e., all the mappers had finished before applying selection, with a few exceptions [62]. The main key is that evolution operators are designed to make each application independent from the others, and thus, reducers can generate a new candidate solution each, concurrently. One advantage of these models is the possibility of using very large population sizes, because the number and overhead of non-concurrent operations are limited to the minimum.

McNabb et al. [43] presented another model adapting particle swarm optimization to the MapReduce framework. In this case, given that each particle movement is independent from others, once the global best position has been determined, the generation of a new candidate solution is carried out by mappers. Reducers just update their personal and global best positions.

Population-evolution-concurrent algorithms. Distributed algorithms divide the population into several sub-populations which evolve independently from each other, but migrations occur in some occasions. Therefore, they can be executed in a MapReduce framework where each sub-population evolves into a MapReduce job and migration relies the communication through the distributed file system [16, 52]. One advantage of these models is that each MapReduce job may evolve the sub-populations through several generations, reducing the network and I/O overhead of the framework.

Particular approaches. There are some other models that are hardly described according to the characteristics of previous groups. As an example, the extended compact genetic algorithm in [64] executes several MapReduce jobs per iteration, and thus algorithm steps (evaluation, selection, marginal probabilities computation, model construction and model sampling), are distributed into mappers and reducers with a finer granularity.

It is interesting to point out that, to our knowledge, all the proposals generate many new candidate solutions per iteration, because they are either generational evolutionary algorithms or models based on swarms of partially independent agents. On the contrary, our base model, MA-SW-Chains, generates just one solution per iteration. This will be adapted for a high exploitation of the available computational resources.

Algorithm 1 Pseudocode algorithm of the MA-SW-Chains.

1: Generate the initial population
2: Perform the steady-state GA
3: $S_{LS} \leftarrow$ Build the set of individuals potentially refined by LS
4: $C_{LS} \leftarrow$ Pick the best individual in S_{LS}
5: **if** C_{LS} belongs to an existing LS chain **then**
6: Initialize LS with the state in C_{LS}
7: **else**
8: Initialize LS with the default LS state
9: **end if**
10: $R_{LS} \leftarrow$ Apply SW to C_{LS} to improve the individual
11: Replace C_{LS} by R_{LS} in the GA population
12: Store the LS state in R_{LS}
13: **if** (not termination-condition) **then** go to step 2

2.3. MA-SW-Chains

The idea of memetic algorithms with local search chains was firstly introduced in 2010 with MA-LSCh-CMA [46]. This algorithm consisted in a steady-state genetic algorithm, which applied CMA-ES [24] as a local search operator. Given the costly adjustment of the internal parameters of each local search invocation, MA-LSCh-CMA stored their values together with the resulting solutions. Then, a new invocation of the local search method on these solutions would restore the corresponding internal parameter values, *chaining* the successive invocations to simulate a single but longer execution. MA-LSCh-CMA was tested on the benchmark of the CEC 2005 competition on real-parameter optimization, with up to 50 variables, providing very competitive results. However, the high computational requirements of CMA-ES for larger problems make it unusable as a local search for LSGO, at least for this model [47]. In CEC 2010 LSGO competition, Molina et al. presented a new version of the method, named MA-SW-Chains [47], that applied Solis Wets' algorithm instead of CMA-ES, and the same chaining mechanism on the new local search parameters. This algorithm obtained the best general results of the competition, and recently, it has been ranked second for CEC 2013 LSGO functions, with regards to the best algorithms of these competitions [35]. MA-SW-Chains (shown in Algorithm 1) is a steady-state memetic algorithm which comprises a steady-state genetic stage plus the Solis Wets' local search with the following special characteristics:

- It employs the idea of local search chain to adjust the computational efforts with which candidate solutions were optimized. This is implemented by encoding and updating the local search parameter values into the solution representation. The mechanism is particularly useful when applying intense local search operators that adapt their strategic decisions with the aim of increasing the probability of generating even better solutions.

- The algorithm favours the enlargement of local search chains on promising solutions that can be further optimized, and activate innovative chains when better solutions are hardly locally enhanced. To do that, it constructs the set of solutions without local optimization failure associated, and selects the best one for enhancement.

- It ensures a fixed and predetermined distribution of the computational resources between the exploitation of the local search and the exploration of the genetic algorithm. That is carried out by evolving the genetic algorithm several generations per invocation of the local search operator. Note that each generation of the steady-state genetic algorithm only produces just one new candidate solution, but one invocation to the local search consumes many evaluations.

After its success in 2010, MA-SW-Chains has been principally enhanced in three different directions:

- In [48], Solis Wets algorithm explores a random subset of variables instead of all of them. The idea is that it is very difficult to find the right direction for improvement in high-dimensional problems, so reducing the dimensionality often increases the probability of successful local search invocations. The new algorithm is called MA-SSW-Chains after the new Subgrouping Solis Wets (SSW) method. Note the similarity of the model with cooperative coevolution strategies [34].

  Subgrouping Solis Wets creates a new individual using a vector of differences randomly generated around a small area from the initial solution. The method does not explore all variables at the same time, but a random subset of consecutive variables at each time. For each number of evaluations, the subset of variables to be improved is changed. The same individual could be improved several times by the LS method, using several groups of variables at each time. The number of variables

to be changed is set to 20% of the number of solution components. The optimization is lead by the bias array that determines the direction, and the step size (constant for all variables), calculated by default as the half of the distance to the nearest neighbor and updated during the local search process.

- In [31], Lacroix et al. introduce a new niching strategy to maintain higher diversity in the population of the memetic algorithm. The search space is divided into adaptive hypercubes with room for just one solution each.

- In [32], the availability of powerful graphical processing units (GPU) encouraged Lastra et al. to design a GPU-based version of the algorithm capable of addressing extremely high-dimensional problems with up to 3 million variables, in contrast to the problems addressed in LSGO competitions with up to 1,000 variables. The main problem with the growth of the dimensionality is that both the computation time and the memory requirements increase. Eventually, algorithms cannot run within reasonable time nor there is memory to allocate the population with very large sizes. Morever, the data structures to store the local search states require additional memory. The GPU-based implementation by Lastra et al. is limited by the GPU's memory capacity.

In our work, the sequential fashion of MA-SW-Chains on iteratively improving a single solution is extended to a parallel environment generating multiple solutions per iteration. Our proposal takes advantage of the MapReduce paradigm and the available distributed computational resources. This allowed us to improve the results quality on 1 and 3 million of variables, as well as on 10 million variables, such extremely high dimensionality has never been addressed to date by any other LSGO algorithm to the best of our knowledge.

3. MapReduce for Large Scale Global Optimization

This section presents the MapReduce approach to scale benchmark functions to extremely high dimensionalities and the proposal of an algorithm that takes advantage of the MapReduce programming model to distribute computation based on a cooperative coevolution adaptation of the MA-SW-Chains.

3.1. Scaling Benchmark Functions with MapReduce

Real optimization benchmarks are usually minimization functions whose expressions are known. The IEEE CEC 2013 Special Session and Competition on LSGO defined a set of 15 functions which introduced nonuniform subcomponent sizes, imbalanced in the contribution of subcomponents, overlapping subcomponents and new transformations to the base functions by means of ill-conditioning, symmetry breaking, and irregularities. The base functions used to form the separable and non-separable subcomponents were Sphere, Rastrigin's, Ackely's, Schwefel's, and Rosenbrock's functions.

A particular property of these 15 functions is that they are additively decomposable, even though they are not fully-separable. This means that they can be written as the sum of component functions, each of which depends on a smaller number of variables [59]. And particularly, they are actually written as the sum of component functions [37, 58]. Note that in this context, it is necessary for non-separability that some variables participate in several component functions or in a non-separable component function. We exploit the decomposable property to parallelize the evaluation of new candidate solutions by means of the concurrent evaluation of multiple subcomponents or subsets of variables. Concretely, each solution evaluation is carried out by a MapReduce job where:

1. First, the solution is decomposed into groups of variables $(g_1, \ldots, g_N)$, which coincide with the arguments of the component functions. Note that variables used in more than one subfunction must be copied in all the corresponding groups (some cases of CEC 2013 functions).
2. Second, mappers, see (1), receive the solution identifier id_{sol} as key and a structure with a group of variables g_i and some parameters for the subfunction sf_i to be computed (subfunction weights, among others, for CEC 2013 functions) as value. They return id_{sol} as key and the result of $sf_i(g_i)$ as value.
3. Finally, reducers, see (2), are given id_{sol} as key and a set of partial fitness values from mappers and output the corresponding aggregation of these values. Note that, though there is only one reducer per evaluation of a complete solution, computing the aggregation is a light task, so it does not burdens the computation.

$$map(id_{sol}, \{g_i, params_{sf_i}\}) \rightarrow list(id_{sol}, sf_i(g_i)) \quad (1)$$

$$reduce(id_{sol}, list(sf_i(g_i))) \rightarrow list(id_{sol}, fitness) \quad (2)$$

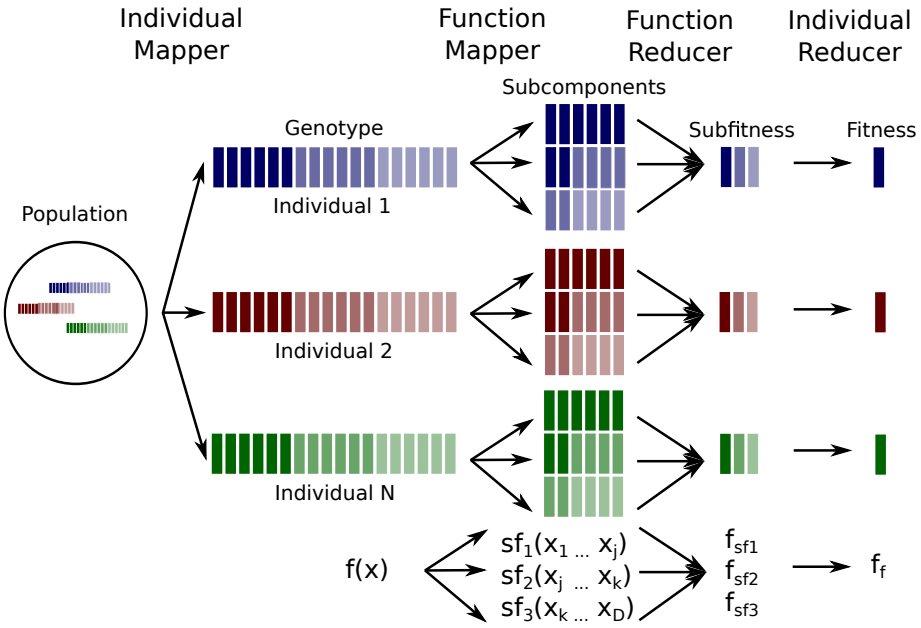

Figure 1: MapReduce model for parallel population and function evaluation.

