Noname manuscript No.
(will be inserted by the editor)

A cloud-based enhanced Differential Evolution algorithm for parameter estimation problems in computational systems biology

Diego Teijeiro · Xoán C. Pardo · David R. Penas · Patricia González · Julio R. Banga · Ramón Doallo

Received: date / Accepted: date

Abstract Metaheuristics are gaining increasing recognition in many research areas, computational systems biology among them. Recent advances in metaheuristics can be helpful in locating the vicinity of the global solution in reasonable computation times, with Differential Evolution (DE) being one of the most popular methods. However, for most realistic applications, DE still requires excessive computation times. With the advent of Cloud Computing effortless access to large number of distributed resources has become more feasible, and new distributed frameworks, like Spark, have been developed to deal with large scale computations on commodity clusters and cloud resources. In this paper we propose a parallel implementation of an enhanced DE using Spark. The proposal drastically reduces the execution time, by means of including a selected local search and exploiting the available distributed resources. The performance of the proposal has been thoroughly assessed using challenging parameter estimation problems from the domain of computational systems biology. Two different platforms have been used for the evaluation, a local cluster and the Microsoft Azure public cloud. Additionally, it has been also compared with other parallel approaches, another cloud-based solution (a MapReduce implementation) and a traditional HPC solution (a MPI implementation).

Keywords Parallel Metaheuristics · Differential Evolution · Local Search · Cloud Computing · Spark

Grupo de Arquitectura de Computadores. Universidade da Coruña. Spain (E-mail: {diego.teijeiro,xoan.pardo,patricia.gonzalez,doallo}@udc.es) · BioProcess Engineering Group. IIM-CSIC. Spain (E-mail: {davidrodpenas,julio}@iim.csic.es)

1 Introduction

Many key problems in computational systems biology can be formulated and solved using global optimization techniques. The development of dynamic (kinetic) models is one of the current key issues in the field. Dynamics, usually represented as sets of nonlinear ordinary differential equations models, are used to explain function in biological systems. In recent years, research has been focused on scaling-up these kinetic models [2,19,20,31], from medium and large-scale up to the level of whole-cell models [17]. In this context, the problem of parameter estimation (model calibration) remains as a very challenging task [7,16]. Global optimization methods can be used to solve this type of problems. In particular, methods based on heuristics, and their combination (hybrids) with more traditional approaches, have shown promising results [4,5,34]. In any case, the complexity of the underlying models requires the use of efficient solvers to achieve adequate results in reasonable computation times. Differential Evolution (DE) [33] is one of the most popular methods, and it has been successfully used in many different areas [10]. However, in most realistic applications, this population-based method requires a very large number of evaluations (and therefore, large computation time) to obtain an acceptable result. Hence, different parallel DE schemes have been proposed, most of them focused on traditional parallel programming interfaces and infrastructures.

Recently, Cloud Computing has emerged as a new paradigm for on-demand delivery of computing resources. However, scientific computing community has been quite hesitant in using the cloud, simply because the traditional programming models do not fit well with the new paradigm. Furthermore, earliest cloud programming models, like MapReduce [11], do not allow most

scientific computations being efficiently run in the cloud. However more recent proposals like Spark [46] or Flink [15] have added improved support for iterative algorithms which, at first, make them more promising in executing scientific codes efficiently on cloud resources.

Two are the main objectives of this contribution. The first aim is to obtain a cloud-based implementation of the DE algorithm that achieves a good trade-off between exploration (diversification or global search) and exploitation (intensification or local search). This balance is at the core of modern metaheuristics [41]. To this end, a local search and a tabu list have been included to enhance the performance of DE in parameter estimation problems in systems biology. The second aim is to thoroughly assess the performance of the proposal using different infrastructures, such as a local cluster and a public cloud. The evaluation includes also a comparison with other parallel approaches: another cloud-based implementation using MapReduce, and a traditional HPC implementation using MPI. Thus, the results obtained in this paper can be particularly useful, not only for the computational systems biology community, but also for those interested in the potential of new cloud distributed frameworks for developing novel parallel metaheuristic methods. To this end, the source code is made publicly available.

The organization of the paper is as follows. Section 2 discusses related work. Section 3 presents a brief overview of the DE and the new features included in the proposal to improve the search. The Spark implementation of the proposed enhanced parallel DE is described in Section 4. Section 5 assesses the performance of the proposal. Finally, Section 6 concludes the paper.

2 Related Work

This section covers different approaches that follow any of the strategies explored in this work. First, we list different works that contribute to improve the performance of the classical DE algorithm either by means of modifications to enhance the original algorithm, or through parallel implementations of the DE. Note that the number of researches in this field is significant, thus, here we focus on those that relate more closely to the enhancements included in our proposal. Then, we briefly describe the few cloud-based proposals, focal point of this work, existing in the literature.

Many researches have tried to improve DE by proposing modifications to enhance the original algorithm. Interesting reviews can be found in [10,9]. In several cases, the original DE algorithm was improved with additional algorithmic components exploiting certain aspects of a given class of problems. In [45] a modified DE approach is proposed to improve the search performance by using generation-varying control parameters to prevent premature convergence to local minima. A hybrid algorithm using DE as an evolutionary framework and a crossover-based local search was proposed in [25,26]. A DE with Scale Factor Local Search was introduced in [40,24] for self-adaptive DE schemes. The use of a tabu list in the DE has also been applied in recent works [32,18,30].

On the other hand, several studies have considered parallel versions of DE, most of them focused on traditional parallel programming interfaces and infrastructures. We focus here on those approaches following an island-based model. A parallel synchronous approach was proposed in [36]. It is based on the distribution of the population data among different processors which communicate through data migrations and are managed by a central processor. Being implemented with synchronous communications, this proposal leads to low speedup results. A simple approach was also proposed in [27], consisting also of a master-slave architecture with several independent processes, which communicate through the filesystem. A more recent distributed DE implementation was presented in [3] exploiting an island-model with asynchronous communications.

Several other works studied improvements to island-model schemes. In [29], a complete study about the impact on the performance of different communication topologies between the islands was presented. Several studies suggest that randomization of the control parameters can be a propitious mechanism for enhancing the DE performance [6]. Different randomization schemes have been proposed to develop self-adaptive DE frameworks and investigate the effect of changing control parameters in distributed DE [47,44]. Two mechanisms to avoid the loss of diversity when the size of the population is small are described in [43]. The first one was based on shuffling: the individuals from a specific subpopulation were randomly reorganized. The second one, an update mechanism, changed and adapted scaling factors for each subpopulation. The results indicate that these techniques obtain a very significant performance when the dimensionality of the functions grow.

Research on cloud-oriented parallel metaheuristics, based mainly on the use of MapReduce, has also received increasing attention in recent years. Some proposals investigate how to apply MapReduce to parallelize the DE algorithm to be used in the Cloud. In [48] the fitness evaluation in the DE algorithm is performed in parallel using Hadoop (the well-known open-source MapReduce framework). However, the experimental results reveal that the extra cost of Hadoop DFS I/O operations and the system bookkeeping overhead signifi-

Table 1: Overview of the cloud-based DE proposals described in the related work.

	model		parallel scheme			framework		evaluation	
proposals	steady-state	generational	master-slave	island-based	**optimiz.**	Hadoop	Spark	cluster	cloud
Zhou et al. [48]		✔	✔			✔		✔	
Tagawa et al. [35]	✔		✔			✔		✔	
Daoudi et al. [8]		✔	✔			✔		✔	
Deng et al. [12]		✔	✔				✔	✔	
Teijeiro et al. [37]		✔	✔	✔			✔		✔
`eSiPDE`		✔		✔	✔		✔	✔	✔

cantly reduces the benefits of the parallelization. In [35], a concurrent implementation of the DE steady-state model based on MapReduce is proposed. However, the way the population is accessed limits its applicability to shared-memory architectures. In [8] a parallel implementation of DE based clustering using MapReduce is also proposed. This algorithm was implemented in three levels, each consisting of different DE operations.

