Enhanced parallel Differential Evolution algorithm for problems in computational systems biology

Article *in* Applied Soft Computing · April 2015

DOI: 10.1016/j.asoc.2015.04.025

4 authors, including:

David R. Penas
Spanish National Research Council
10 PUBLICATIONS **17** CITATIONS
SEE PROFILE

Julio R. Banga
Spanish National Research Council
281 PUBLICATIONS **5,395** CITATIONS
SEE PROFILE

Patricia González
University of A Coruña
96 PUBLICATIONS **291** CITATIONS
SEE PROFILE

Applied Soft Computing 33 (2015) 86–99

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Enhanced parallel Differential Evolution algorithm for problems in computational systems biology

D.R. Penas [a], J.R. Banga [a,*], P. González [b,**], R. Doallo [b]

[a] BioProcess Engineering Group, IIM-CSIC, Spanish National Research Council, C/ Eduardo Cabello, 6, E-36208 Vigo, Spain
[b] Computer Architecture Group, Department of Electronics and Systems, University of A Coruña, Campus de Elviña s/n, 15071 A Coruña, Spain

ARTICLE INFO

Article history:
Received 16 October 2014
Received in revised form 13 April 2015
Accepted 14 April 2015
Available online 21 April 2015

Keywords:
Computational systems biology
Parallel metaheuristics
Distributed differential evolution

ABSTRACT

Many key problems in computational systems biology and bioinformatics can be formulated and solved using a global optimization framework. The complexity of the underlying mathematical models require the use of efficient solvers in order to obtain satisfactory results in reasonable computation times. Metaheuristics are gaining recognition in this context, with Differential Evolution (DE) as one of the most popular methods. However, for most realistic applications, like those considering parameter estimation in dynamic models, DE still requires excessive computation times.

Here we consider this latter class of problems and present several enhancements to DE based on the introduction of additional algorithmic steps and the exploitation of parallelism. In particular, we propose an asynchronous parallel implementation of DE which has been extended with improved heuristics to exploit the specific structure of parameter estimation problems in computational systems biology. The proposed method is evaluated with different types of benchmarks problems: (i) black-box global optimization problems and (ii) calibration of non-linear dynamic models of biological systems, obtaining excellent results both in terms of quality of the solution and regarding speedup and scalability.

1. Introduction

Global optimization methods are playing an increasingly important role in computational biology [1], bioinformatics [2] and systems biology [3]. In the case of the field of systems biology, the aim is to understand complex biological systems by combining experimental data with mathematical modelling and computational methods.

Building these mathematical models is an iterative process which starts with the definition of the purpose of the model and the selection of a model framework. Then, a mathematical structure is proposed with a set of non-measurable parameters. After that, these parameters are estimated in order to obtain quantitative predictions. Finally, the model is (in)validated with new experiments, obtaining feedback which can be subsequently used in a refinement process.

The parameter estimation step is key in this iterative model building process and can be formulated as a mathematical optimization problem subject to the dynamic constraints which describe the time-dependent behaviour of the system. Most biological models are highly non-linear dynamical systems, resulting in challenging multi-modal problems which are very difficult to solve, as described in [4].

Global optimization methods are robust alternatives to solve these complex optimization problems. They can be roughly classified in two classes, deterministic methods and stochastic (including heuristic) methods. In the first type, the solution retrieved will be the global optimum, though the computational effort to ensure global optimality might be extremely large, making them impractical. Stochastic and other heuristic methods do not guarantee convergence to the global optimum, but they usually provide near-global solutions in reasonable computation times.

In the case of stochastic methods, many research efforts have focused on developing metaheuristic (i.e. guided heuristic) methods which are able to locate the vicinity of the global solution in reasonable computation. In order to reduce further the computational cost of these methods, a number of researchers have studied parallel strategies for metaheuristics [5,6]. In the area of computational systems biology, parallel methods have already shown promising results [7–9].

