A cloud based approach for parameter estimation problems in computational systems biology

Diego Teijeiro[1], Xoán C. Pardo[1], Patricia González[1], Julio R. Banga[2], and Ramón Doallo[1]

[1] Grupo de Arquitectura de Computadores. Universidade da Coruña. Spain ({diego.teijeiro,xoan.pardo,patricia.gonzalez,doallo}@udc.es)
[2] BioProcess Engineering Group. IIM-CSIC. Spain (julio@iim.csic.es)

Abstract. Metaheuristics are gaining increasing recognition in many research areas, computational systems biology among them. Parameter estimation, formulated as a global optimization problem, is a key step in the succesful modelling of complex biological systems. The calibration of biological models (expressed as non-linear dynamical systems) results in very challenging multi-modal problems. 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, are available to deal with large scale computations on commodity clusters and cloud resources. In this paper we propose a parallel implementation using Spark of an enhanced DE that includes a selected local search, exploiting the available distributed resources and drastically reducing the execution time. The performance of the proposal has been assessed using challenging parameter estimation problems from the domain of computational systems biology. The results obtained can be particularly useful for those interested in the potential of new cloud distributed frameworks for developing novel parallel metaheuristic methods.
Keywords: Parallel Metaheuristics, Differential Evolution, Local Search, Cloud Computing, Spark

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 [1–4], from medium and large-scale

up to the level of whole-cell models [5]. In this context, the problem of parameter estimation (model calibration) remains as a very challenging task [6, 7]. 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 [8–10]. In any case, the complexity of the underlying models requires the use of efficient solvers to achieve adequate results in reasonable computation times. Metaheuristics attain great consideration as an efficient way of solving hard global optimization problems. Differential Evolution (DE) [11] is one of the most popular methods, and it has been successfully used in many different areas [12]. 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. Therefore, several 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, and the earliest cloud programming models, like MapReduce [13], do not allow most scientific computations being efficiently run in the cloud. However more recent proposals like Spark [14] 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.

In this contribution our aim is to develop, using Spark, a parallel version of the DE algorithm following a cooperative strategy that will allow to efficiently exploit the available resources in a distributed system. In order to improve the performance of DE in parameter estimation problems in systems biology our proposal incorporates some enhancements to the basic DE, specifically a local search and a tabu list. The overall objective is to obtain a high performance implementation that achieves a good trade-off between exploration (diversification or global search) and exploitation (intensification or local search), which is at the core of modern metaheuristics [16].

The organization of the paper is as follows. Section 2 discusses related work. Section 3 presents a brief overview of the DE as well as the new features that have been included in the proposal to improve the search. The implementation with Spark 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

Many researches have tried to improve DE by proposing modifications to enhance the original algorithm. Interesting reviews can be found in [12, 17]. In several cases, the original DE algorithm was improved with additional algorithmic components exploiting certain aspects of a given class of problems. In [18] a modified DE approach using generation-varying control parameters is proposed

to improve the search performance in preventing of premature convergence to local minima. A hybrid algorithm using DE as an evolutionary framework and a crossover-based local search was proposed in [19, 20]. A DE with Scale Factor Local Search was introduced in [21, 22] for self-adaptive DE schemes. The use of a tabu list in the DE has also been applied in recent works [23–25].

On the other hand, several studies have considered parallel versions of DE, most of them focused on traditional parallel programming interfaces and infrastructures. A simple approach was proposed in [26], consisting of a master-slave architecture with several independent processes, where the communications were not established directly but through the filesystem. Another distributed DE was presented in [27] exploiting an island-model with asynchronous communication.

Several other works studied improvements to island-model schemes. In [28], a complete study about the impact on the performance of different communication topologies of the islands was presented. Several studies suggest that randomization of the control parameters can be a propitious mechanism for enhancing the DE performance [29]. Different randomization schemes have been proposed to develop self-adaptive DE frameworks and investigate the effect of changing control parameters in distributed DE [30, 31]. Two mechanisms to avoid the loss of diversity when the size of the population is small are described in [32]. 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 indicated that these techniques obtained a very significant performance when the dimensionality of the functions grew.

Research on cloud-oriented parallel metaheuristics based mainly on the use of MapReduce has also received increasing attention in recent years. Some proposals are specific on studying how to apply MapReduce to parallelize the DE algorithm to be used in the Cloud. In [33] 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 significantly reduces the benefits of the parallelization. In [34], 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 [35] a parallel implementation of DE based clustering using MapReduce is also proposed. This algorithm was implemented in three levels, each of which consists of different DE operations.