This MapReduce model is shown in Figure 1 and it details the functions mapper, reducer, and variables distribution. The figure uses multiple colors to illustrate the cases in which variables are shared by several subcomponents. Moreover, it is shown how it is extended to run parallel evaluation of the individuals in the population, i.e., mappers can be run for multiple individuals concurrently.

Lastra et al. [32] implementation of the GPU-MA-SW-Chains employed a similar approach for the function evaluation using the CUDA programming model for NVIDIA GPUs. They decomposed the functions into subcomponents computed concurrently by multiple GPU threads and thread blocks. Their approach benefits from millions of threads that can be mapped to the variables computation. However, the actual number of parallel resident threads running in a given time is limited to 2048 per multiprocessor. For instance, the latest M40 GPU based on the GM200 Maxwell architecture comprises 24 multiprocessors that would lead to up to 49152 resident threads. The GPU is very efficient and competitive for LSGO. Nevertheless, the main problem of the implementation of LSGO on GPUs is due to the limitation in the amount of global memory available, currently 6-12 GB in top GPUs, which limits the function's dimensionality.

On the other hand, the advantage of using the MapReduce approach on regular CPUs in a cluster is that the amount of memory available is significantly much larger, then allowing to scale the function's dimensionality to a larger number of variables. Moreover, while MapReduce is a programming model for distributing the computation, the compute unit in which the subcomponent is to be evaluated may be either a local or remote CPU or GPU. Therefore, it allows for a scalable, flexible and heterogeneous compute architecture that enables to address extremely high-dimensional decomposable problems.

3.2. MapReduce-based MA-SW-Chains

The main idea when designing the MapReduce optimization algorithm is to distribute computation among all the compute units to take full advantage from all the computational and memory resources available with low network overhead. However, the optimization process of the original MA-SW-Chains algorithm is essentially sequential, both at the steady-state (SSGA) and the local search (LS) stages a given solution is iteratively optimized at a time. Therefore, the benefits of applying straightforward parallelization or MapReduce seem limited to parallel fitness function evaluation and genetic operators. For a better exploitation of the available computation resources, we propose a distributed adaptation of MA-SW-Chains that applies the population-evolution-concurrent idea (see Section 2.2), named MR-MA-SSW-Chains, which is detailed next.

3.2.1. Architecture

The architecture of the MR-MA-SSW-Chains algorithm consists of a master node and a set of intermediate compute units connected through a high-speed low-latency local network. The compute nodes hold a local population of solutions and run the steady-state GA and local search strategies locally. The master node synchronizes computation among the compute nodes and leads migrations of solutions among populations. This architecture minimizes data transfers while utilizing all the computational resources from the compute nodes.

The MR-MA-SSW-Chains method is described in Algorithm 2, where operations are run by the master node (MN) and a set of compute nodes (CN). Firstly, all the compute nodes generate an initial population and these solutions are evaluated using the MapReduce process for function evaluation. Then, the algorithm iterates until the number of evaluations has been

Algorithm 2 Distributed MR-MA-SSW-Chains.

```
1: CN: Generate initial population
2: while evaluations < maxEvaluations do
3:    CN: Run SSGA
4:    MN: Select best individual from all the SSGA
5:    MN: Transfer best individual to all CN for LS
6:    CN: Run the SSW local search
7:    CN: Transfer optimized LS solutions to the MN
8:    MN: Build combined solution
9:    MN: Replicate best solution in all CN
10: end while
```

reached. For every iteration, the master node controls the iteration process, which consists in the following steps. First, every intermediate node runs the SSGA, similar to the original MA-SW-Chains algorithm. Second, the master node selects the best individual from all populations and transfer this solution to the compute nodes. Third, compute nodes run the local search using the transferred solution and return the optimized solution to the master node. Fourth, the master node combines the solutions from the compute nodes. Finally, the master node replicates the best solution among the local search solutions and the combined solution to all the compute nodes so that it is introduced into each population replacing the worst solution.

In extremely high-dimensional problems it is very difficult to optimize a high number of variables simultaneously. This architecture allows to generate in parallel multiple solutions per iteration. The combination of the solutions from the different local searches is a key component of the optimization procedure. The original implementation of the SW/SSW local search optimizes iteratively a single individual (the best solution not already optimized from the SSGA population) for a given number of iterations. Here we faced two design options: 1) follow the original implementation and simply distribute the local search evaluation into multiple Mappers and Reducers along all the compute nodes; 2) taking into account the extremely high dimensionality of the functions, multiple instances of the local search may be run in parallel by the compute nodes addressing different sets of variables following the Subgrouping Solis Wets methodology, that eventually combined would lead to better solutions. Then, it is expected that the combination of the multiple local searches over different variables may produce a better joint solution (see

Section 3.2.3). This option also employs Mappers and Reducers to maximize performance by using the CPU cores, but within the context of each compute node, and it also avoids network overheads due to synchronization in every iteration.

3.2.2. Subgrouping Solis Wets

The Subgrouping Solis Wets method has been modified to maximize compute utilization. The original method iteratively improves a single solution, learning and updating a direction of optimization of each variable, $bias_d$. However, a single space point is explored at a time. Owing the advantages of multi-core processors and the parallel processing capabilities, the method is modified to explore multiple points at a time using a random displacement based on the individual's bias. Eventually, the bias vector is updated with the point that produced better fitness improvement.

The original Subgrouping Solis Wets algorithm [48] selected randomly a consecutive subset of dimensions in which the local search was run. It improved a subset of variables whose size is 20% of the number of dimensions. However, the modified method employed in our algorithm runs multiple local searches over different subsets of dimensions to combine them ultimately. Therefore, individuals contain an array which defines the probability of selecting each dimension, i.e,. the local search is not restricted to a consecutive subset of dimensions as in the original method. The idea is that promising dimensions which led to fitness improvements in previous local search runs are more likely to be selected for the next local search. Each dimension d has an individual probability P_d to be improved in the local search. Let the sum of these intermediate probabilities be P_D. Initially the probabilities for all dimensions are the same ($P_d = 0.5$). When the local search is initialized, it selects a random subset of variables using the expression (3) to select each dimension.

$$0.2 \times D \times \frac{P_d}{P_D} \tag{3}$$

Then, the number of random variables for each local search, selected according to their relative probabilities P_d, is about 20% of the number of dimensions, as in [48]. Consequently, the search size of the local search is also variable for each dimension (σ_d), in contrast to the uniform search size for all variables in the original SSW described in Section 2.3. This allows those variables with a high number of local search successes to increase σ_d to improve exploitation whereas other dimensions not explored by the local

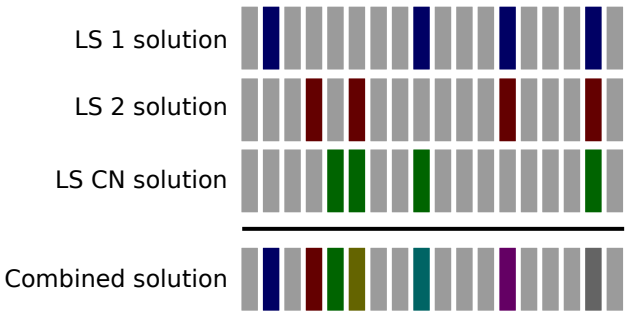

Figure 2: Combination of solutions from multiple local searches.

search may better have a smaller value to increase the probability of successful local optimizations. Although the probabilities P_d change along the evolution, the relation between P_d and P_D remains in such a way that 20% of the variables are optimized.

LaTorre et al. [34] uses similar approaches to automatically select the most appropriate heuristic for each function and search phase. They employ the multiple offspring sampling framework to combine different metaheuristics to find the best performing hybridization strategy. The hybridization allows a collaborative synergy and competitive selection among algorithms that improves the performance of the best one when it is used individually.

3.2.3. Combination of solutions

Multiple instances of the local search are run in parallel in the intermediate compute nodes, expecting each one to be focused on a given subset of variables and producing different solutions. At the end, these solutions are combined to build a joint solution. The outcome of combining solutions from multiple local searches allows to identify fitness improvements and the subset of dimensions optimized which lead to those improvements. Given the set of local search solutions optimized in the compute nodes, they are transferred to the master node where they are combined. The combined solution is constructed for each dimension d by means of the average value from the k LS solutions that improved the fitness value and optimized that dimension. It is calculated as $solution_d = \frac{1}{k}\sum_{i=1}^{k} LS_{i,d}$. This procedure is illustrated in Figure 2, which shows in color the dimensions optimized by each local search and how they are combined.