Differential Evolution (DE) [10] is one of the most popular heuristics for global optimization, and it has been successfully used in many different areas [11–13]. In particular, DE remains

* Corresponding author. Tel.: +34 986 214473.
** Corresponding author. Tel.: +34 981 167000.
E-mail addresses: julio@iim.csic.es (J.R. Banga), pglez@udc.es (P. González).

http://dx.doi.org/10.1016/j.asoc.2015.04.025
1568-4946/

as a widely used method for parametric identification of complex models [14–16]. 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. Thus, in order to improve the runtime of the classical DE algorithm, two main strategies have been preliminary explored by us in [17]. First, including a selected local search and other algorithmic improvements in order to enhance the classical DE through intensification, drastically reducing the number of evaluations required. Second, exploiting parallelism at different levels, so as to reduce the computational time needed. In this paper, we extend significantly our previous work to include: (i) further description of the asynchronous parallel strategy and its improvement in the global search through the enhancement in diversification due to a cooperative scheme, (ii) an additional extensive experimental analysis both in black-box global optimization problems and in calibration of non-linear dynamic models of biological systems. Regarding experimental results, both *vertical* and *horizontal* views [18] are analyzed in this paper. A vertical approach assesses the performance of a fix number of evaluations, i.e., a predefined effort; while an horizontal view assesses the performance by measuring the time needed to reach a given target value. Our main goal is to improve the horizontal approach, however, the vertical approach will also benefit from the proposed solution, which is interesting for many real world applications where the total number of evaluations is limited.

The organization of this paper is as follows. Section 2 covers the related work, while Section 3 presents a brief overview of the DE algorithm. Section 4 describes the asynchronous strategy proposed to parallelize an island-based DE algorithm. Section 5 describes the new heuristics for parameter estimation that have been added to improve local search. The performance of the proposed method is evaluated in Section 6, demonstrating its good efficiency and scalability. Finally, Section 7 summarizes the main conclusions of our study.

2. Related work

Many researches have tried to improve DE by proposing modifications to the original algorithm. Interesting reviews can be found in [19] and more recently in [13]. In several cases, the original DE algorithm was improved with additional algorithmic components exploiting certain aspects of a given class of problems. In [20] 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 [21,22]. A DE with Scale Factor Local Search was introduced in [23] and extended in [24] for self-adaptive DE schemes. The use of a tabu list in the DE has also been applied in recent works [25–27].

Several studies have considered parallel versions of the algorithm. A distributed adaptive DE version was proposed in [28]. This algorithm was based on the island-mode paradigm with a random communication topology for individuals exchange. Another parallel approach was proposed in [29], based on the distribution of the population data among different processors (slaves), which communicate through data migrations, all of them managed by a central processor (master). The latter was also responsible of checking the stopping criteria. The algorithm was implemented in PVM (Parallel Virtual Machine) [30] with presumably synchronous communication, resulting in low speed-up results.

A simple approach to asynchronous parallelization was proposed in [31], consisting of a master-slave architecture with several independent processes, where the communications were not established directly but through the filesystem. The master process was in charge of selecting the best individuals. When the stopping condition was satisfied, the slaves were stopped. Another asynchronous proposal based on a master–slave approach is the parallel metaheuristic based on DE and simulated annealing proposed in [32]. The master asynchronously assigns different tasks to the slaves, thus, allowing for simultaneous evaluation of several trial solutions. The proposal is implemented in MATLAB, using PVM for the communications between master and slaves.

A parallel DE with asynchronous communications and an island-model scheme is presented in [33]. Its termination criteria was controlled by a master node and was implemented with POSIX (Portable Operating System Interface) threads. A heterogeneous local configuration was used, where each parallel processor had a different mutation strategy (MSt). The results indicated an improvement in the reliability and the performance of the algorithm with a configuration of five islands, compared to the time spent by five sequential versions for the same optimization problems. Another asynchronous distributed DE was presented in [34]. In this case, the algorithm was also exploiting an island-model with asynchronous communication. The topology was an unidirectional ring, where the individuals exchanged were selected randomly.

Several other works studied improvements to island-model schemes. In [35], a complete study about the impact on the performance of different communication topologies of the islands was presented. These authors used a synchronous parallel DE on a set of standard benchmarks with different topologies, concluding that ring topology was the best option. In [36], subpopulations were grouped into two families. The first family uses a ring topology and a best-random like migration strategy to evolve its subpopulations. In the second one each subpopulation independently evolves with a population size reduction strategy. The solutions generated by the second family are moved into the first family.