The use of Spark for the parallelization of the DE algorithm was explored in [36]. In that paper Spark-based implementations of two different parallel schemes of the DE algorithm, the master-slave and the island-based, are proposed and evaluated. Results showed that the island-based solution is by far the best suited to the distributed nature of Spark. In this paper we investigate the inclusion of further optimizations, namely a local search and a tabu list, to improve the convergence of the Spark-based island parallel DE and its scalability.

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 x of the P matrix do
        a⃗, b⃗, c⃗ ⇐ different random individuals from P matrix
        for k ⇐ 0 to D do
            if random point is less than CR then
                Ind⃗(k) ⇐ a⃗(k) + F(b⃗(k) - c⃗(k))
            end
        end
        if Evaluation(Ind⃗) is better than Evaluation(P(x)⃗) then
            Replace_Individual(P,Ind⃗)
        end
    end
until Stop conditions;
```

3 Differential Evolution

Differential Evolution (DE) [11] 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. For each iteration, new individuals are generated in the population matrix through operations performed among individuals of the matrix (mutation - F), with old solutions replaced (crossover - CR) 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 of metaheuristics pursues one or more of the following goals: increase the size of the problems that can be solved, speed-up the computations,

or attempt a more thorough exploration of the solution space. The solution explored in this work pursues the development of an efficient parallel variant of the serial DE, focussed on both the acceleration of the computation by performing separate evaluations in parallel, and the convergence improvement through the stimulation of the diversification in the search and the cooperation between the parallel threads.

In the literature, different parallel models can be found [37] aiming to improve both computational time and number of iterations for convergence. The *master-slave* and the *island-based* models are the most popular. In the *master-slave* model the behavior of the sequential DE is preserved by parallelizing the inner-loop of the algorithm, where a master processor distributes computation between the slave processors. The implementation of the DE master-slave model does not fit well with the distributed nature of frameworks like Spark [36]. 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 in subpopulations (*islands*) where the algorithm is executed isolated. Sparse individual exchanges are performed among islands to introduce diversity into the subpopulations, preventing search from getting stuck in local optima. Although the implementation of the *island-based* model in Spark drastically reduces the communications between different islands, the scalability is heavily restrained by the small size of the population matrix in the DE method. Thus, founded on the ideas outlined in [38], the *island-based* method can be used to perform a different DE in each island, using a different population matrix and different combinations of CR and F values to enhance diversity, while they cooperate through sparse migrations modifying the systemic properties of the individual searches.

3.2 Enhancing Differential Evolution with local search

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 by applying local changes, until a solution considered optimal 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 [39] is used. NL2SOL is a method for solving non-linear least-squares problems that has demonstrated to be particularly effective for parameter estimation problems [40, 38].