Similarly, the probabilities array (P_d values) is also updated based on the fitness improvements from the multiple local search solutions. The hyper-

cube framework idea [5] is adopted for this learning coevolutive step [42]. LS processes are ranked according to the improvement produced (r_i), with the highest rank for the best LS process. Then, auxiliary probabilities for the optimised dimensions are computed as the fraction of their corresponding ranking divided by the number of successful LS processes ($r_i/|LS|$). The final probability of each dimension is adapted from the previous probability towards the auxiliary value according to formula (4), where α is a *learning rate* parameter.

$$P_d = P_d + \alpha \cdot \left(\frac{r_i}{|LS|} - P_d \right) \tag{4}$$

Notice that, given that the group of variables is selected according to (3), increasing P_d for some dimensions reduces the probability of selecting the others. Therefore, the amount of variables to be selected at any local search iteration remains constant at 20%.

Finally, notice that in order to enable the LS chaining idea in our adapted Subgrouping Solis Wets' method, an individual of the MR-MA-SSW-Chains is comprised of the following components: the solution array, the $bias_d$ array, the search size σ_d array, the probabilities P_d array, and the fitness value.

3.2.4. Fitness precomputation in local search

The efficiency in the evaluation of the local search solutions using subgroups and additively decomposable functions may be improved significantly. The subgrouping local search optimizes a relatively small number of variables as compared with the total number of dimensions. Specifically, only 20% of the variables change during the local search according to (3). This means that 80% of the variables are not being addressed by the local search and whose values are not changed along subsequent invocations of the fitness function. Therefore, the contribution of these fixed variables to the fitness function may be computed only once and its value to be reused in subsequent evaluations. This way, only the updated values on the subset of dimensions being optimized by the local search are submitted to the fitness function, then saving considerable time, especially when the dimensionality increases.

4. Experiments

The experimental study comprises two experiments to analyze and compare the performance of the proposal on high-dimensional and extremely high-dimensional problems using the benchmark functions from the IEEE CEC 2010 and 2013 LSGO competitions. The contribution of both the parallel local search and the parallel generation of solutions per compute node are analyzed to see how they impact fitness improvement. Finally, a runtime analysis is carried out to compare the efficiency improvements of the MapReduce and fitness precomputation approaches.

4.1. Experimental setup

Experiments were run on a CPU cluster with a master node and 8 intermediate compute nodes connected through gigabit interfaces. Nodes comprised two Intel Xeon E5645 processors at 2.4 GHz each with 6 cores and 24 GB of DDR3 RAM memory, where the MapReduce-based evaluation of solutions takes place. Then, the total amount of hardware used in the experiments is 108 CPU cores and 216 GB of RAM. The operating system was Linux Rocks Cluster 6.1-x86-64. The parameter settings for the algorithms employ the values recommended by Molina et al. in [47] and [48], including the population size (100) and LS intensity.

4.2. MapReduce implementation

The initial implementation of the MR-MA-SSW-Chains was based on Apache Hadoop [56]. However, Hadoop is not an appropriate implementation for LSGO since this is a compute-intensive and not a data-intensive application. Hadoop introduces a significant amount of overhead because the Hadoop architecture design of Mappers and Reducers employs the Hadoop Distributed File System (HDFS) to read/write input/output data in the disk drives. Moreover, every time an evaluation is requested the mappers/reducers are instanced as processes, which takes time an introduces latency and overheads in the range of seconds [55]. In the context of LSGO, as shown in Section 4.6, the evaluation time of a solution is less than a second or a few seconds. Therefore, the latency introduced to read/write the solutions in the HDFS and instance the mappers/reducers would significantly impact the performance, as compared to the actual time required to evaluate a solution.

On the contrary, our approach employs persistent mappers/reducers (avoiding process creation overhead in every evaluation), as well as it runs memory-to-memory transfers of solutions, avoiding virtual memory (disk drive). Nevertheless, it is true that in other data intensive problems where evaluation may take minutes (e.g. big data classification), an overhead in the range of seconds due to the HDFS does not have a significant impact in the overall evaluation time. Specifically, we employed an implementation based on remote invocations in Java using RMI that simplifies coding, reduces delays and maximized gigabit throughput. The implementation is not coupled with the algorithm and it employs the open-source publicly available JCLEC software [8, 61] (Java Class Library for Evolutionary Computation).

4.3. IEEE CEC 2013 LSGO functions

The first experiment evaluates and compares the performance of the MA-SW-Chains, MA-SSW-Chains, and MR-MA-SSW-Chains algorithms on the set of 15 benchmark functions from the IEEE CEC 2013 LSGO competition using 1,000 dimensions. It is important to notice that our model principally differ from the two others in the parallel exploitation of available computational units. Therefore, the aim of this experiment is to check if the adapted MR-MA-SSW-Chains is able to provide results similar to those of its predecessors, and eventually better results thanks to the usage of more computational resources. However, in order to show a fair comparison we provide both the results for an equal number of function evaluations (FEs) and FEs per compute node. This way, the reader can observe the impact in the quality of the results due to having a larger bucket of FEs. On the other hand, the MR-MA-SSW-Chains is biased and unfavoured in a comparison with the other methods for an absolute exactly number of FEs. The reason is that the algorithm consumes 8 times (8 compute nodes) the number of evaluations per iteration and consequently, it does not have time to evolve as much as sequential versions.

Experiments were run 25 times and the best, median, worst, mean and standard deviation results are shown in Tables 1 and 2. Results are provided at the level of 3 million evaluations, and additionally for 3 million evaluations per compute unit to evaluate the benefits of generating multiple solutions in parallel. The algorithms with the best results are highlighted in gray for each of the functions.

Table 1: Comparison on CEC 2013 functions for 1,000 D and 3M FEs (F1-F10).

Algorithm	Result	F1	F2	F3	F4	F5
MA-SW-Chains	Best	3.83E-14	9.84E+02	2.13E+01	1.26E+09	1.09E+06
	Median	4.49E-13	1.22E+03	2.14E+01	4.97E+09	1.91E+06
	Worst	2.09E-12	1.49E+03	2.14E+01	1.08E+10	2.64E+06
	Mean	6.27E-13	1.24E+03	2.14E+01	4.96E+09	1.86E+06
	Std	5.34E-13	1.42E+02	4.52E-02	2.69E+09	3.66E+05
MA-SSW-Chains	Best	2.46E-13	9.42E+02	2.02E+01	6.35E+09	7.19E+05
	Median	5.42E-13	1.03E+03	2.03E+01	1.21E+10	1.31E+06
	Worst	3.28E-12	1.62E+03	2.04E+01	2.03E+10	3.04E+06
	Mean	8.18E-13	1.06E+03	2.03E+01	1.28E+10	1.34E+06
	Std	6.88E-13	1.22E+02	3.90E-02	3.79E+09	3.86E+05
MR-MA-SSW-Chains 3M FEs	Best	0.00E+00	2.11E+03	2.00E+01	5.92E+09	3.67E+06
	Median	0.00E+00	2.41E+03	2.00E+01	1.73E+10	5.01E+06
	Worst	4.21E-15	2.63E+03	2.00E+01	2.58E+10	6.40E+06
	Mean	1.26E-15	2.39E+03	2.00E+01	1.63E+10	5.00E+06
	Std	1.50E-15	1.31E+02	1.21E-05	6.62E+09	7.56E+05
MR-MA-SSW-Chains 3M FEs per node	Best	0.00E+00	2.80E+02	2.00E+01	2.04E+09	3.66E+06
	Median	0.00E+00	3.35E+02	2.00E+01	4.96E+09	5.01E+06
	Worst	0.00E+00	3.53E+02	2.00E+01	1.14E+10	6.39E+06
	Mean	0.00E+00	3.30E+02	2.00E+01	4.84E+09	5.00E+06
	Std	0.00E+00	1.95E+01	2.31E-13	3.02E+09	7.56E+05
Algorithm	Result	F6	F7	F8	F9	F10
MA-SW-Chains	Best	9.96E+05	2.91E+05	3.30E+13	1.80E+08	9.06E+07
	Median	1.01E+06	3.98E+06	4.87E+13	5.11E+08	9.10E+07
	Worst	1.04E+06	4.33E+06	6.36E+13	1.06E+09	9.31E+07
	Mean	1.01E+06	3.69E+06	4.82E+13	5.38E+08	9.13E+07
	Std	1.38E+04	1.01E+06	9.92E+12	2.34E+08	8.18E+05
MA-SSW-Chains	Best	1.04E+06	4.11E+07	8.12E+13	9.51E+07	9.28E+07
	Median	1.05E+06	7.90E+07	1.42E+14	2.09E+08	9.35E+07
	Worst	1.06E+06	1.73E+08	2.17E+14	6.34E+08	9.39E+07
	Mean	1.05E+06	8.41E+07	1.44E+14	2.56E+08	9.35E+07
	Std	3.64E+03	2.68E+07	3.18E+13	1.45E+08	2.38E+05
MR-MA-SSW-Chains 3M FEs	Best	1.01E+06	1.42E+07	1.64E+14	2.79E+08	9.13E+07
	Median	1.02E+06	7.00E+07	3.34E+14	4.13E+08	9.23E+07
	Worst	1.03E+06	1.95E+08	5.22E+14	5.02E+08	9.26E+07
	Mean	1.02E+06	8.77E+07	3.30E+14	4.01E+08	9.21E+07
	Std	7.64E+03	5.74E+07	1.07E+14	5.93E+07	4.13E+05
MR-MA-SSW-Chains 3M FEs per node	Best	9.96E+05	8.80E+03	2.32E+13	2.79E+08	9.05E+07
	Median	1.00E+06	7.46E+04	8.27E+13	4.13E+08	9.15E+07
	Worst	1.01E+06	1.72E+06	1.27E+14	5.02E+08	9.24E+07
	Mean	1.00E+06	3.18E+05	8.38E+13	3.99E+08	9.14E+07
	Std	5.00E+03	4.24E+05	2.56E+13	5.98E+07	5.40E+05

Table 2: Comparison on CEC 2013 functions for 1,000 D and 3M FEs (F11-F15).