Several studies suggest that randomization of the control parameters can be a propitious mechanism for enhancing the DE performance [37]. Different randomization schemes have been proposed to develop self-adaptive DE frameworks and investigate the effect of changing control parameters in distributed DE [38–41]. Two mechanisms to avoid the loss of diversity when the size of the population is small are described in [42]. 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 subpopulations. The results indicated that these techniques obtained a very significant performance when the dimensionality of the functions grew. An improved which entails four different scale factors updating schemes, associated to the binomial crossover in a distributed DE structure, is presented in [41]. With the exception of one scheme in which equally spaced scale factors are considered, in all the others the scale factors are randomly initialized for each subpopulation. Although proper choice of a scale factor scheme appears to be dependent on the distributed structure, each of these simple schemes proposed has proven to significantly improve upon the single-scale factor distributed DE algorithms.

Other parallelization strategies have appeared with the emergence of new hardware and software technologies. This is the case of [43], where a DE improved through local Pattern Search method was parallelized through CPU-GPU heterogeneous computing, using a cellular model scheme. Also, a parallel DE based on GPUs is explored in [44], which employs self-adapting control parameters and generalized opposition-based learning (GOBL) to improve the quality of candidate solutions. In [45] a multiagent framework was proposed to create a distributed cooperative structure based on agents. This scheme was implemented in Java, defining a communication API (Application Programming Interface) to send information to the different agents of the environment.

In this contribution, our aim was (i) to improve the DE algorithm incorporating several algorithmic steps which exploit the structure of parameter estimation problems, and (ii) develop a parallel version following a cooperative asynchronous strategy. The overall objective was 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 [46].

3. Differential Evolution

Differential Evolution (DE) is an iterative mutation algorithm where vector differences are used to create new candidate solutions. Algorithm 1 shows a simple pseudocode for the classical sequential DE (which we will denominate `seqDE` here).