One drawback of the previous local search is that it tends to become stuck in suboptimal regions or on plateaus where many solutions are equally fit. As a means 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 describe 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*, so that the algorithm does not consider that possibility repeatedly. This technique improves the diversity among members of the population, and consequently contributes to the computational efficiency of the algorithm.

Next section describe in detail the implementation of the enhanced Spark-based parallel Differential Evolution proposed.

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), that persist and transform them 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 (`eSPDE`) 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 [36], modified to allow for heterogeneous islands and to incorporate to the islands the result of a local search 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. As it has been said, the proposed enhanced parallel DE (`eSPDE`) is based on the island-based parallel DE (`SiPDE`), in which 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. With this respect, `eSPDE`

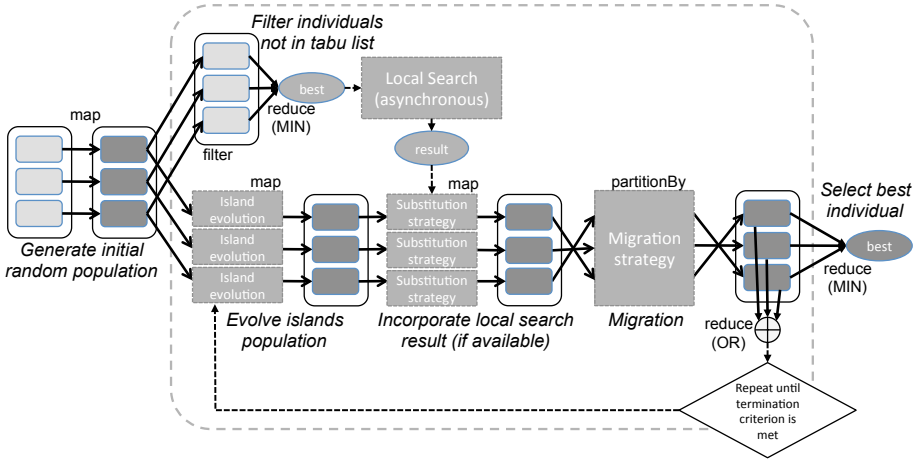


Fig. 1: Enhanced Spark-based parallel implementation of the DE algorithm (eSPDE).

enhances SiPDE by allowing islands to be heterogeneous, that is, having different combinations of CR and F values to enhance 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, after which 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, where a new search is initiated if no other is in progress. The candidate solution used 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 the input is selected by executing a Spark distributed filtering followed by a reduce action (a distributed MIN operation).
- Once the local search finishes, if it has improved the candidate solution, a substitution strategy is applied in between the evolution and migration steps to incorporate the improved solution into the population. For this work, an

strategy that replace 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 its result is included in every island.

5 Experimental Results

In order to evaluate the Spark implementation proposed in this paper (`eSPDE`) three challenging parameter 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[41, 42]:

- *Circadian* model: parameter estimation in a dynamic model of the circadian clock in the plant *Arabidopsis thaliana*, as presented in [43]. 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 [44] 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 nonlinear pathway with 8 differential equations and 36 parameters, and data sets from 16 experiments, as presented in [41].

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 standard instances (4 cores, 7GB) for head and worker nodes. 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 (`eSPDE`) can be used in two different manners: (i) dividing the population among islands and using

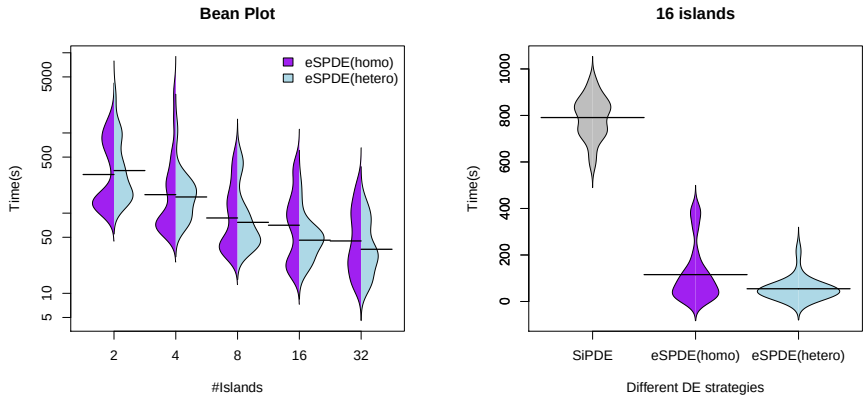
Table 1: Performance evaluation of different DE strategies for the Circadian benchmark. Parameters: D=13, NP=256, VTR=1e-5.

method	**#np**	**#evals**	**#mig.**	**time(s)**		**speedup**
				mean±std	**median**	
`seqDE`	1	6437670	-	40883.39±3712.56	40916.76	-
`SiPDE`	2	5980416	117	19275.65±1281.63	19015.77	2.12
	4	5729536	112	9305.30±1038.59	9071.51	4.39
	8	3904256	74	3319.33±296.88	3256.62	12.32
	16	1835776	36	790.97±90.50	815.51	51.69
	32	1577216	30	348.36±43.47	355.05	117.36
`eSPDE` (homo)	2	179456	3.5	472.41±441.29	143.80	86.54
	4	230656	4.5	388.31±736.39	104.44	105.29
	8	171776	3.3	134.01±140.78	75.26	305.07
	16	225536	4.4	115.48±119.04	77.82	354.04
	32	235776	4.6	67.60±63.63	40.56	604.76
`eSPDE` (hetero)	2	161536	3.1	524.28±631.98	311.08	77.98
	4	120576	2.3	201.81±217.42	165.09	199.62