An attempt to parallelize the DE algorithm using Spark was presented in [12]. However, in that work only the computation of the fitness values of the individuals is performed in parallel following a master-slave approach. An entire parallelization of the DE algorithm with Spark was explored in [37]. In that paper Spark-based implementations of two different parallel schemes of the DE algorithm, the master-slave and the island-based, were proposed and evaluated. Results showed that the island-based scheme is by far the best suited to the distributed nature of Spark. A thorough evaluation of the Spark-based island implementation can be found in [38]. It has been also compared in [39] with a MapReduce implementation, concluding that Spark outperforms MapReduce in this kind of iterative algorithms.

Table 1 summarizes the main features of the cloud-based DE proposals commented above, specifying: the algorithm model, the strategy followed in the parallelization, the inclusion of further optimizations to the basic DE algorithm, the distributed framework used, and the infrastructure where the evaluations have been performed. Our proposal (called `eSiPDE`) is also included in the table. There are two main contributions in this work with respect to our previous proposals [37–39]. First, we include further optimizations, a local search and a tabu list, to improve the convergence of the Spark-based island parallel DE. Second, we further compare the new enhanced DE algorithm with other parallel approaches: another cloud-based implementation using MapReduce and a traditional HPC implementation using MPI.

3 Differential Evolution

Differential Evolution (DE) [33] is an iterative mutation algorithm where vector differences are used to create new candidate solutions. Starting from an initial population matrix composed of NP D-dimensional solution vectors (individuals), DE attempts to achieve the optimal solution iteratively through changes in its vectors. Algorithm 1 shows the basic pseudocode for the DE algorithm. New individuals are generated in the population matrix, in each iteration, through operations (crossover - CR; mutation - F) performed among individuals of the matrix. Old solutions are replaced only when the fitness value of the objective function is better than the current one. A population matrix with optimized individuals is obtained as output of the algorithm. The best of these individuals are selected as solution close to optimal for the objective function of the model.

However, typical runtimes for many realistic problems are in the range from hours to days due to the large number of objective function evaluations needed, making the performance of the classical sequential DE unacceptable. Therefore, in order to improve the runtime of the DE algorithm, two main strategies have been explored. First, exploiting parallelism so as to reduce the computational time needed and to improve global search through diversification. Second, including a selected local search to enhance the method through intensification, drastically reducing the number of evaluations required.

3.1 Improving global search with a parallel cooperative scheme

The parallelization proposed in this work pursues the development of an efficient parallel variant of the serial DE. It accelerates the computation by performing separate evaluations in parallel. Besides, it also improves the convergence by stimulating the diversification in

the search and the cooperation between the parallel threads.

In the literature, different parallel models can be found [1] aiming to improve both the computational time and the number of iterations for convergence. The *master-slave* and the *island-based* models are the most popular. In the *master-slave* model the behaviour of the sequential DE is preserved by parallelizing the inner-loop of the algorithm, where a master processor distributes computations among the slave processors. The implementation of the DE master-slave model does not fit well with the distributed nature of frameworks like Spark [37]. The reason is that when the mutation strategy is applied to each individual, random different individuals have to be selected from the whole population. Considering that the population would certainly be partitioned and distributed among slaves, any solution to this problem would introduce an unfeasible communications overhead.

In the *island-based* model the population matrix is divided into subpopulations (*islands*) where the algorithm is executed isolated. Sparse individual exchanges are performed among islands to introduce diversity into the subpopulations. Thereby, the search avoids stagnation in local optima. Although the implementation of the *island-based* model in Spark drastically reduces the communications between islands, the scalability is heavily restrained by the small size of the DE population matrix. Thus, founded on the ideas outlined in [28], the *island-based* model can be used to perform a different DE in each island. A different population matrix and different combinations of CR and F values are used in each island to enhance diversity. These islands cooperate through sparse migrations, therefore modifying the systemic properties of the individual searches.

Algorithm 1: Differential Evolution algorithm (`seqDE`)

input : A population matrix P with size D x NP
output: A matrix P whose individuals were optimized

repeat
  for *each element i of the P matrix* **do**
    choose randomly different r_1, r_2, $r_3 \in [1, NP]$
    choose randomly an integer $j_r \in [1, D]$
    for $j \leftarrow 1$ **to** D **do**
      choose a randomly real $r \in [0, 1]$
      if $r \leq CR$ *or* $j = j_r$ **then**
        $u_i^{G+1}(j) \leftarrow x_{r1}^G(j) + F \cdot (x_{r2}^G(j) - x_{r3}^G(j))$
      else
        $u_i^{G+1}(j) \leftarrow x_i^G(j)$
      end
    end
    evaluate (u_i^{G+1})
    if $f(u_i^{G+1}) < f(x_i^G)$ **then**
      $x_i^{G+1} \leftarrow u_i^{G+1}$
    else
      $x_i^{G+1} \leftarrow x_i^G$
    end
  end
until *Stop conditions*;

3.2 Enhancing DE with local search and tabu list

Hybrid methods, that combine global with local search, have a long tradition in numerical optimization. In order to improve the computational effort required by the DE algorithm a local search has been added, thus, reducing the number of objective function evaluations required. The local search moves from solution to solution in the space of candidate solutions, applying local changes until an optimal solution is found or a time bound is elapsed. Different local solvers should be chosen to fit better with the problem at hand. In this work the NL2SOL [13] is used. NL2SOL is a method for solving non-linear least-squares problems that has demonstrated to be particularly effective for parameter estimation problems [14, 28].

One drawback of local search is that it tends to become stuck in suboptimal regions. To avoid this problem, the concept of *tabu list* is introduced in the algorithm. Tabu search enhances the performance of local methods by avoiding revisits to the same place during the search. This is achieved using memory structures that keep track of the visited solutions. If the vicinity of a potential solution has been previously visited within a certain short-term period it is marked as *tabu*. As a result, the algorithm does not consider that solution again. This technique improves the diversity among members of the population, and consequently contributes to the computational efficiency of the algorithm.

In the next section the proposed implementation of the enhanced Spark-based parallel DE is described in detail.

4 Enhanced Spark-based Parallel Differential Evolution

To understand the enhanced Spark-based parallel implementation of the DE algorithm, some previous insight into the way data is distributed and processed by Spark is needed. Spark uses the *resilient distributed dataset* (RDD) abstraction to represent fault-tolerant distributed data. RDDs are immutable sets of records that optionally can be in the form of key-value pairs.

Spark programs are run by a driver (the master in Spark terminology) which partitions RDDs and distributes the partitions to workers (the slaves in Spark terminology). The workers persist and transform the data and return results to the driver. There is no communication among workers. Shuffle operations (i.e. join, groupBy) that need data movement among workers through the network are expensive and should be avoided.