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 in P do
        a, b, c ← different random individuals from P matrix
        for k ← 1 to D do
            if random point is less than CR then
                Ind(k) ← a(k) + F(b(k) - c(k));
            end
        end
        if Evaluation(Ind) is better than Evaluation(P(x)) then
            P(x) = Ind
        end
    end
until Stop conditions;
```

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. For each iteration, new individuals are generated in the population matrix through mutation operations performed among individuals of the matrix. These mutation operations depend on the mutation factor (F), which is used in the creation of new solutions in different ways depending of the selected mutation scheme. There are different mutation strategies (MSt) to generate new individuals, this work uses some of them:

- `DE/best/1`:

$$Ind_k \leftarrow bestP_k + F \cdot (b_k - c_k) \quad (1)$$

- `DE/best/2`:

$$Ind_k \leftarrow bestP_k + F \cdot (b_k - c_k) + F \cdot (d_k - e_k) \quad (2)$$

- `DE/rand/1`:

$$Ind_k \leftarrow a_k + F \cdot (b_k - c_k) \quad (3)$$

- `DE/rand/2`:

$$Ind_k \leftarrow a_k + F \cdot (b_k - c_k) + F \cdot (d_k - e_k) \quad (4)$$

where $\vec{a}, \vec{b}, \vec{c}, \vec{d}, \vec{e} \in P$ are solution vectors randomly selected, $\vec{bestP}_k \in P$ is the current best solution of the population, and $\vec{Ind} = (Ind_1, \ldots, Ind_D)$ is the new candidate solution created in the mutation process.

These mutation operations are applied in specific positions k of the old solution vector of the matrix. These positions are determined through the crossover constant (CR): if random generated value is less than CR, the mutation strategy is applied in the position k of the current vector. Thus, candidate solutions have one part of the old solutions that are intended to replace, and on the other hand, they have values obtained from the mutation process.

Finally, only when the fitness value of the new candidate solution is better than the current one, the new individual is included in the population matrix.

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.

4. Improving global search through an asynchronous 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. However, to achieve an efficient parallelization of metaheuristics is usually a complex task, since the search of new solutions depends on previous iterations of the algorithm, which not only complicates the parallelization itself but also limits the achievable speedup. Different strategies can be used to address this problem: attempting to find parallelism in the sequential algorithms, preserving its behaviour; finding parallel variants of the sequential algorithms, slightly varying their behaviour to obtain a more easily parallelizable algorithm; or developing fully decoupled algorithms, where each process executes its part without communication with other processes, at the expense of reducing its effectiveness.

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. Thus, when evaluating the performance of the proposal, both vertical and horizontal approaches will benefit.

The parallel algorithm proposed is based on the island model approach [47]. The population matrix is divided in subpopulations (*islands*) where the algorithm is executed isolated. Phases such as selection, recombination and mutation are performed only within each island, which implies absence of collaboration among

processes. Sparse individual exchanges are performed among islands to introduce diversity into the subpopulations, preventing search from getting stuck in local optima. After m iterations, a migration phase links the different populations: selected individuals from each island are communicated to another island. Both the migration operation and the checking of the termination criteria imply an exchange of communications among processes.

The simplest implementation of the parallel island DE is a synchronous algorithm (`synPDE`). The drawback of the synchronous algorithm is that processors are idle during a significant amount of time, while they are waiting for each other during the migration steps.

Replacing synchronous communications with asynchronous ones, each process will send the information to a memory buffer associated with the remote process, enabling the reception of the message later on (whenever that process is ready to receive it), thus, avoiding idle periods. However, in such asynchronous version, when a migration is planned, each process would need to wait for the reception of its required new data. This could stall processes and cause idle periods.

The algorithm proposed in this work avoids this by implementing a variation of the classical parallel island DE. The pseudocode for the proposed solution (`asynPDE`) is shown in Algorithm 2. Each process receives an island population. For each iteration of the main loop, mutation and crossover operations are performed within each island, in the same way as in the serial implementation. Every m iterations, a migration phase is performed to link the evaluations in different islands. Whenever a process reaches the migration phase, it sends a set of individuals to the selected remote process using an asynchronous communication (`ISend()` function in Algorithm 2). Then, the process in the migration phase checks if the message with the new individuals of a remote process has already arrived to its memory buffer (`IRecv()` function in Algorithm 2). However, if the new solutions have not arrived yet, the process proceed with the next evaluation. After each evaluation the process searches for the reception of data missed in previous migrations (`Test()` function in Algorithm 2), however avoiding stalls if the messages have not arrived yet. To deal with the migration data received asynchronously, a dynamic data list is created. In each migration phase a new node is attached to the list, to be filled with the expected data. Once the data of a node is used, the node is removed from the list.

In addition to the migration step, the checking of the stopping criteria may also involve communications between processes. Stopping criteria are needed to terminate the execution of the algorithms. They can be as simple as using a maximum number of evaluations, which do not imply exchange of communications. However, other criteria, that allow to react adaptively to the state of the optimization progress, need communications between processes. Asynchronous MPI communications are also used in the proposed algorithm for those communications, so that processes may continue running independently. Each parallel process opens a buffer where it expects to receive a termination message. This buffer is checked every iteration of the algorithm. Thus, the control of the stopping criteria of the global search is distributed among all the processes: when a stopping condition is fulfilled in a process, this condition is communicated to the rest of processes, then all of them can stop the algorithm almost at the same time.

Algorithm 2. Asynchronous island-based parallel DE – `asynPDE`

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