Algorithm	Result	F11	F12	F13	F14	F15
MA-SW-Chains	Best	9.15E+11	1.08E+03	1.59E+07	1.16E+08	4.32E+06
	Median	9.21E+11	1.25E+03	1.88E+07	1.45E+08	5.12E+06
	Worst	9.42E+11	1.43E+03	2.30E+07	1.85E+08	6.86E+06
	Mean	9.24E+11	1.24E+03	1.89E+07	1.46E+08	5.17E+06
	Std	7.57E+09	9.69E+01	2.13E+06	1.75E+07	6.40E+05
MA-SSW-Chains	Best	9.16E+11	1.11E+03	2.75E+09	8.72E+09	5.99E+06
	Median	9.26E+11	1.33E+03	4.43E+09	3.78E+10	8.20E+06
	Worst	9.46E+11	1.53E+03	9.28E+09	6.58E+10	1.19E+07
	Mean	9.29E+11	1.34E+03	4.92E+09	3.78E+10	8.38E+06
	Std	9.54E+09	1.05E+02	1.63E+09	1.61E+10	1.52E+06
MR-MA-SSW-Chains 3M FEs	Best	9.83E+08	6.77E+02	9.57E+08	3.13E+09	5.73E+06
	Median	1.75E+09	1.16E+03	4.34E+09	2.75E+10	9.03E+06
	Worst	8.76E+09	1.51E+03	6.09E+09	5.52E+10	1.21E+07
	Mean	3.17E+09	1.10E+03	4.35E+09	2.96E+10	8.90E+06
	Std	2.41E+09	2.36E+02	1.17E+09	1.44E+10	1.93E+06
MR-MA-SSW-Chains 3M FEs per node	Best	4.07E+07	8.32E-05	4.43E+06	1.71E+07	4.27E+05
	Median	1.63E+08	5.57E+00	2.02E+08	1.00E+08	5.30E+05
	Worst	4.59E+08	5.24E+02	5.03E+08	1.41E+08	6.55E+05
	Mean	1.88E+08	7.06E+01	2.44E+08	8.85E+07	5.39E+05
	Std	1.46E+08	1.02E+02	1.41E+08	3.55E+07	6.12E+04

MR-MA-SSW-Chains obtains the best median results in 10 of the 15 functions and the best results in 12 of the 15 functions. Specifically, MR-MA-SSW-Chains improves significantly in F7, F11, F12, and F15 functions. On the contrary, there are some functions for which MR-MA-SSW-Chains could not get averaged or median results better than those of its competitors (F5, F8, F9, F10, F13), even though it was allowed to evaluate more solutions. Leaving apart F10, Ackley's function, where performance differences are small due to the fact that most of the search space is associated with similar fitness values in this function; there are some functions where MA-SW-Chains win (8 and 13, both unimodal with a high condition number), and some where the winner is MA-SSW-Chains (5 and 9, both based on Rastrigin's function), according to averaged and median results:

- For the former cases, functions F8 and F13, our hypothesis is that the non-subgrouping SW Local Search is performing better than the subgrouping one because of the unimodality and conditioning of the function. Interestingly, the results of MA-SSW-Chains in these two functions is similar or worse to those of MR-MA-SSW-Chains. This

means that the complete Solis Wets is being able to find the interesting variables, because of the high condition number of the problem, faster than the subgrouping versions. This is because it evaluates new solutions changing the values of all the variables of the solution, whereas the subgrouping versions waste evaluations trying changes on not that interesting variables.

- For the latter, functions F5 and F9, we think that our MR-MA-SSW-Chains is getting stuck in worse local optima than MA-SSW-Chains, because the information combination is mostly restricted to the island's populations. Given that the global population size of our method is the same as that of the other method, but distributed along 8 compute nodes, each one has a small population that might be suffering from premature convergence much more than MA-SSW-Chains.

The use of statistical tests to assess the performance of algorithms has become widespread in computational intelligence [14]. The Friedman's test is a non-parametric statistical test in which the null-hypothesis states that the results from all the algorithms are equivalent. The rejection of the null-hypothesis implies the existence of differences among the performance of the algorithms. The test ranks the algorithms for each benchmark function separately, the best performing algorithm getting the rank of 1, the second best rank 2, and so on, and in the case of ties it averages ranks. The Friedman statistic χ^2 with k-1 degrees of freedom (where k is the number of algorithms) and 15 benchmark functions from the IEEE CEC 2013 competition is 45.9, whereas the p-value computed for the Friedman test is 0. Moreover, the Iman and Davenport test which runs a similar procedure for multiple comparisons, employing the F-distribution, establishes a p-value of 7E-12. Therefore, the null-hypothesis is rejected and thus a post-hoc statistical test can be applied.

The Holm's test is a post-hoc procedure that detect those algorithms which are worse than the algorithm with the best results (control algorithm). The Holm's procedure rejects those algorithms that have a p-value < 0.05. According to the Holm's test results shown in Table 3, MR-MA-SSW-Chains achieves significant differences with the rest of the algorithms.

On the other hand, the Wilcoxon test is used to perform pairwise comparisons between two algorithms. It measures the performance differences for each benchmark function and it ranks them according to their values. Let R^+ be the sum of ranks for the functions in which the control algorithm outperformed the compared, and R^- the sum of ranks for the opposite. According

Table 3: Holm's test for 1,000 D and $\alpha = 0.05$, MR-MA-SSW-Chains (3M FEs per node) is the control algorithm.

Algorithm	z	p-value
MA-SW-Chains	2.6536	0.0079
MA-SSW-Chains	6.1237	0.0000
MR-MA-SSW-Chains (3M FEs)	5.2664	0.0000

Table 4: Pairwise comparison using Wilcoxon test for 1,000 D.

Comparison	R^+	R^-	p-value
MR-MA-SSW-Chains vs MA-SW-Chains	647	343	0.0769
MR-MA-SSW-Chains vs MA-SSW-Chains	916	119	1.26E-6
MR-MA-SSW-Chains (3M FEs) vs MA-SW-Chains	247	787	≥ 0.2
MR-MA-SSW-Chains (3M FEs) vs MA-SSW-Chains	609	426	≥ 0.2
MR-MA-SSW-Chains vs MR-MA-SSW-Chains (3M FEs)	972	18	2.87E-11

to the Wilcoxon test results shown in Table 4, MR-MA-SSW-Chains achieves significant differences with MA-SSW-Chains (p-value 0.0769) and MA-SSW-Chains (p-value 1.26E-6). Moreover, it also compares the MR-MA-SSW-Chains algorithm for an equal number of FEs, resulting better performance than MA-SSW-Chains but worse than MA-SW-Chains. Furthermore, the last row compares the benefits of a larger FEs bucket generating in parallel multiple solutions per iteration. The results of these statistical analyses confirm that our MR-MA-SSW-Chains is exploiting correctly the additional computational hardware to globally produce better results than its sequential competitors, even though there were some individual functions with characteristics that promoted different results.

Figure 3 illustrates the convergence curves of the three algorithms on the 15 functions from the IEEE CEC 2013. For each function, the single convergence curve has been plotted using the median results over all 25 runs. These log–log plots show how the algorithms improve their fitness as the number of evaluations increase. Curves are provided for MR-MA-SSW-Chains at an equal number of function evaluations as well as function evaluations per compute node. We observe that, in many cases, fitness values are highly improved in early iterations, while it flattens in the latest. Additionally, we can see confirmed our previous hypothesis: MR-MA-SSW-Chains suffers symptoms of premature convergence on several multimodal functions, specially F5 and F9, and hardly scape from the corresponding local optima.