Our enhanced Spark-based parallel DE implementation (`eSiPDE`) follows the scheme shown in Figure 1. In the figure, boxes with solid outlines are RDDs. Partitions are shaded rectangles, darker if they are persistent in memory. A key-value pair RDD has been used to represent the population where each individual is uniquely identified by its key. There are two execution flows that run asynchronously in different threads of the Spark driver. The main flow is a version of the island-based parallel DE implementation (`SiPDE`) described in [37]. It has been modified in this work to allow for heterogeneous islands, and also to incorporate the result of a local search into the islands using a substitution strategy. The secondary flow executes an asynchronous local search on the best individual, found up to that moment, that is far enough away from those used in previous searches.

Some steps in the main flow of the algorithm are executed in a distributed fashion:

- The random generation and initial evaluation of individuals that form the population, implemented as a Spark map transformation.
- The evolution of the population. Every partition of the population RDD is considered to be an island, all with the same number of individuals. Islands evolve isolated during a number of evolutions. This number can be configured and is the same for all islands. During these evolutions every worker calculates mutations picking random individuals from its local partition only. As it has been said, the proposed enhanced parallel DE (`eSiPDE`) is an improvement of the island-based parallel DE (`SiPDE`) [37]. With this respect, `eSiPDE` enhances `SiPDE` by allowing islands to be heterogeneous, that is, having different combinations of CR and F values to enrich diversity.
- The migration strategy, which introduces diversity by exchanging selected individuals among islands every time the evolution of the islands ends. In order to evaluate the communications overhead, it has been implemented a custom Spark *partitioner* that randomly and evenly shuffles elements among partitions without replacement.
- The checking of the termination criterion, implemented as a Spark reduce action (a distributed OR operation).

The main flow repeats this evolution-migration loop until the termination criterion is met. Then the best individual is selected by means of a Spark reduce action (a distributed MIN operation).

An asynchronous local search runs concurrently with the main flow using a different thread on the Spark driver. As it can be seen in Figure 1, synchronization with the main flow takes place at two points:

- Before the evolution of the islands (label "1" in the figure), where a new search is initiated if no other is in progress. The candidate solution selected as input of the local search would be the best individual, found up to that moment, that was far enough away from candidate solutions used in previous searches. A tabu list is used to keep track of already explored candidate solutions and input selection is made by means of a Spark distributed filtering followed by a reduce action (a distributed MIN operation).
- Once the local search finishes (label "2" in the figure), if the candidate solution has been improved by the local search, a substitution strategy is applied in between the evolution and migration steps to incorporate it into the population. For this work, an strategy that replaces the worst individual in each island with the local search solution (only if it is better) is used. It has been implemented as a Spark map transformation.

Note that with this approach it would be at most one local search running concurrently with islands evolution at every moment. If the local search finishes before the islands evolution, its result is incorporated to the population once the evolution ends and a new local search is initiated before the following evolution. By the contrary, if the islands evolution finishes before the local search, a migration is done and a new evolution started without waiting for the local solver to end. This avoids the drawback of synchronous approaches in which the evolution of the population gets blocked waiting for a local search to finish. Note also that the input to the local search is selected from the whole population, so only one global tabu list is needed, and that its result is included in every island.

5 Experimental Results

In order to evaluate the Spark implementation proposed in this paper (`eSiPDE`), three challenging param-

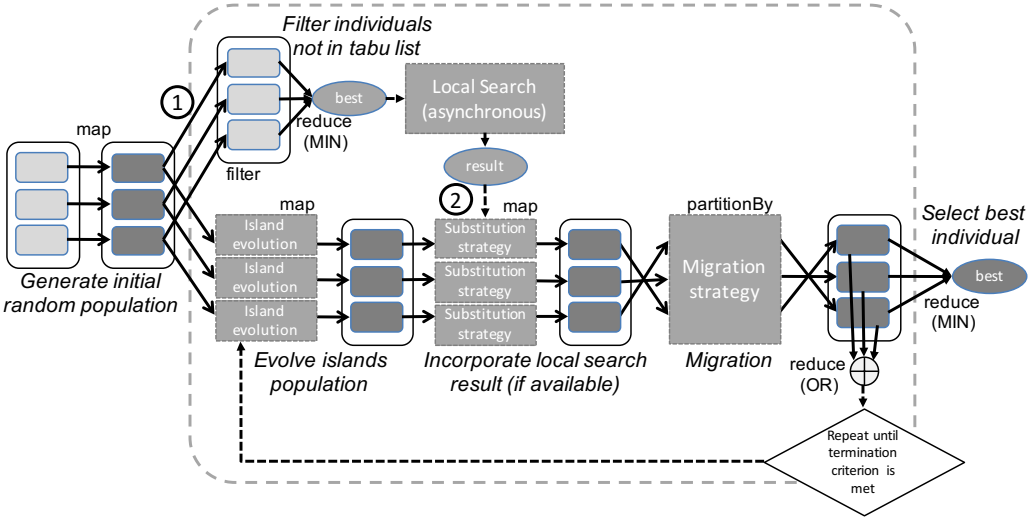


Fig. 1: Enhanced Spark implementation of the island-based DE algorithm (`eSiPDE`).

eter estimation problems from the domain of computational systems biology were considered. These problems are known to be particularly hard due to their ill-conditioning and non-convexity [23,42]:

- *Circadian* model: parameter estimation in a dynamic model of the circadian clock in the plant *Arabidopsis thaliana*, as presented in [22]. The model consists of 7 ordinary differential equations with 27 parameters (13 of them were estimated) with data sets from 2 experiments.
- *NFKB* model: this problem is based on the model in [21] and consists of 15 ordinary differential equations with 29 parameters and data sets from 2 experiments.
- *3-step pathway* model: problem considering a 3-step generic and highly non-linear pathway with 8 differential equations and 36 parameters, and data sets from 16 experiments, as presented in [23].

For the experimental testbed two different platforms have been used. First, experiments were conducted in our local cluster Pluton, that consists of 16 nodes powered by two octa-core Intel Xeon E5-2660 CPUs with 64 GB of RAM, and connected through an InfiniBand FDR network. Second, experiments were deployed with default settings in the Microsoft Azure public cloud using an standard HDInsight Spark cluster with A3 instances (4 cores, 7GB) for head and worker nodes. Unless otherwise noted, Scala v2.10 was the programming language and Spark v1.4.1 the distributed framework used in the experiments. In both testbeds, each experiment was executed a number of 20 independent runs. Note that, since Spark runs on the Java Virtual Machine (JVM), usual precautions (i.e. warm-up phase, effect of garbage collection) have been taken into account to avoid distortions on the measures.

As described in Section 3, the proposed implementation (`eSiPDE`) can be used in two different manners: (i) dividing the population among islands and using the same CR and F parameters for every island (`homogeneous` approach), and (ii) attempting a more thorough exploration of the solution space by means of the cooperation between different DE with different F and CR parameters in each island (`heterogeneous` approach). We compare the performance of both homogeneous and heterogeneous approaches with the performance of a sequential implementation of the classical DE (`seqDE`) and the implementation of the island-based parallel DE (`SiPDE`) described in [37].