	8	128256	2.5	115.51±135.43	45.76	353.92
	16	107776	2.1	54.81±43.34	48.07	745.90
	32	161536	3.2	46.85±36.01	31.55	872.69

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 in Scala (`seqDE`) and the implementation of an island-based parallel DE in Spark described in [36] (`SiPDE`).

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), whose selection may have a great impact in the algorithm performance. Since the objective of this work is not to evaluate the impact of these parameters, only results for one configuration are reported here. Previous tests have been done to select those parameters that lead to reasonable computation times. For all the experiments we used MSt=DE/rand/2. For testing the homogeneous configuration of `eSPDE` F=0.9 and CR=0.8 were used, while for the heterogeneous configuration different combination of CR=$\{0.7,0.8,0.9\}$ and F=$\{0.8,0.9\}$ values were randomly selected for each island. 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



(a) Bean plots for eSPDE

(b) Bean plots for 16 islands

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

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 1. This table displays, for each experiment, the number of cores (*#np*) used, the mean number of evaluations needed (*#evals*), the mean number of migrations in the island-model algorithms (*#mig.*), the mean and the median of the execution times (*time(s)*), and the speedup achieved versus the `seqDE`. Results show that the parallelization improves the execution time required for convergence by performing the evaluations in parallel. `SiPDE`, an island-based implementation similar to the one proposed in this work but without the inclusion of the tabu list and the local solver, achieves already a good speedup versus the sequential algorithm. However, the local search included in the `eSPDE` implementation significantly reduces the execution time required for convergence, by means of a decrease in the number of evaluations. Note the drastically reduction in the number of migrations performed by the algorithm 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(a) shows the bean plots to compare the distribution of the homogeneous versus the heterogeneous configuration using the `eSPDE` implementation. Note that the logarithmic scale has been used in the y axis and the mean of each distribution are also shown in each bean. It can be noted that for two islands the performance of the homoge-

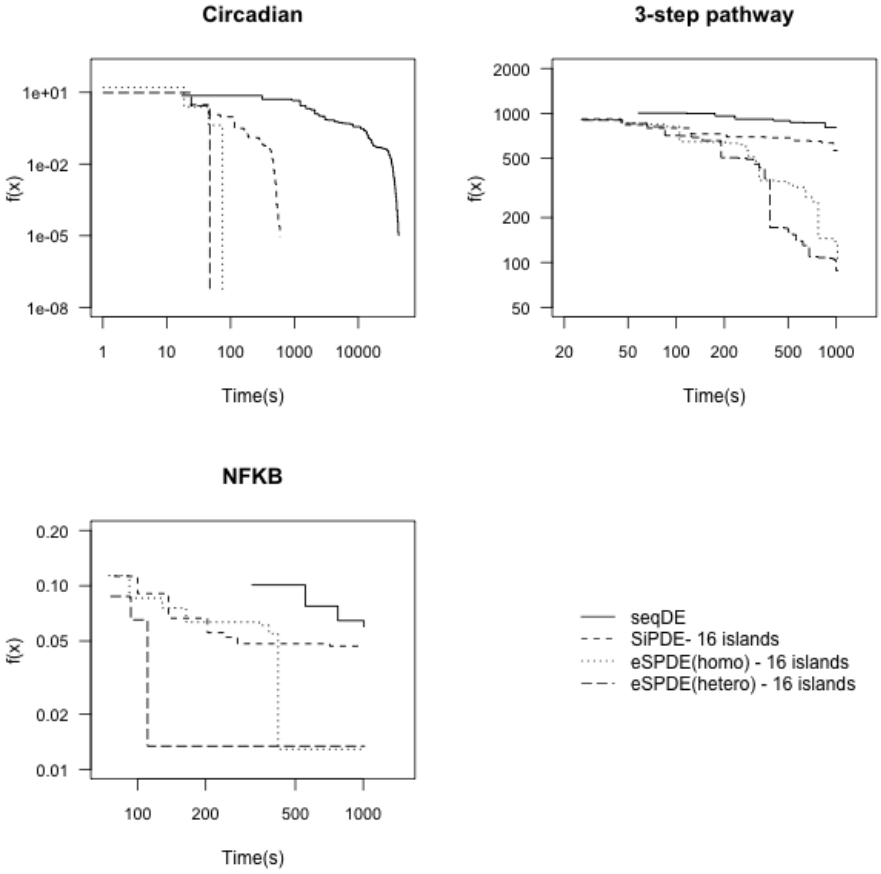
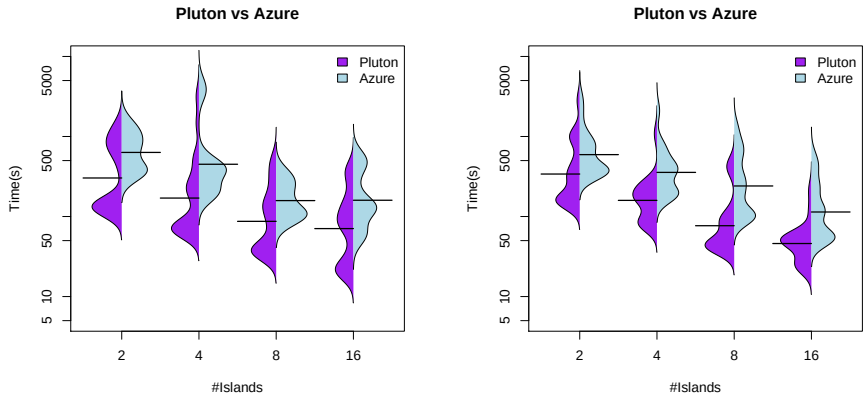


Fig. 3: Convergence curves: Circadian using as stopping criterium a VTR=1e-5, 3-step pathway and NFKB using as stopping criterium a predefined effort of $T_{max} = 1000s$.

neous configuration was slightly better because the heterogeneous configuration presented more outliers. However, when the number of islands increases, the heterogeneous configuration drastically reduces the dispersion in the results and achieves better performance. Figure 2(b) allows for a more comprehensible comparison showing, without the logarithmic scale in the y axis, the distribution of the results for 16 islands and including also the results for `SiPDE`.

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 curves depicted are those that fall in the median values of the results distribution. It can be seen that, as expected, the local solver improves the convergence rate in all



(a) eSPDE(homo) (b) eSPDE(hetero)

Fig. 4: Beanplots comparing execution times for the Circadian benchmark in the local cluster Pluton and the Azure public cloud.

the benchmarks. Also the heterogeneous configuration exhibits a slightly better performance than the homogeneous one.