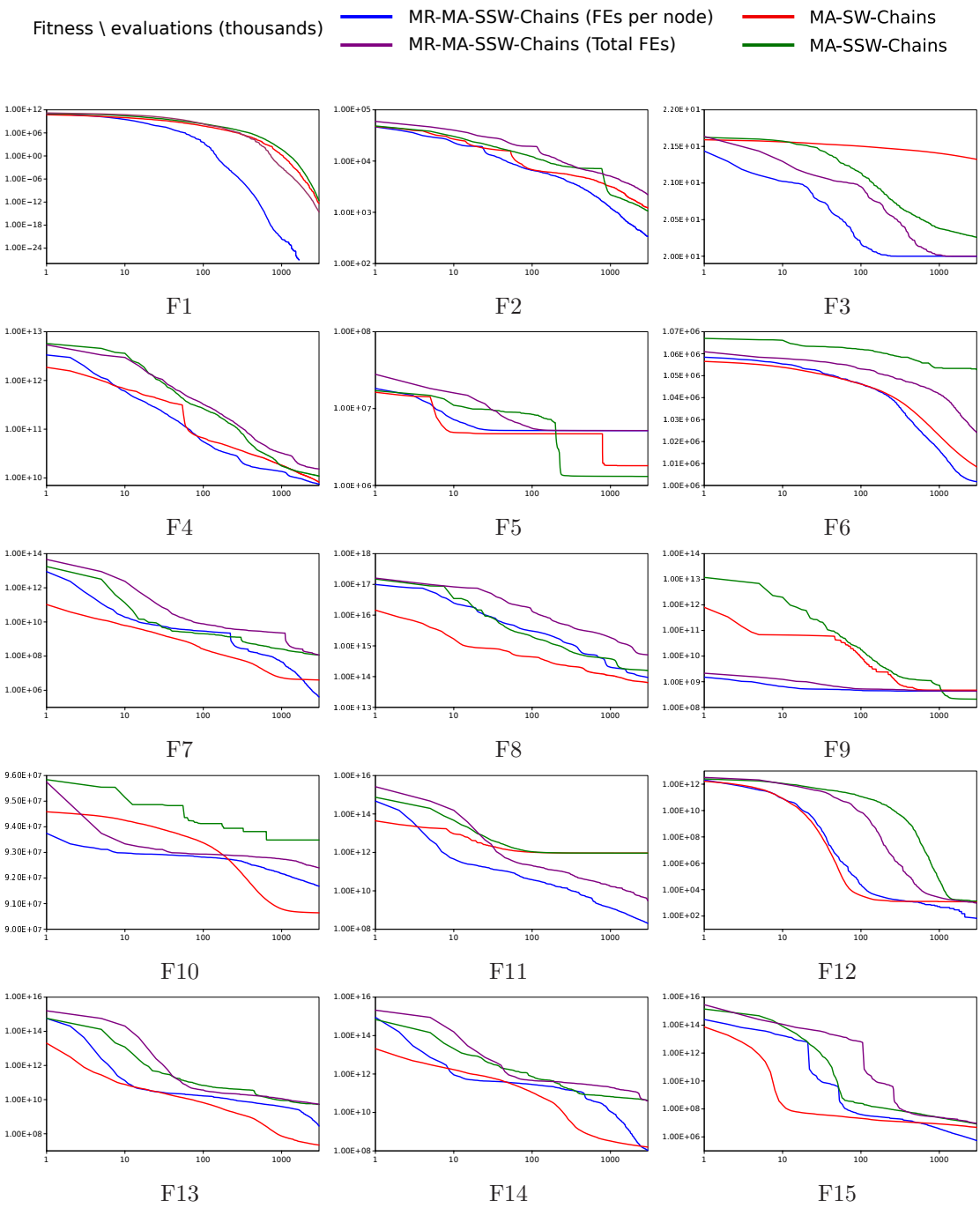

Figure 3: Covergence results for CEC 2013 functions on 1,000 D and 3M evaluations.

The method also provides a slower convergence to the optimum on F8 and F13. Finally, another interesting detail in the graphs is that the lines corresponding to the MR-MA-SSW-Chains method with fitness per evaluation per compute node is always better than that of the same method with fitness per absolute evaluation, but the difference is not excessive. Particularly, the lines show a certain parallelism due to the fact that the extended model just benefits of 8 times more evaluations, which just produces a shift of the line to the left side.

4.4. Extremely high-dimensional functions

Increasing the dimensionality to millions of variables is a challenging issue in terms of computational performance and heuristic-driven search on huge spaces. In fact, researchers usually discuss and concern about the effectiveness of heuristics on such extremely high-dimensional problems. To shed some light in this regard, we include random search in this experiment to measure the benefits of our method when the search space is that huge [15, 21].

Lastra et al. [32] implemented MA-SW-Chains using GPUs (GPU-MA-SW-Chains) and evaluated its performance on a subset of the IEEE CEC 2010 LSGO functions. Specifically, 9 benchmark functions were scaled up to 1M and 3M dimensions. The second experiment evaluates and compares the performance of the GPU-MA-SW-Chains, MR-MA-SSW-Chains, and random search algorithms on 1M, 3M, and 10M dimensions using the IEEE CEC 2010 functions included in [32].

Due to runtime restrictions Lastra et al. [32] limited the number of maximum function evaluations to 500K on extremely high-dimensional problems. Therefore, in order to perform results comparisons on 1M and 3M variables, we also show results for that number of evaluations. Specifically, for the 10M variables comparison, the number of evaluations is reduced to 100K. All methods are compared for an equal absolute number of function evaluations.

Tables 5, 6, and 7 show the results for 1M, 3M and 10M dimensions respectively. Random search is shown to perform worst on all the functions, especially on 10M dimensions. MR-MA-SSW-Chains improves the results of GPU-MA-SW-Chains in 8 out of the 9 benchmark functions. The only function where GPU-MA-SW-Chains excels is F8, based on Rosenbrok's function. There are two principal operational differences between the memetic algorithms, both involving the usage of the Solis and Wets local search method, that explain these results: 1) MR-MA-SSW-Chains executes several local

optimisation processes on one solution, taking advantage of the distributed hardware; whereas GPU-GA-SW-Chains carries out just one optimisation, whose operations are distributed over the GPU cores; 2) the optimisation processes consider subgroups of variables in MR-MA-SSW-Chains, whereas all the variables are treated in GPU-MA-SW-Chains. These differences may explain the general superior results of MR-MA-SSW-Chains, since optimisation if often more fruitfull when dealing with a reduced set of variables, and even more if multiple processes are deployed. However, Rosenbrock function, key component in F8, is characterised by having a parabolic shaped flat valley easy to reach but very difficult to make progress therein (because of its flatness). This fact is specially problematic for the subgroup Solis Wets' method in our proposal, because many evaluations are wasted looking for the subset of variables that produces an improvement in fitness, i.e., the flatness makes difficult to find the right improving direction. On the contrary, the non-subgroup Solis Wets' method in GPU-MA-SW-Chains is not this much affected because all the variables are taking into account at each step, so the probability of producing steps in the right direction is to some extent higher. Interestingly, these hypothesis are supported by the results provided by Molina et al. [48]. There, the Rosenbrock function is F6 and one can observe that MA-SSW-Chains can not obtain as good results as the non-subgrouping version for the case with more dimensions (D=1000).

The Friedman's test rejects the null-hypothesis on the three dimensionalities with p-values lower than 1E-5. Table 8 shows the results of the Holm's test for 1M and 3M dimensions, which demonstrates the global better performance of MR-MA-SSW-Chains achieving significant p-values. The test cannot be applied on 10M dimensions since there are only two algorithms in the comparison. Therefore, we apply the Wilcoxon test to analyze pairwise comparisons, whose results are shown in Table 9. The Wilcoxon test indicates the superior performance of the two heuristic methods as compared with the random search. Moreover, it is shown that MR-MA-SSW-Chains overcomes the GPU-MA-SW-Chains with low p-values.

Table 5: Results for 1,000,000 D and 500,000 evaluations (CEC 2010).

GPU-MA-SW-Chains	F1	F2	F3	F4	F6	F8	F13	F18	F20
Median	2.60E+14	2.00E+07	2.20E+01	2.80E+14	2.20E+01	4.30E+07	2.00E+15	4.20E+15	4.20E+15
Mean	2.60E+14	2.00E+07	2.20E+01	2.80E+14	2.20E+01	1.10E+08	2.00E+15	4.20E+15	4.20E+15
Std	1.20E+12	3.60E+04	2.80E-03	9.90E+11	6.40E-01	7.80E+07	4.10E+12	7.50E+12	1.20E+13
MR-MA-SSW-Chains	F1	F2	F3	F4	F6	F8	F13	F18	F20
Best	1.39E+14	1.57E+07	2.08E+01	1.45E+14	2.09E+01	2.51E+09	7.34E+14	1.65E+15	1.72E+15
Median	1.40E+14	1.57E+07	2.09E+01	1.53E+14	2.09E+01	2.65E+09	7.73E+14	1.72E+15	1.78E+15
Worst	1.41E+14	1.58E+07	2.09E+01	1.55E+14	2.09E+01	2.67E+09	7.99E+14	1.76E+15	1.79E+15
Mean	1.40E+14	1.57E+07	2.09E+01	1.51E+14	2.09E+01	2.60E+09	7.59E+14	1.70E+15	1.77E+15
Std	8.40E+11	3.24E+04	1.04E-02	3.61E+12	5.11E-03	6.21E+07	2.56E+13	3.94E+13	2.32E+13
Random Search	F1	F2	F3	F4	F6	F8	F13	F18	F20
Best	4.45E+14	2.55E+07	2.16E+01	1.58E+15	2.06E+07	4.58E+16	4.54E+15	9.11E+15	9.29E+15
Median	4.46E+14	2.55E+07	2.16E+01	1.79E+15	2.07E+07	5.72E+16	4.54E+15	9.11E+15	9.30E+15
Worst	4.46E+14	2.55E+07	2.16E+01	1.95E+15	2.08E+07	6.95E+16	4.55E+15	9.12E+15	9.30E+15
Mean	4.46E+14	2.55E+07	2.16E+01	1.78E+15	2.07E+07	5.63E+16	4.54E+15	9.11E+15	9.30E+15
Std	3.29E+11	7.10E+03	2.97E-04	1.42E+14	5.19E+04	7.55E+15	3.81E+12	4.24E+12	1.07E+12

Table 6: Results for 3,000,000 D and 500,000 evaluations (CEC 2010).

GPU-MA-SW-Chains	F1	F2	F3	F4	F6	F8	F13	F18	F20
Median	8.60E+14	6.10E+07	2.20E+01	8.90E+14	2.20E+01	4.30E+07	6.40E+15	1.37E+16	1.30E+16
Mean	8.60E+14	6.10E+07	2.20E+01	8.90E+14	2.20E+01	4.40E+07	6.40E+15	1.30E+16	1.30E+16
Std	1.80E+12	4.40E+04	2.80E-03	3.60E+12	5.60E-01	1.20E+06	1.10E+13	6.50E+12	1.60E+13
MR-MA-SSW-Chains	F1	F2	F3	F4	F6	F8	F13	F18	F20
Best	6.53E+14	5.53E+07	2.12E+01	6.58E+14	2.12E+01	1.04E+10	3.82E+15	8.87E+15	8.96E+15
Median	6.54E+14	5.55E+07	2.12E+01	6.60E+14	2.12E+01	1.05E+10	3.84E+15	8.96E+15	9.05E+15
Worst	6.56E+14	5.56E+07	2.12E+01	6.65E+14	2.12E+01	1.06E+10	3.87E+15	9.03E+15	9.08E+15
Mean	6.54E+14	5.55E+07	2.12E+01	6.61E+14	2.12E+01	1.05E+10	3.85E+15	8.95E+15	9.03E+15
Std	1.52E+12	1.52E+05	1.53E-02	3.19E+12	1.03E-03	1.13E+08	2.64E+13	8.02E+13	6.31E+13
Random Search	F1	F2	F3	F4	F6	F8	F13	F18	F20
Best	1.34E+15	7.67E+07	2.16E+01	2.73E+15	2.05E+07	5.24E+16	1.37E+16	2.74E+16	2.80E+16
Median	1.35E+15	7.67E+07	2.16E+01	2.98E+15	2.07E+07	7.51E+16	1.37E+16	2.74E+16	2.80E+16
Worst	1.35E+15	7.67E+07	2.16E+01	3.33E+15	2.08E+07	8.12E+16	1.37E+16	2.75E+16	2.80E+16
Mean	1.35E+15	7.67E+07	2.16E+01	3.01E+15	2.07E+07	7.07E+16	1.37E+16	2.74E+16	2.80E+16
Std	6.89E+11	7.69E+03	9.70E-05	2.29E+14	1.13E+05	1.13E+16	8.09E+12	8.80E+12	6.98E+12

Table 7: Results for 10,000,000 D and 100,000 evaluations (CEC 2010).