There are many configurable parameters in the classical DE algorithm, such as the mutation scaling factor (F), the crossover constant (CR) or the mutation strategy (MSt). The selection of these parameters may have a great impact in the algorithm performance. Since the objective of this work is not to evaluate their impact, only results for one configuration are reported here. Previous tests have been done to select a configuration that leads to reasonable computation times. For all the experiments we used MSt=DE/rand/1. For testing the homogeneous configuration of `eSiPDE`, F=0.9 and CR=0.8 were used, while for the heterogeneous configuration different combination of CR=$\{0.2,0.7,0.8,0.9\}$ and F=$\{0.8,0.9\}$ values were randomly selected for each island. Besides, in island-based parallel DE algorithms, new parameters have to be also considered, such as the migration frequency (μ) or the island size (λ). In the following experiments the island size has been $\lambda =$

$NP/nproc$ and the migration frequency has been set to 200 local iterations between migrations. Nevertheless, the proposal can be applied to any other configuration parameters. Also, it is worth noting that further performance improvements can be achieved by further fine-tuning settings.

Since the aim of this work is to accelerate the execution time required for convergence in complex parameter estimation problems, the best way to fairly assess the performance of the proposal is to define a *value-to-reach* (VTR) to be used as stopping criteria for the algorithm. However, in the 3-step pathway and the NFKB benchmarks the execution of only one test could take several days to complete. Thus, we decided to use as stopping criterium: (a) a VTR=1e-5 for the circadian benchmark, evaluating its performance from an horizontal view; and (b) a predefined effort of maximum execution time $T_{max} = 1000s$ for the 3-step pathway and the NFKB benchmarks, assessing their performance from a vertical view.

Results for the Circadian benchmark in cluster Pluton are shown in Table 2. This table displays, for each experiment, the number of cores (*#np*) used, the mean number of evaluations required (*#evals*), the mean number of migrations (*#mig.*), the mean and the median of the execution times (*time(s)*), and the speedup achieved versus the `seqDE`. Due to the large dispersion in the obtained results for the `eSiPDE` implementation, the speedup was calculated using the median of the measures. Note that the number of cores matches the number of islands used. Results show that the parallelization improves the execution time required for convergence by performing the evaluations in parallel. `SiPDE` achieves already a good speedup versus the sequential algorithm (`seqDE`). However, the local search included in the `eSiPDE` implementation significantly reduces the execution time required for convergence by decreasing the number of evaluations. Note the radical reduction in the number of migrations when the local search is used. Moreover, the diversification introduced in the heterogeneous approach outperforms the homogeneous approach, specially when the number of islands grows.

Since the values in the table hide the underlying distribution, that in this kind of stochastic problems is very important, Figure 2 shows the bean plots to compare the distribution of the homogeneous versus the heterogeneous configuration of the `eSiPDE` implementation. Note that the logarithmic scale has been used in the y axis and the median of each distribution is also shown in each bean. It can be noted that for two islands the performance of the homogeneous configuration was slightly better because the heterogeneous configuration exhibited more outliers. However, when the number of islands increases, the heterogeneous configuration drastically reduces the dispersion in the results and achieves better performance.

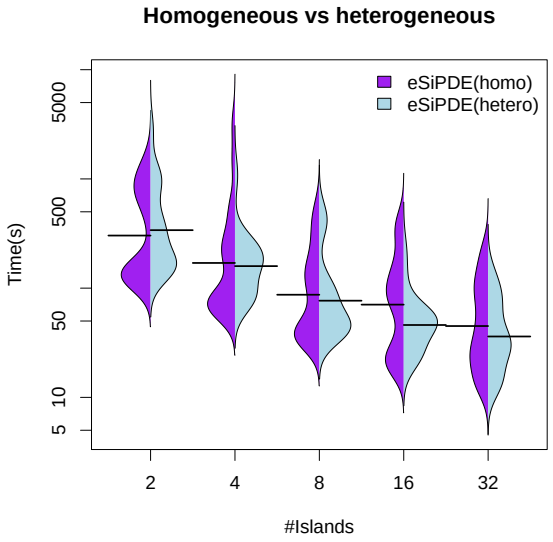


Fig. 2: Bean plots comparing different DE strategies in the Circadian benchmark.

Results for 3-step pathway and NFKB benchmarks are shown in Table 3. This table displays, for each experiment, the number of cores (*#np*) used, the average of the evaluations performed (*#evals*), and the average of the best value for each run (*fbest*). Results show that the parallelization improves the convergence rate since, in the same amount of time, more evaluations are executed in parallel achieving better quality solutions.

For 3-step pathway benchmark, Table 4 shows the number of executions from a total of 20 samples (%hits) that achieved convergence using a VTR=100 in a maximum time of 1000s, as well as the mean and minimum time of all those executions that reached the VTR. As it can be seen, as the number of islands grows, the number of executions that achieve the quality solution increases. These results show the effectiveness of the parallel algorithm in terms of quality of the solution. Also, it should be noted, that the heterogeneous configuration achieves always better results in terms of execution times.

To better illustrate the improvement in convergence time, Figure 3 shows the convergence curves for the three benchmarks using the sequential algorithm and the parallel implementations with 16 islands. The convergence curve represents the current best objective function value as the algorithm proceeds. The convergence curves depicted here are those that fall in the median values of the results distribution. It can be seen that, as expected, the local solver improves the con-

Table 2: Performance evaluation of different DE implementations for the Circadian benchmark in Pluton. Parameters: D=13, NP=256, VTR=1e-5.

method	#np	#evals	#mig.	time(s)		speedup
				mean±std	median	
seqDE	1	6,437,670	-	40883.39±3712.56	40916.76	-
SiPDE	2	5,980,416	117	19275.65±1281.63	19015.77	2.15
	4	5,729,536	112	9305.30±1038.59	9071.51	4.51
	8	3,904,256	74	3319.33±296.88	3256.62	12.56
	16	1,835,776	36	790.97±90.50	815.51	50.17
	32	1,577,216	30	348.36±43.47	355.05	115.24
eSiPDE (homo)	2	179,456	3.5	472.41±441.29	143.80	284.54
	4	230,656	4.5	388.31±736.39	104.44	391.77
	8	171,776	3.3	134.01±140.78	75.26	543.67
	16	225,536	4.4	115.48±119.04	77.82	525.79
	32	235,776	4.6	67.60±63.63	40.56	1008.55
eSiPDE (hetero)	2	161,536	3.1	524.28±631.98	311.08	131.53
	4	120,576	2.3	204.81±217.42	165.09	247.85
	8	128,256	2.5	115.51±135.43	45.76	894.16
	16	107,776	2.1	54.81±43.34	48.07	851.19
	32	161,536	3.2	46.85±36.01	31.55	1296.89

Table 3: Performance evaluation of the 3-step pathway and NFKB benchmarks in Pluton. Stopping criterion: predefined effort, $T_{max} = 1000s$. Parameters for 3-step pathway: D=36, NP=512. Parameters for NFKB: D=29, NP=512.

	method	#np	#evals	fbest
3-step pathway	seqDE	1	90,624	820.54
	SiPDE	2	191,232	753.52
		4	358,912	711.55
		8	653,312	690.06
		16	1,179,392	632.65
	eSiPDE (homo)	2	209,483	573.16
		4	369,972	363.18
		8	572,015	126.26
		16	945,646	92.13
	eSiPDE (hetero)	2	199,624	468.31
		4	350,903	305.97
		8	552,291	102.56
		16	912,968	91.52
NFKB	seqDE	1	21,274	0.06868
	SiPDE	2	44,032	0.06051
		4	81,408	0.05472
		8	143,104	0.05208
		16	239,104	0.04980
	eSiPDE (homo)	2	44,334	0.03295
		4	82,748	0.03358
		8	146,516	0.03386
		16	240,678	0.03340
	eSiPDE (hetero)	2	43,930	0.03268
		4	84,328	0.03365
		8	143,436	0.03256
		16	231,715	0.03719

vergence rate in all the benchmarks. Also the heterogeneous configuration exhibits a slightly better performance than the homogeneous one.