Finally, in order to evaluate the performance of the proposal in a public cloud, some experiments were conducted in the Microsoft Azure public cloud. Beanplots comparing the results obtained in our local cluster Pluton and in Azure are shown in Figure 4. As it can be seen, the proposal achieves similar results as the ones obtained in the local cluster in terms of convergence and scalability, however, the overheads introduced in Azure due to virtualization and use of non-dedicated resources in a multi-tenant platform are not negligible, being the execution times obtained in Azure between 1.5x and 5x the times obtained in Pluton. Further studies to determine a more accurate interpretation of this overhead are left for future work.

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 of the 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 and that using cloud resources shows similar behavior in terms of convergence and scalability as using resources from a local cluster but at the expense of a not negligible overhead.

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

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 from the Galician Government under the Consolidation Program of Competitive Research Units (Network Ref. R2014/041 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. Smallbone, K., Mendes, P.: Large-scale metabolic models: From reconstruction to differential equations. Industrial Biotechnology **9**(4) (2013) 179–184
2. Almquist, J., Cvijovic, M., Hatzimanikatis, V., Nielsen, J., Jirstrand, M.: Kinetic models in industrial biotechnology - Improving cell factory performance. Metabolic Engineering (2014) 1–22
3. Link, H., Christodoulou, D., Sauer, U.: Advancing metabolic models with kinetic information. Current Opinion in Biotechnology **29** (March 2014) 8–14
4. Le Novère, N.: Quantitative and logic modelling of molecular and gene networks. Nature Reviews Genetics **16**(3) (2015) 146–158
5. 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) (2012) 389–401
6. Gábor, A., Banga, J.R.: Robust and efficient parameter estimation in dynamic models of biological systems. BMC systems biology **9**(1) (2015) 74
7. Cedersund, G., Samuelsson, O., Ball, G., Tegnér, J., Gomez-Cabrero, D.: Optimization in Biology Parameter Estimation and the Associated Optimization Problem. In: Uncertainty in Biology: A Computational Modeling Approach. Springer International Publishing, Cham (2016) 177–197
8. Banga, J., Balsa-Canto, E.: Parameter estimation and optimal experimental design. Essays Biochem **45** (2008) 195–210

9. Ashyraliyev, M., Fomekong-Nanfack, Y., Kaandorp, J.A., Blom, J.G.: Systems biology: parameter estimation for biochemical models. Febs Journal **276**(4) (2009) 886–902
10. 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) (2012) 185–202
11. Storn, R., Price, K.: Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization **11**(4) (1997) 341–359
12. Das, S., Suganthan, P.N.: Differential evolution: A survey of the state-of-the-art. Evolutionary Computation, IEEE Transactions on **15**(1) (2011) 4–31
13. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: The 6th USENIX Symposium on Operating Systems Design and Implementation. (2004)
14. 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)
15. Ewen, S., Tzoumas, K., Kaufmann, M., Markl, V.: Spinning fast iterative data flows. CoRR **abs/1208.0088** (2012)
16. Črepinšek, M., Liu, S.H., Mernik, M.: Exploration and exploitation in evolutionary algorithms: A survey. ACM Computing Surveys **45**(3) (July 2013) 35:1–35:33
17. Neri, F., Tirronen, V.: Recent advances in differential evolution: A survey and experimental analysis. Artificial Intelligence Review **33**(1-2) (2010) 61–106
18. 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) (2012) 6150 – 6156