MR-MA-SSW-Chains	F1	F2	F3	F4	F6	F8	F13	F18	F20
Best	3.02E+15	2.01E+08	2.14E+01	3.08E+15	2.68E+05	4.44E+10	2.15E+16	4.41E+16	4.47E+16
Median	3.04E+15	2.08E+08	2.14E+01	3.16E+15	2.68E+05	4.56E+10	2.21E+16	4.50E+16	4.60E+16
Worst	3.18E+15	2.12E+08	2.14E+01	3.32E+15	2.68E+05	4.61E+10	2.29E+16	4.53E+16	4.72E+16
Mean	3.08E+15	2.06E+08	2.14E+01	3.18E+15	2.68E+05	4.58E+10	2.19E+16	4.49E+16	4.58E+16
Std	7.12E+13	4.52E+06	8.16E-03	1.00E+14	2.10E+00	7.56E+08	9.59E+15	5.05E+14	1.02E+15
Random Search	F1	F2	F3	F4	F6	F8	F13	F18	F20
Best	4.50E+15	2.56E+08	2.16E+01	6.28E+15	2.09E+07	8.41E+16	4.58E+16	9.17E+16	9.35E+16
Median	4.50E+15	2.56E+08	2.16E+01	6.86E+15	2.10E+07	9.94E+16	4.58E+16	9.17E+16	9.35E+16
Worst	4.50E+15	2.56E+08	2.16E+01	7.51E+15	2.10E+07	1.18E+17	4.58E+16	9.17E+16	9.36E+16
Mean	4.50E+15	2.56E+08	2.16E+01	6.82E+15	2.10E+07	1.01E+17	4.58E+16	9.17E+16	9.36E+16
Std	1.62E+12	1.42E+04	1.09E-04	4.31E+14	3.86E+04	1.33E+16	7.88E+12	7.54E+12	1.56E+13

Table 8: Holm's test for 1M and 3M dimensions for $\alpha = 0.05$, MR-MA-SSW-Chains is the control algorithm.

Algorithm	z	p-value
Random Search	4.8333	1.00E-6
GPU-MA-SW-Chains	1.6667	0.0955

Table 9: Wilcoxon test for extremely high-dimensional problems.

Comparison on 1M dimensions	R^+	R^-	p-value
MR-MA-SSW-Chains vs GPU-MA-SW-Chains	156	15	0.0010
MR-MA-SSW-Chains vs Random Search	171	0	7.63E-6
GPU-MA-SW-Chains vs Random Search	168	3	3.81E-5
Comparison on 3M dimensions	R^+	R^-	p-value
MR-MA-SSW-Chains vs GPU-MA-SW-Chains	156	15	0.0010
MR-MA-SSW-Chains vs Random Search	171	0	7.63E-6
GPU-MA-SW-Chains vs Random Search	168	3	3.81E-5
Comparison on 10M dimensions	R^+	R^-	p-value
MR-MA-SSW-Chains vs Random Search	378	0	1.49E-8

Moreover, it is important to analyze the convergence of the functions in an extremely high-dimensional scenario with 10M dimensions. Since the runtime of analyzing the convergence of all functions would be unfeasible, we analyzed the convergence of the F1, F4, F6, and F8 CEC 2013 functions, whose convergence speed and optimal value are easy to visualize. Figure 4 shows the convergence curves for the fitness functions along the number of function evaluations (total).

The curves indicate that the fitness was continuously improved along generations and the fitness landscape did not flat for functions F1, F4, and F8. Besides, it can bee seen that it usually takes a very large number of evaluations to reduce the error in an order of magnitude. This is because the search space is huge and progress, which is significant in absolute terms, takes an extraordinary number of steps when the functions dimensionality is this large. Apart from that, we observe that: F1, which is the elliptic unimodal function, is simple enough for the algorithm to show the highest convergence acceleration; F4, which differs from F1 in its non-separability property, is addressed with a little more difficulty; the same occurs on the also non-separable Rosenbrock function, F8 where, contrary to what was commented earlier for inferior dimensionalities, the algorithm seems to have

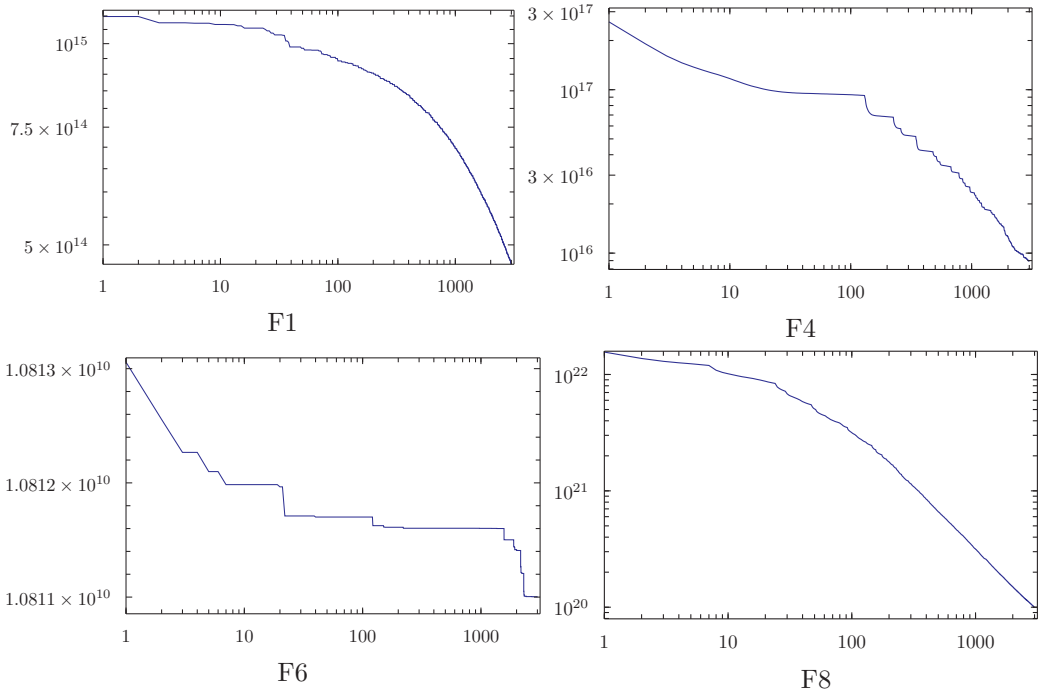

Figure 4: MR-MA-SSW-Chains convergence for some CEC 2013 functions on 10M D.

not reached the valley yet (the progress is significant, but the acceleration is inferior to that for F1); and finally, progress in F6, Ackley's function, is extremely difficult because it is highly multimodal with similar fitness values for most of the search space.

4.5. Parallel local search analysis

In Section 3.2.3 we introduced the multiple local searches run in parallel in the compute nodes and their combination (Figure 2) to produce a better joint solution. Therefore, it is necessary to analyze as well the performance improvement due to the concurrent optimization of different subgroups of variables and their combination in a new solution. Figure 5 shows the convergence of the F1, F4, F6, and F8 functions from CEC 2013 on 1,000 dimensions (blue line), and the fitness improvement obtained after the combination of the solutions from the multiple local searches (red dots). This fitness improvement is calculated as the difference between the fitness from the best solution among the parallel local searches and the fitness from the solution resulted from the combination. This way, it is easy to visualize that

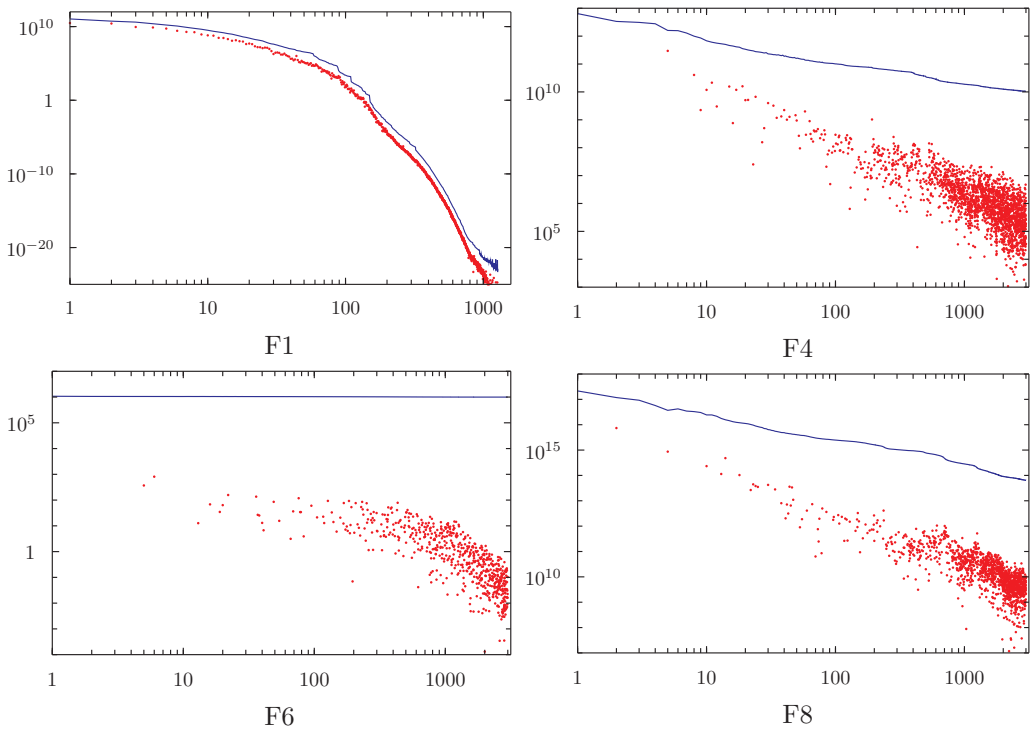

Figure 5: Fitness improvement due to combined solution from parallel LS on 1,000 D.