Table 4: Performance evaluation of the 3-step pathway using as stopping criterion the combination of a predefined effort ($T_{max} = 1000s$) and quality of solution ($VTR = 100$).

method	#np	%hits	time(s)	
			mean	min
seqDE	1	0%	-	-
eSiPDE (homo)	2	0%	-	-
	4	10%	927	890
	8	25%	818	563
	16	85%	632	188
eSiPDE (hetero)	2	0%	-	-
	4	5%	774	774
	8	30%	693	361
	16	75%	477	150

Finally, in order to evaluate the performance of the proposal in a public cloud, some experiments were conducted in the Microsoft Azure public cloud. As it can be seen in Table 5, the proposal achieves similar results in Azure as the ones obtained in the local cluster in terms of convergence (number of evaluations) and scalability. However, the overheads introduced in Azure due to virtualization and use of non-dedicated resources in a multitenant platform are not negligible. The execution times of Azure are between 1.3x and 1.4x times worst than those of Pluton. Bean plots comparing the results obtained in both platforms for the heterogeneous configuration are shown in Figure 4. This figure clearly shows, not only the larger execution time but also the larger dispersion in the results obtained in Azure (note the logarithmic scale in the y axis).

Table 5: Performance evaluation of different DE implementations for the Circadian benchmark in Azure. Parameters: D=13, NP=256, VTR=1e-5.

method	#np	#evals	#mig.	time(s)		speedup
				mean±std	median	
seqDE	1	6,554,317	-	95294.70±5623.22	95286.86	-
SiPDE	2	6,180,096	121	47895.80±5091.67	49066.32	1.99
	4	5,642,496	110	21106.12±1549.87	20732.02	4.52
	8	3,917,056	76	11449.79±1951.18	11260.30	8.32
	16	1,899,776	37	3246.46±376.04	3178.94	29.35
eSiPDE (homo)	2	100,096	1.9	725.16±392.86	734.27	129.77
	4	199,936	3.9	874.22±1362.30	393.79	242.01
	8	87,296	1.7	177.68±90.83	111.53	854.39
	16	171,776	3.4	216.61±177.06	130.16	732.08
eSiPDE (hetero)	2	102,656	2.0	745.88±656.98	383.50	248.47
	4	120,576	2.3	453.78±431.47	384.70	247.69
	8	156,416	3.0	355.51±347.84	200.70	474.78
	16	135,936	2.6	160.30±151.85	112.83	844.48

5.1 Comparison with other parallel approaches

Several tests have been also performed to assess how competitive the Spark parallel implementation can be with respect to other parallel approaches.

Since MapReduce is still the de-facto standard for large scale data-intensive applications, it has been selected as representative of other cloud-based approaches for the comparison. We have compared a MapReduce implementation of SiPDE using Hadoop v2.7.1 and Java v1.7.0. Figure 5 shows some bean plots that allow for an easy comparison of the execution times obtained using the MapReduce and the SiPDE implementations in the local cluster. Note that not only the execution time is larger for the MapReduce implementation but also the dispersion of the results obtained is bigger.

The experimental results show that MapReduce has significant higher overhead per iteration than Spark mainly caused by longer task initialization times and HDFS access. To evaluate this overhead we have used a modified version of our implementation in which the evolution of the population was removed. This modified implementation was executed for a total of 8 *evolution-migration* iterations and the overhead of each iteration was measured separately in order to assess differences between them. Figure 6 shows the results obtained both for the Spark and the MapReduce implementations in the local cluster. As it can be seen, the first iteration in the Spark implementation is always the most time-consuming (it corresponds to the outliers in the box plots). However, the rest of the iterations show lower overhead and lower dispersion in the results. By the contrary, in MapReduce there is no significant difference between the first and the subsequent iterations. The figures clearly indicate a higher overhead and large dispersion in the results, being the mean overhead of each iteration 17.95±2.50s versus the 0.027±0.006s in Spark.

In order to evaluate the competitiveness of the proposed cloud-based solution with a traditional HPC solution, we have also compared the Spark eSiPDE implementation with an MPI implementation. The same previous experiments were carried out with the implementation of the asynchronous parallel enhanced DE (asynPDE) described in [28]. This implementation is coded in C and uses the OpenMPI library. It must be noted that, as already available implementations in C/C++ and/or FORTRAN existed for all the benchmarks, we have wrapped them in the Scala code of eSiPDE by using Scala native interfaces (i.e JNI, JNA, SNA). Thus, the code used for the benchmark function evaluation has been the same in both the asynPDE and eSiPDE implementations.

To perform the fairest comparison, the MPI implementation includes also a local solver and a tabu list, like eSiPDE, to improve the convergence rate of the DE. Results for these experiments are reported in Table 6. This table displays, for each experiment, the number of cores (*#np*) used, the mean number of evaluations needed (*#evals*), and the mean of the execution times (*time(s)*). The homogeneous configuration with the following parameters: F=0.9, CR=0.8, NP=256, DE/rand/1 as mutation strategy, and a VTR=1e-5 as stopping criterion, has been used in all the cases. As it can be observed, the MPI implementation achieves convergence between 5 and 7 times more quickly than the Spark implementation. This is mostly because it also achieves an important reduction in the number of function evaluations required (between 2x and 3x). Two can be the main causes, both of them arising from the inherent features of the programming paradigm used in each implementation. First, since the communication