19. Noman, N., Iba, H.: Enhancing differential evolution performance with local search for high dimensional function optimization. In: GECCO 2005 - Genetic and Evolutionary Computation Conference. (2005) 967–974
20. Noman, N., Iba, H.: Accelerating differential evolution using an adaptive local search. IEEE Transactions on Evolutionary Computation **12**(1) (2008) 107–125
21. 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. (2009) 94–101
22. 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. (2009) 118–125
23. 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) (2007) 3410–3421
24. 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. (2012) 639–644
25. Schneider, E., Krohling, R.A.: Differential evolution and tabu search to find multiple solutions of multimodal optimization problems. In: Soft Computing in Industrial Applications. Volume 223 of Advances in Intelligent Systems and Computing. Springer International Publishing (2014) 61–69
26. Ntipteni, M.S., Valakos, I.M., Nikolos, I.K.: An asynchronous parallel differential evolution algorithms. In: Proceedings of the ERCOFTAC conference on design optimisation: methods and application. (2006)

27. Apolloni, J., García-Nieto, J., Alba, E., Leguizamón, G.: Empirical evaluation of distributed differential evolution on standard benchmarks. Applied Mathematics and Computation **236** (2014) 351–366
28. Ruciński, M., Izzo, D., Biscani, F.: On the impact of the migration topology on the island model. Parallel Computing **36**(10-11) (2010) 555–571
29. 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) (2006) 646–657
30. Zhao, S.Z., Suganthan, P.N., Das, S.: Self-adaptive differential evolution with multi-trajectory search for large-scale optimization. Soft Computing **15**(11) (2011) 2175–2185
31. Weber, M., Neri, F., Tirronen, V.: A study on scale factor/crossover interaction in distributed differential evolution. Artificial Intelligence Review **39**(3) (2013) 195–224
32. Weber, M., Neri, F., Tirronen, V.: Shuffle or update parallel differential evolution for large-scale optimization. Soft Computing **15**(11) (2011) 2089–2107
33. Zhou, C.: Fast parallelization of differential evolution algorithm using MapReduce. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, ACM (2010) 1113–1114
34. Tagawa, K., Ishimizu, T.: Concurrent differential evolution based on MapReduce. International Journal of Computers **4**(4) (2010) 161–168
35. 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), IEEE (2014) 337–341
36. Teijeiro, D., Pardo, X.C., González, P., Banga, J.R., Doallo, R.: Implementing Parallel Differential Evolution on Spark. In: Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Proceedings, Part II. (2016) 75–90
37. Alba, E., Luque, G., Nesmachnow, S.: Parallel metaheuristics: recent advances and new trends. International Transactions in Operational Research **20**(1) (2013) 1–48
38. Penas, D., Banga, J., González, P., Doallo, R.: Enhanced parallel differential evolution algorithm for problems in computational systems biology. Applied Soft Computing **33** (2015) 86 – 99
39. 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) (September 1981) 369–383
40. Egea, J., Rodríguez-Fernández, M., R. Banga, J., Martí, R.: Scatter search for chemical and bio-process optimization. Journal of Global Optimization **37**(3) (2007) 481–503
41. Moles, C., Mendes, P., R. Banga, J.: Parameter estimation in biochemical pathways: A comparison of global optimization methods. Genome Research **13**(11) (2003) 2467–2474
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) (2014) art. no. 20130505
43. 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) (2005) 383–393
44. Lipniacki, T., Paszek, P., Brasier, A., Luxon, B., Kimmel, M.: Mathematical model of nf-κb regulatory module. Journal of Theoretical Biology **228**(2) (2004) 195–215