the combination of solutions from the parallel local searches results in fitness improvements. The red dots are not linked with lines because the amount of fitness improvement due to the solutions combination in a given iteration is not strictly related with the expected improvement in the following iteration, yet there is trendline in which in early iterations the exploitation of the search space leads to better performance improvements. Interestingly, the fitness improvement in F1 follows very closely to the actual value of the fitness because of the simplicity and separability of the function.

4.6. Runtime analysis

This section discusses and evaluates the optimizations addressed to increase the efficiency of the fitness computation. The MapReduce framework allowed for the parallel computation of solutions using concurrent individual evaluations and subfitness computation. The fitness precomputation in the local search optimization reduced the recurrent evaluation of unchanged subcomponents. Table 10 shows the evaluation time of a solution (in mil-

Table 10: Solution evaluation time (ms) on F1 function.

Dimensions	1,000	1M	3M	10M
Sequential	0.0927	134.37	410.07	1363.51
MapReduce	0.0731	38.75	61.71	182.65
GPU	0.0553	0.60	1.65	–
LS fitness precomputation	0.0492	2.89	3.61	4.69
LS MapReduce + fitness precomputation	0.0417	2.57	3.02	3.77

liseconds) for each of the approaches on different dimensionalities. The first row shows the sequential evaluation approach that follows the classical loop iteration model to compute the fitness. The second row shows the MapReduce approach that maps and computes in parallel multiple subfitness that are reduced to produce the fitness value. The speedup becomes relevant as the number of dimensions increase, since the overhead makes the speedup smaller when the dimensionality is low. The third row shows the evaluation time for the GPU implementation by Lastra et al. [32] using a NVIDIA Tesla C2050. The GPU demonstrates to achieve high performance, specially when the dimensionality increases and it benefits from the large number of compute cores. However, the function dimensionality of the GPU implementation is limited to 3 million variables due to the GPU's memory size and no results can be obtained for 10M (out of memory). Similarly to the MapReduce scenario, the impact of the overhead on large dimensionalities is smaller as compared with the actual computation time devoted to the function. In both cases, we agree that on small dimensionality, the advantages of parallelization are not significant.

On the other hand, it is also interesting to analyze the performance impact of the fitness precomputation carried out in the local search. The fourth row shows the evaluation time using fitness precomputation. For the sake of righteousness, the local search requires to compute the whole solution only once at the beginning of the process, while for the rest of evaluations it takes the precomputed values and only runs the evaluation on the subcomponents. Finally, the last row shows the MapReduce and fitness precomputation approaches combined. Again, since the MapReduce is only applied on the subcomponent variables, the speedup that can be achieved is smaller due to overheads.

5. Conclusion

In this paper we presented a MapReduce approach for scaling large scale global optimization functions to extremely high-dimensional problems with millions of variables. We proposed a distributed algorithm based on the MA-SW-Chains method and the MapReduce framework named MR-MA-SSW-Chains. The local search method based on Subgrouping Solis Wets was improved to optimize multiple subgroups concurrently on different subsets of variables. The solutions from the multiple parallel local searches were combined to build a joint solution that improved individual solutions.

The experimental study compared the performance of the proposal with the MA-SW-Chains and MA-SSW-Chains algorithms on the 15 benchmark functions from the IEEE CEC 2013 LSGO competition using 1,000 dimensions. For extremely high-dimensional problems the proposal was compared with the GPU-MA-SW-Chains and random search methods on 1, 3, and 10 million variables. Experimental results were analyzed using non-parametric statistical analysis, which reported that MR-MA-SSW-Chains improves MA-SW-Chains and MA-SSW-Chains. On the other hand, significant differences are achieved with GPU-MA-SW-Chains and random search on extremely high-dimensional problems. Random search was overcome by the memetic methods which shows the superior performance of heuristics even on extremely high-dimensional problems with huge search spaces. Furthermore, the convergence of the fitness along evaluations and the efficiency of the fitness precomputation were evaluated.

In conclusion, we showed that MapReduce is a suitable framework to scale optimization methods to extremely high-dimensional problems. First results are reported for 10 million dimensions on IEEE CEC functions. In future work, we will explore other heuristics and components to scale optimization methods to even larger number of variables.

Acknowledgments

This research was supported by the Spanish Ministry of Economy and Competitiveness project TIN-2014-55252-P, and by FEDER funds. We gratefully acknowledge the anonymous reviewers for their helpful comments.

References

[1] N. Al-Madi, S. Ludwig, Scaling genetic programming for data classification using MapReduce methodology, in: Proc. of the World Congress on Nature and Biologically Inspired Computing (2013), pp. 132–139.

[2] K.K. Bali, R. Chandra, Multi-island competitive cooperative coevolution for real parameter global optimization, in: Proc. of the International Conference on Neural Information Processing (2015), volume 9491 LNCS, pp. 127–136.

[3] O. Barrière, E. Lutton, Experimental analysis of a variables size mono-population cooperative-coevolution strategy, in: Nature Inspired Cooperative Strategies for Optimization (2009), pp. 139–152.

[4] F.V. den Bergh, A.P. Engelbrecht, A cooperative approach to particle swarm optimization, IEEE Transactions on Evolutionary Computation 8 (2004) 225–239.

[5] C. Blum, M. Dorigo, The Hyper-Cube Framework for Ant Colony Optimization, IEEE Trans. on Systems, Man, and Cybernetics – Part B: Cybernetics 34 (2004) 1161–1172.

[6] S. Boyd, L. Xiao, A. Mutapcic, J. Mattingley, Notes on Decomposition Methods, Technical Report, Notes for EE364B, Standford University, 2007.

[7] J. Brest, B. Bošković, A. Zamuda, I. Fister, M. Maucec, Self-adaptive differential evolution algorithm with a small and varying population size, in: Proc. of the IEEE Congress on Evolutionary Computation (2012), pp. 1–8.

[8] A. Cano, J.M. Luna, A. Zafra, S. Ventura, A Classification Module for Genetic Programming Algorithms in JCLEC, Journal of Machine Learning Research 16 (2015) 491–494.

[9] X. Cao, H. Quiao, J. Keane, A low-cost pedestrian-detection system with a single optical camera, IEEE Transactions on Intelligent Transportation Systems 9 (2008) 58–67.

[10] R. Cheng, Y. Jin, A competitive swarm optimizer for large scale optimization, IEEE Trans. on Cybernetics 45 (2015) 191–204.

[11] J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, Communications of the ACM 51 (2008) 107–113.

[12] J. Dean, S. Ghemawat, MapReduce: A flexible data processing tool, Communications of the ACM 53 (2010) 72–77.

[13] K. Deb, A. Reddy, G. Singh, Optimal scheduling of casting sequence using genetic algorithms, Materials and Manufacturing Processes 18 (2003) 409–432.

[14] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm and Evolutionary Computation 1 (2011) 3–18.

[15] S. Droste, T. Jansen, I. Wegener, Optimization with randomized search heuristics—the (A)NFL theorem, realistic scenarios, and difficult functions, Theoretical Computer Science 287 (2002) 131–144.

[16] X. Du, Y. Ni, Z. Yao, R. Xiao, D. Xie, High performance parallel evolutionary algorithm model based on MapReduce framework, Int. J. of Computer Applications in Technology 46 (2013) 290–295.

[17] R. Etemaadi, M. Chaudron, Distributed optimization on super computers: Case study on software architecture optimization framework, in: Proc. of the Genetic and Evolutionary Computation Conference (2014), pp. 1125–1132.

[18] J. Fan, J. Wang, M. Han, Cooperative coevolution for large-sclae optimization based on kernel fuzzy clustering and variable trust region methods, IEEE Trans. on Fuzzy Systems 22 (2014) 829–839.

[19] P. Fazenda, J. McDermott, U.M. O'Reilly, A library to run evolutionary algorithms in the cloud using MapReduce, in: Proc. of the European Conference on Applications of Evolutionary Computation (2012), volume 7248 LNCS, pp. 416–425.

[20] C. García-Martínez, F.J. Rodríguez, M. Lozano, Role differentiation and malleable mating for differential evolution: an analysis on large-scale optimisation, Soft Computing 15 (2011) 2109–2126.

[21] C. García-Martínez, F.J. Rodríguez, M. Lozano, Arbitrary Function Optimisation with Metaheuristics, Soft Computing 16 (2012) 2115–2133.

[22] N. García-Pedrajas, C. Hervás-Martínez, J. Muñoz Pérez, Covnet: a cooperative coevolutionary model for evolving artificial neural netwoks, IEEE Transactions on Neural Networks 14 (2003) 575–596.