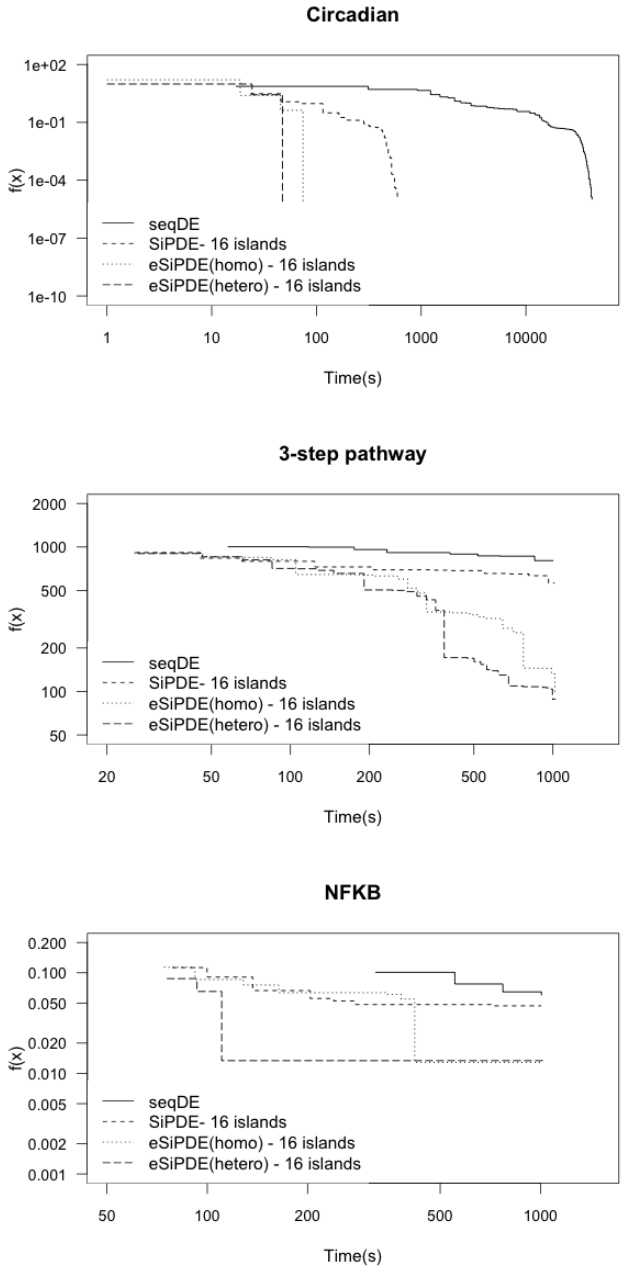


Fig. 3: Convergence curves: Circadian using as stopping criterium a VTR=1e-5, 3-step pathway and NFKB using as stopping criterium a predefined effort of $T_{max} = 1000s$.

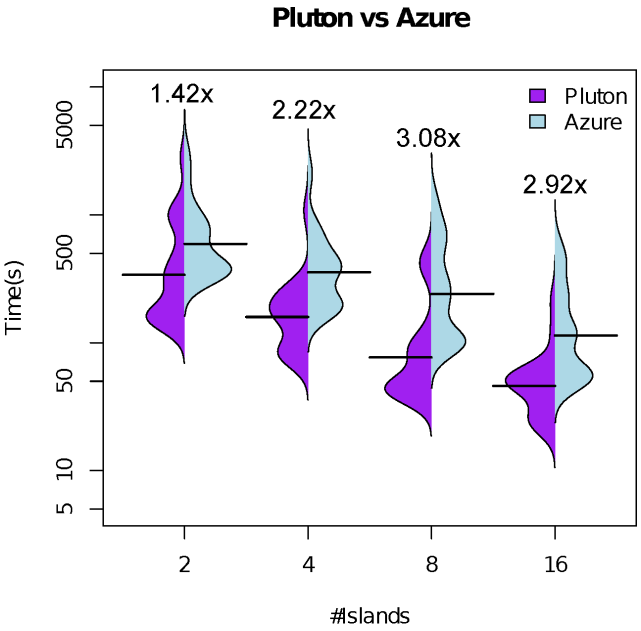


Fig. 4: Bean plots comparing execution times for the Circadian benchmark in the local cluster Pluton and the Azure public cloud for the heterogeneous configuration. The speedup achieved in Pluton vs Azure is displayed on top of each bean.

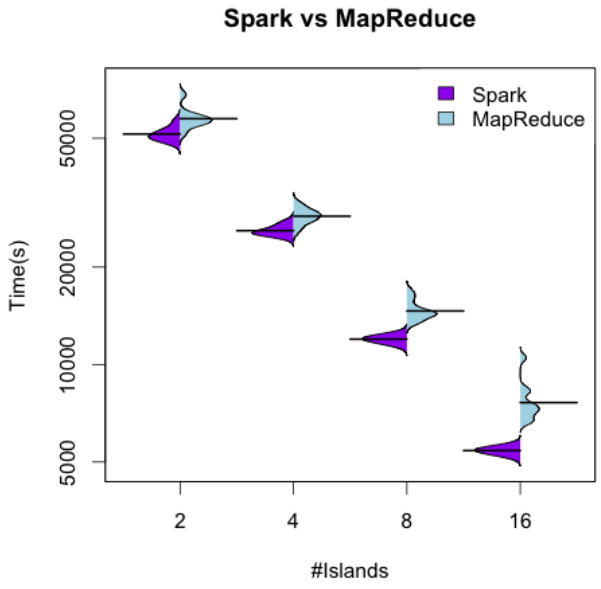
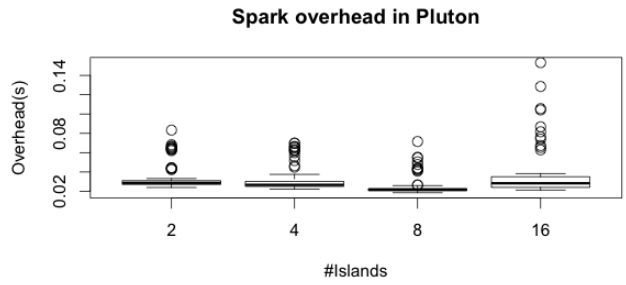
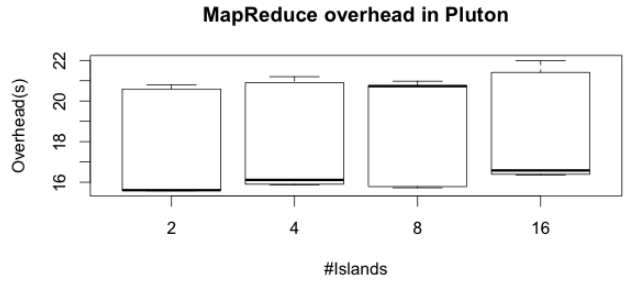


Fig. 5: Bean plots comparing Spark `SiPDE` vs MapReduce implementations in cluster Pluton for the Circadian benchmark. Parameters: D=13, NP=640, VTR=1e-5.

among workers is not allowed in Spark, the migration strategy is implemented with a partitioner that introduces an implicit synchronization step in the Spark implementation. The MPI implementation, on the contrary, performs the information exchange between islands through non-blocking asynchronous message passing operations. Another consequence of the lack of communications between workers in Spark is that the fulfillment of the stopping criterion by one ore more is-



(a) Spark overhead



(b) MapReduce overhead

Fig. 6: Boxplot of the overhead times per evolution-migration iteration in Pluton.

Table 6: Comparison of Spark and MPI parallel DE implementations for the Circadian benchmark in the local cluster Pluton and the Azure public cloud. Parameters: F=0.9, CR=0.8, NP=256, MtSt=DE/rand/1, and VTR=1e-5.

	method	#np	#evals	time(s)
Pluton	Spark eSiPDE	2	179,456	472.41±441.29
		4	230,656	388.31±736.39
		8	171,776	134.01±140.78
		16	225,536	115.48±119.04
	MPI asynPDE	2	78,276	94.62±66.75
		4	78,903	49.23±35.79
		8	79,992	26.38±21.12
		16	87,341	17.13±14.11
Azure	Spark eSiPDE	2	102,656	745.88±656.98
		4	120,576	453.78±431.47
		8	156,416	355.51±347.84
		16	135,936	160.30±151.85
	MPI asynPDE	2	70,332	201.49±110.32
		4	57,195	84.68±2.55
		8	69,469	54.45±22.06
		16	72,244	30.54±5.07

lands during island evolution cannot be informed to the rest until the reduce operation at the end of the stage (see Figure 1). Thus, the Spark implementation cannot stop just right when the stopping criterion is reached (as the MPI one does). Second, the migration strategy is different in both implementations. In the MPI implementation a selection of the best individuals in one island replace the worst individuals in the neighbour. In the Spark implementation a partitioner randomly and evenly shuffles elements among islands without replacement. Hence, to allow for a further comparison, Figure 7 shows the number of evaluations per second and core (*eval/s/core*) achieved for both implementations and the two platforms used. Note that this metric includes not only the CPU time for the evaluation itself but also the communication time and other implementation overheads. We encountered that the number of evaluations per second and core of the MPI implementation was between 2.18x and 2.69x times that of the Spark implementation in Pluton, and between 2.54x and 2.90x in the case of Azure. However, note that in Pluton the MPI implementation achieves more than 400 eval/s/core while in Azure it only achieves around 150 eval/s/core. Another interesting result that this figure illustrates is the fact that the number of eval/s/core decreases with the number of cores. This happens for both implementations and both platforms, but its impact is larger for the MPI implementation in Pluton. The reason is that the computation time decreases with the number of cores due to the tasks distribution. In the MPI implementation, the number of communications increases with the number of cores, thus, the trade-off between computation and communication is not preserved with the number of cores. By the contrary, in the Spark implementation, the number of communications remains constant with the number of cores. However, as the number of cores grows this amount of communications are spread between a large number of nodes which also impacts on the computation time/communication time ratio.

All these results show that, as it was expected, the MPI implementation outperforms Spark in terms of execution times. This is mainly due to its low level programming language and reduced overhead. Nevertheless, there are other tradeoffs to be concerned with, apart from efficiency. The Spark implementation should be positively considered since it allow easier programmability and because it also presents further advantages, such as native support to node failure and data replication.

6 Conclusions

In this paper, we presented a cloud based approach for parameter estimation problems in computational systems biology using an enhanced Differential Evolution algorithm. The proposal aims to benefit from the

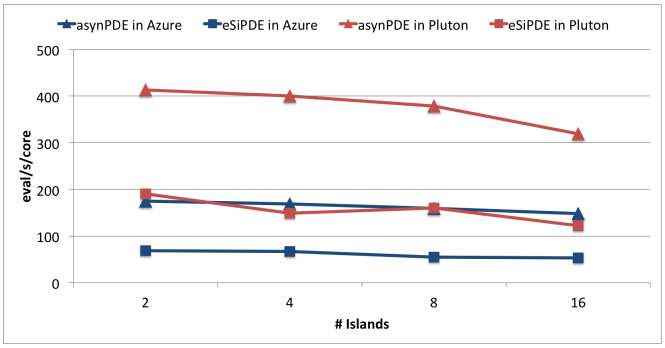


Fig. 7: Number of evaluations per second and core (evals/s/core) achieved by `asynPDE` vs `eSiPDE` in the local cluster Pluton and the Azure public cloud.

exploration abilities of DE and the exploitation abilities of efficient local search. The method improves global search through a parallel implementation based on a cooperative island-model. The local search, on its turn, is improved including a local solver, together with a tabu list, that exploits the structure of parameter estimation problems in systems biology. The enhancement in the local search is fundamental to successfully exploit the special characteristics of these problems, which are typically very ill-conditioned and highly multimodal.

The proposal has been implemented using Spark and thoroughly evaluated with three challenging parameter estimation problems from the domain of computational systems biology on two different platforms: a local cluster and a virtual cluster on the Microsoft Azure public cloud. Results show that the enhanced DE significantly reduces the execution time required for convergence in all the benchmarks. Besides, using cloud resources shows similar behaviour in terms of convergence and scalability as using resources from a local cluster, but at the expense of a not negligible overhead.

Finally, a comparison with other parallel approaches has been performed: a MapReduce implementation, to compare with the de-facto standard for cloud-based applications, and a MPI implementation, to compare with traditional HPC solutions. The results conclude that, on the one hand, Spark presents better support for iterative algorithms than MapReduce, reducing the overhead between the first and subsequent iterations. On the other hand, as it was expected, the MPI implementation outperforms Spark in terms of processing speed. But Spark can be still of interest due to its easier programmability and inherent support to node failure and data replication.

Although the proposed Spark implementation was designed and tested with focus on parameter estimation problems in computational systems biology, it can also be applied to solve arbitrary global optimization problems. In particular, we believe that both the description of the implementation and the results obtained in this work can be useful for those interested in the potential of new cloud-based programming models for the development of novel parallel metaheuristic methods.

The source code is publicly available at: `https://bitbucket.org/xcpardo/sipde`.

Acknowledgements This research received financial support from the Spanish Government (and the FEDER) through the projects DPI2014-55276-C5-2-R, TIN2013-42148-P and TIN2016-75845-P, and from the Galician Government under the Consolidation Program of Competitive Research Units (Network Ref. R2016/045 and Project Ref. GRC2013/055), all of them cofunded by FEDER funds of the EU. We also acknowledge Microsoft Research for being awarded with a sponsored Azure account.

References

1. Alba, E., Luque, G., Nesmachnow, S.: Parallel metaheuristics: recent advances and new trends. International Transactions in Operational Research **20**(1), 1–48 (2013)
2. Almquist, J., Cvijovic, M., Hatzimanikatis, V., Nielsen, J., Jirstrand, M.: Kinetic models in industrial biotechnology - Improving cell factory performance. Metabolic Engineering pp. 1–22 (2014)
3. Apolloni, J., García-Nieto, J., Alba, E., Leguizamón, G.: Empirical evaluation of distributed differential evolution on standard benchmarks. Applied Mathematics and Computation **236**, 351–366 (2014)
4. Ashyraliyev, M., Fomekong-Nanfack, Y., Kaandorp, J.A., Blom, J.G.: Systems biology: parameter estimation for biochemical models. Febs Journal **276**(4), 886–902 (2009)
5. Banga, J., Balsa-Canto, E.: Parameter estimation and optimal experimental design. Essays Biochem **45**, 195–210 (2008)
6. Brest, J., Greiner, S., Boskovic, B., Mernik, M., Zumer, V.: Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. Evolutionary Computation, IEEE Transactions on **10**(6), 646–657 (2006)
7. Cedersund, G., Samuelsson, O., Ball, G., Tegnér, J., Gomez-Cabrero, D.: Uncertainty in biology: a computational modeling approach, chap. Optimization in Biology Parameter Estimation and the Associated Optimization Problem, pp. 177–197. Springer International Publishing, Cham (2016)
8. Daoudi, M., Hamena, S., Benmounah, Z., Batouche, M.: Parallel differential evolution clustering algorithm based on MapReduce. In: 6th International Conference of Soft Computing and Pattern Recognition (SoCPaR), pp. 337–341. IEEE (2014)
9. Das, S., Mullick, S.S., Suganthan, P.: Recent advances in differential evolution–an updated survey. Swarm and Evolutionary Computation **27**, 1–30 (2016)
10. Das, S., Suganthan, P.N.: Differential evolution: a survey of the state-of-the-art. IEEE transactions on evolutionary computation **15**(1), 4–31 (2011)
11. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: The 6th USENIX Symposium on Operating Systems Design and Implementation (2004)