[23] Y. Guo, W. Bland, P. Balaji, X. Zhou, Fault tolerant mapreduce-mpi for hpc clusters, in: Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis (2015), pp. 34:1–12.

[24] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evolutionary Computation 9 (2001) 159–195.

[25] Q. He, T. Shang, F. Zhuang, Z. Shi, Parallel extreme learning machine for regression based on mapreduce, Neurocomputing 102 (2013) 52–58.

[26] D.W. Huang, J. Lin, Scaling Populations of a Genetic Algorithm for Job Shop Scheduling Problems using MapReduce, in: Proc. of the IEEE International Conference on Cloud Computing Technology and Science (2010), pp. 780–785.

[27] C. Jin, C. Vecchiola, R. Buya, MRPGA: An Extension of MapReduce for Parallelizing Genetic Algorithms, in: Proc. of the IEEE International Conference on eScience (2008), pp. 214–221.

[28] Y. Kim, K. Shim, M.S. Kim, J. Sup Lee, DBCURE-MR: An efficient density-based clustering algorithm for large data using MapReduce, Information Systems 42 (2014) 15–35.

[29] J. Koko, Parallel uzawa method for large-scale minimization of partially separable functions, Journal of Optimization Theory and Applications 158 (2013) 172–187.

[30] P. Korošec, K. Tashkova, J. Šilc, The Differential Ant-Stigmergy Algorithm for Large-Schale Global Optimization, in: Proc. of the IEEE World Congress on Computational Intelligence (2010), pp. 18–23.

[31] B. Lacroix, D. Molina, F. Herrera, Region based memetic algorithm for real-parameter optimisation, Information Sciences 262 (2014) 15–31.

[32] M. Lastra, D. Molina, J.M. Benítez, A high performance memetic algorithm for extremely high-dimensional problems, Information Sciences 293 (2015) 35–58.

[33] A. LaTorre, S. Muelas, J. Peña, A MOS-based dynamic memetic differential evolution algorithm for continuous optimization: a scalability test, Soft Computing 15 (2011) 2187–2199.

[34] A. LaTorre, S. Muelas, J. Peña, Large scale global optimization: experimental results with MOS-based hybrid algorithms, in: Proc. of the IEEE Congress on Evolutionary Computation (2013), pp. 1–8.

[35] A. LaTorre, S. Muelas, J. Peña, A comprehensive comparison of large scale global optimizers, Information Sciences 316 (2015) 517–549.

[36] M. Lescrenier, Partially separable optimization and parallel computing, Annals of Operations Research 14 (1988) 213–224.

[37] X. Li, K. Tang, M.N. Omidvar, Z. Yang, K. Qin, Benchmark Functions for the CEC'2013 Special Session and Competition on Large-Scale Global Optimization, Technical Report, Evolutionary Computation and Machine Learning Group, RMIT University, Australia, 2013.

[38] X. Li, K. Tang, P. Suganthan, Z. Yang, Editorial for the special issue of information sciences journal (isj) on nature-inspired algorithms for large scale global optimization, Information Sciences 316 (2015) 437–439.

[39] X. Li, X. Yao, Cooperatively coevolving particle swarms for large scale optimization, IEEE Trans. on Evolutionary Computation 16 (2012) 210–224.

[40] J. Liu, K. Tang, Scaling Up Covariance Matrix Adaptation Evolution Strategy Using Cooperative Coevolution, in: Proc. of the Intelligent Data Engineering and Automated Learning (2013), volume 8206 LNCS, pp. 350–357.

[41] M. Lozano, F. Herrera, D. Molina, Special Issue on scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems, Soft Computing 15 (2011) 2085–2087.

[42] S. Mahdavi, M.E. Shiri, S. Rahnamayan, Metaheuristics in large-scale global continues optimization: A survey, Information Sciences 295 (2015) 407–428.

[43] A.W. McNabb, C.K. Monson, K.D. Seppi, Parallel PSO using MapReduce, in: Proc. of the IEEE Congress on Evolutionary Computation (2007), pp. 7–14.

[44] X. Mingming, Z. Jun, C. Kaiquan, C. Xianbin, T. Ke, Cooperative coevolution with weighted random grouping for large-scale crossing waypoints locating in air route network, in: Proc. of the IEEE International Conference on Tools with Artificial Intelligence (2011), pp. 215–222.

[45] D. Molina, F. Herrera, Iterative hybridization of DE with local search for the CEC'2015 special session on large scale global optimization, in: Proc. of the IEEE Congress on Evolutionary Computation (2015), pp. 1974–1978.

[46] D. Molina, M. Lozano, C. García-Martínez, F. Herrera, Memetic algorithms for continuous optimisation based on local search chains, Evolutionary Computation 18 (2010) 27–63.

[47] D. Molina, M. Lozano, F. Herrera, MA-SW-Chains: memetic algorithm based on local search chains for large scale continuous global optimization, in: Proc. of the IEEE Congress on Evolutionary Computation (2010), pp. 1–8.

[48] D. Molina, M. Lozano, A. Sánchez, F. Herrera, Memetic algorithms based on local search chains for large scale continuous optimisation problems: MA-SSW-Chains, Soft Computing 15 (2011) 2201–2220.

[49] M.N. Omidvar, X. Li, Y. Mei, X. Yao, Cooperative coevolution with differential grouping for large scale optimization, IEEE Transactions on Evolutionary Computation 18 (2014) 378–393.

[50] M.N. Omidvar, X. Li, K. Tang, Designing benchmark problems for large-scale continuous optimization, Information Sciences 316 (2015) 419–436.

[51] L. Panait, S. Luke, Cooperative multi-agent learning: the state of the art, Autonomous Agents and Multi-agent Systems 11 (2005) 387–434.

[52] M. Pavlech, Framework for development of distributed evolutionary algorithms based on MapReduce, in: Proc. of the International DAAAM Symposium (2011), pp. 1475–1476.

[53] M.A. Potter, K.A. De Jong, Cooperative coevolution: an architecture for evolving coadapted subcomponents, Evolutionary Computation 8 (2000) 1–29.

[54] K. Sastry, D. Goldberg, X. Llora, Towards billion-bit optimization via a parallel estimation of distribution algorithm, in: Proc. of the Genetic and Evolutionary Computation Conference (2007), pp. 577–584.

[55] J. Shafer, S. Rixner, A. Cox, The Hadoop distributed filesystem: Balancing portability and performance, in: Proc. of the IEEE International Symposium on Performance Analysis of Systems Software (2010), pp. 122–133.

[56] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The hadoop distributed file system, in: Proc. of the IEEE Symposium on Mass Storage Systems and Technologies (2010), pp. 1–10.

[57] D. Swagatam, P.N. Suganthan, Differential Evolution: a survey of the state-of-the-art, IEEE Trans. on Evolutionary Computation 15 (2011) 4–31.

[58] K. Tang, X. Li, P. Suganthan, Z. Yang, T. Weise, Benchmark Functions for the CEC'2010 Special Session and Competition on Large-Scale Global Optimization, Technical Report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2009.

[59] R.K. Thompson, A.H. Wright, Additively Decomposable Fitness Functions, Technical Report, Dept. of Computer Science, University of Montana, 1996.

[60] L.Y. Tseng, C. Chen, Multiple trajectory search for Large Scale Global Optimization, in: Proc. of the IEEE Congress on Evolutionary Computation (2008), pp. 3052–3059.

[61] S. Ventura, C. Romero, A. Zafra, J.A. Delgado, C. Hervás, JCLEC: A Java framework for evolutionary computation, Soft Computing 12 (2007) 381–392.

[62] A. Verma, B. Cho, N. Zea, I. Gupta, R.H. Campbell, Breaking the MapReduce Stage Barrier, Cluster Computing 16 (2013) 191–206.

[63] A. Verma, X. Llora, D. Goldberg, R. Campbell, Scaling genetic algorithms using MapReduce, in: Proc. of the International Conference on Intelligent Systems Design and Applications (2009), pp. 13–18.

[64] A. Verma, X. Llorà, S. Venkataraman, D.E. Goldberg, R.H. Campbell, Scaling eCGA model building via data-intensive computing, in: Proc. of the IEEE Congress on Evolutionary Computation (2010), pp. 1–8.

[65] Y. Wang, B. Li, Two-stage based ensemble optimization for large-scale global optimization, in: Proc. of the IEEE Congress on Evolutionary Computation (2010), pp. 1–8.

[66] G.W. Woodford, C.J. Pretorius, M.C. du Plessis, Concurrent controller and simulator neural netwrok development for a differentially-steered robot in evolutionary robotics, Robotics and Autonomous Sytems 76 (2016) 80–92.

[67] N. Xiong, D. Molina, M.L. Ortiz, F. Herrera, A walk into metaheuristics for engineering optimization: principles, methods and recent trends, Int. Journal of Computational Intelligence Systems 8 (2015) 606–636.

[68] Z. Yang, K. Tang, X. Yao, Large scale evolutionary optimization using cooperative coevolution, Information Sciences 178 (2008) 2985–2999.

[69] Z. Yang, K. Tang, X. Yao, Multilevel cooperative coevolution for large scale optimization, in: Proc. of the IEEE Congress on Evolutionary Computation (2008), pp. 1663–1670.

[70] S.Z. Zhao, P.N. Suganthan, D. Swagatam, Dynamic Multi-Swarm Particle Swarm Optimizer with Sub-regional Harmony Search, in: Proc. of the IEEE Congress on Evolutionary Computation (2010), pp. 1–8.

[71] F. Zheng, C. Han, Y. Wang, Parallel SSLE algorithm for large-scale constrained optimization, Applied Mathematics and Computation 217 (2011) 5377–5384.

[72] C. Zhou, Fast parallelization of differential evolution algorithm Using MapReduce, in: Proc. of the Genetic and Evolutionary Computation Conference (2010), pp. 1113–1114.