12. Deng, C., Tan, X., Dong, X., Tan, Y.: A parallel version of differential evolution based on resilient distributed datasets model. In: Bio-Inspired Computing-Theories and Applications, pp. 84–93. Springer (2015)
13. Dennis Jr., J.E., Gay, D.M., Welsch, R.E.: Algorithm 573: Nl2sol - an adaptive nonlinear least-squares algorithm [e4]. ACM Trans. Math. Softw. **7**(3), 369–383 (1981)
14. Egea, J., Rodríguez-Fernández, M., R. Banga, J., Martí, R.: Scatter search for chemical and bio-process optimization. Journal of Global Optimization **37**(3), 481–503 (2007)
15. Ewen, S., Tzoumas, K., Kaufmann, M., Markl, V.: Spinning fast iterative data flows. CoRR **abs/1208.0088** (2012). URL `http://arxiv.org/abs/1208.0088`
16. Gábor, A., Banga, J.R.: Robust and efficient parameter estimation in dynamic models of biological systems. BMC systems biology **9**(1), 74 (2015)
17. Karr, J.R., Sanghvi, J.C., Macklin, D.N., Gutschow, M.V., Jacobs, J.M., Bolival, B., Assad-Garcia, N., Glass, J.I., Covert, M.W.: A whole-cell computational model predicts phenotype from genotype. Cell **150**(2), 389–401 (2012)
18. Kushida, J.I., Oba, K., Hara, A., Takahama, T.: Solving quadratic assignment problems by differential evolution. In: 6th International Conference on Soft Computing and Intelligent Systems, and 13th International Symposium on Advanced Intelligence Systems, SCIS/ISIS 2012, pp. 639–644 (2012)
19. Le Novère, N.: Quantitative and logic modelling of molecular and gene networks. Nature Reviews Genetics **16**(3), 146–158 (2015)
20. Link, H., Christodoulou, D., Sauer, U.: Advancing metabolic models with kinetic information. Current Opinion in Biotechnology **29**, 8–14 (2014)
21. Lipniacki, T., Paszek, P., Brasier, A., Luxon, B., Kimmel, M.: Mathematical model of nf-κb regulatory module. Journal of Theoretical Biology **228**(2), 195–215 (2004)
22. Locke, J., Millar, A., Turner, M.: Modelling genetic networks with noisy and varied experimental data: The circadian clock in arabidopsis thaliana. Journal of Theoretical Biology **234**(3), 383–393 (2005)
23. Moles, C., Mendes, P., R. Banga, J.: Parameter estimation in biochemical pathways: A comparison of global optimization methods. Genome Research **13**(11), 2467–2474 (2003)
24. Neri, F., Tirronen, V., Kärkkäinen, T.: Enhancing differential evolution frameworks by scale factor local search - part ii. In: 2009 IEEE Congress on Evolutionary Computation, CEC 2009, pp. 118–125 (2009)
25. Noman, N., Iba, H.: Enhancing differential evolution performance with local search for high dimensional function optimization. In: GECCO 2005 - Genetic and Evolutionary Computation Conference, pp. 967–974 (2005)
26. Noman, N., Iba, H.: Accelerating differential evolution using an adaptive local search. IEEE Transactions on Evolutionary Computation **12**(1), 107–125 (2008)
27. Ntipteni, M.S., Valakos, I.M., Nikolos, I.K.: An asynchronous parallel differential evolution algorithm. In: Proceedings of the ERCOFTAC conference on design optimisation: methods and application (2006)
28. Penas, D., Banga, J., González, P., Doallo, R.: Enhanced parallel differential evolution algorithm for problems in computational systems biology. Applied Soft Computing **33**, 86–99 (2015). DOI http://dx.doi.org/10.1016/j.asoc.2015.04.025. URL `http://www.sciencedirect.com/science/article/pii/S1568494615002525`
29. Ruciński, M., Izzo, D., Biscani, F.: On the impact of the migration topology on the island model. Parallel Computing **36**(10-11), 555–571 (2010)
30. Schneider, E., Krohling, R.A.: Differential evolution and tabu search to find multiple solutions of multimodal optimization problems. In: Soft Computing in Industrial Applications, *Advances in Intelligent Systems and Computing*, vol. 223, pp. 61–69. Springer International Publishing (2014)
31. Smallbone, K., Mendes, P.: Large-scale metabolic models: from reconstruction to differential equations. Industrial Biotechnology **9**(4), 179–184 (2013)
32. Srinivas, M., Rangaiah, G.: Differential evolution with tabu list for global optimization and its application to phase equilibrium and parameter estimation problems. Industrial and Engineering Chemistry Research **46**(10), 3410–3421 (2007)
33. Storn, R., Price, K.: Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization **11**(4), 341–359 (1997)
34. Sun, J., Garibaldi, J.M., Hodgman, C.: Parameter estimation using metaheuristics in systems biology: a comprehensive review. Computational Biology and Bioinformatics, IEEE/ACM Transactions on **9**(1), 185–202 (2012)
35. Tagawa, K., Ishimizu, T.: Concurrent differential evolution based on MapReduce. International Journal of Computers **4**(4), 161–168 (2010)
36. Tasoulis, D.K., Pavlidis, N.G., Plagianakos, V.P., Vrahatis, M.N.: Parallel differential evolution. In: IEEE Congress on Evolutionary Computation, CEC2004, vol. 2, pp. 2023–2029. IEEE (2004)
37. Teijeiro, D., Pardo, X.C., González, P., Banga, J.R., Doallo, R.: Implementing parallel differential evolution on Spark. In: Applications of Evolutionary Computation. Lecture Notes in Computer Science, Vol. 9598, pp. 75–90. Springer (2016)
38. Teijeiro, D., Pardo, X.C., González, P., Banga, J.R., Doallo, R.: Towards cloud-based parallel metaheuristics: A case study in computational biology with differential evolution and spark. International Journal of High Performance Computing Applications (2016)
39. Teijeiro, D., Pardo, X.C., Penas, D.R., González, P., Banga, J.R., Doallo, R.: Evaluation of parallel differential evolution implementations on MapReduce and Spark. In: Lecture Notes in Computer Science, in press. Springer (2016)
40. Tirronen, V., Neri, F., Rossi, T.: Enhancing differential evolution frameworks by scale factor local search - part i. In: 2009 IEEE Congress on Evolutionary Computation, CEC 2009, pp. 94–101 (2009)
41. Črepinšek, M., Liu, S.H., Mernik, M.: Exploration and exploitation in evolutionary algorithms: A survey. ACM Computing Surveys **45**(3), 35:1–35:33 (2013)
42. Villaverde, A., R. Banga, J.: Reverse engineering and identification in systems biology: strategies, perspectives and challenges. Journal of the Royal Society Interface **11**(91), art. no. 20130,505 (2014)
43. Weber, M., Neri, F., Tirronen, V.: Shuffle or update parallel differential evolution for large-scale optimization. Soft Computing **15**(11), 2089–2107 (2011)
44. Weber, M., Neri, F., Tirronen, V.: A study on scale factor/crossover interaction in distributed differential evolution. Artificial Intelligence Review **39**(3), 195–224 (2013)
45. Weihmann, L., Martins, D., dos Santos Coelho, L.: Modified differential evolution approach for optimization of

planar parallel manipulators force capabilities. Expert Systems with Applications **39**(6), 6150 – 6156 (2012)
46. Zaharia, M., et al.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: The 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012 (2012)
47. Zhao, S.Z., Suganthan, P.N., Das, S.: Self-adaptive differential evolution with multi-trajectory search for large-scale optimization. Soft Computing **15**(11), 2175–2185 (2011)
48. Zhou, C.: Fast parallelization of differential evolution algorithm using MapReduce. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 1113–1114. ACM (2010)