```
Plocal ← scatter population P into N processors
iter = 0;
repeat
    for each element in Plocal do
        Crossover, mutation and evaluation operations
    end
    ! migration phase
    if iter%m==0 then
        migrationSet ← selected individuals from Plocal
        ! asynchronous send of migrationSet to remote destination
        ISend(migrationSet, remoteDestination);
        ! asynchronous reception of migrationSet in the receptionSet
        IRecv(receptionSet, remoteOrigin); ! non-blocking operation that allows for execution progress if no message has arrived yet
    end
    while pending migration do
        ! check for pending messages of previous migrations
        Test(receptionSet, isComplete); ! non-blocking operation that allows for execution progress
        if isComplete then
            ! Insert received individuals into Plocal subpopulation
            Plocal (selected individuals) ← receptionSet
        else
            break;
        end
    end
    iter++;
until Stopping condition;
Gather all subpopulation into matrix P
```

Finally, note that the new parallel algorithm does not implement straightforwardly the serial one. As demonstrated in Section 6, it performs always better in terms of execution time and scalability even if, often, it requires higher number of evaluations when assessing its performance from an horizontal view.

5. Improving local search in Differential Evolution algorithms

In some real applications, such as parameter estimation in dynamic models, the performance of the classical sequential DE is

not acceptable due to the large number of objective function evaluations needed. As a result, typical runtimes for realistic problems are in the range from hours to days. In order to improve the computational effort required by the DE algorithm running in each of the processors in the parallel version, three enhancements that exploit the special structure of these parameter estimation problems have been included.

5.1. Logarithmic space

The classical DE algorithm begins by generating an initial set P of individuals. Typically this generation is performed in a uniform space, where the range is divided into n sub-ranges of equal size and values are randomly chosen from selected sub-ranges. However, in other metaheuristics it has been found that, when variables may have values in a huge range, this uniform distribution for selecting diverse solutions will not generate many trial points with good value [48]. In contrast, a logarithmic distribution will initialize the algorithm with high quality members in the initial population, ensuring a faster convergence. Thus, the search is proposed to be performed in a logarithmic space, which results in a more suitable exploration of the space of parameters when they are positive and potentially span through several orders of magnitude.

5.2. Local solver

A local method is introduced to achieve a fast local convergence, therefore, reducing the number of objective function evaluations required when an horizontal view is assessed. 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. NL2SOL [49], a method for solving non-linear least-squares problems, has demonstrated to be particularly effective for parameter estimation problems [48]. In this work, NL2SOL is called every L iterations of the DE algorithm.

5.3. Tabu list

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 search* 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.

Algorithm 3 shows the pseudocode for the local search and tabu list included at the end of each external iteration of the DE algorithm. The local solver condition is met each L external iterations of the DE algorithm, and the evaluations performed during the local search stage count in the total number of evaluations of the DE algorithm.

Algorithm 3. Local Search and Tabu List

```
TabuList ← set of visited points in the local solver ;
! local solver condition
if iter%L==0 then
    Sort Plocal population;
    Point = Plocal(0);
    for i ← 0 to number of individuals of Plocal do
        ! Calculate all distances among the point Plocal(i) and all points of the Tabu List
        distanceSet = distance_Euclidean(Plocal(i) , TabuList);
        if ∀ d ∈ distanceSet > min_distance then
            Point = Plocal(i);
            break;
        end
    end
    Insert Point in the TabuList;
    newPoint = Run_Local_Search(Point);
    if newPoint is better than Point then
        Replace the worst point in Plocal with newPoint;
    end
end
```

Although all these three enhancements have significantly contributed to accelerate the solution of our real systems biology applications, the use of a local solver (effectively creating a hybrid method) has proved to be particularly useful. It is worth mentioning that hybrid methods have a long tradition in numerical methods in general, and numerical optimization in particular (e.g. Powell's dogleg method [50] is a well known classical example). In evolutionary computation, memetic algorithms [51] use a hybrid approach to combine global with local search methods. Hyper-heuristics make use of an even higher level of generality, with the objective of choosing the right heuristic for a given problem [52]. Here we have chosen a parsimonious hybrid design, combining a classical DE scheme with an specialized local search.

Fig. 1 graphically illustrates the global proposal `asynPDE_IH`: the asynchronous parallel implementation of a DE extended with improved heuristics. Note that different processes are executing a DE in different stages, and cooperation between them is performed in an asynchronous fashion avoiding stalls if any of the processes is involved in a time consuming phase, such as the execution of the local solver (see process ID 3 in the figure), while other processes (processes ID 2 and ID 3 in the figure) are in the migration phase. When a process is not able to attend to a migration reception, the message will be stored in the process as a pending migration, avoiding the blocking of the sender process (see process ID 1 in the figure, attending pending migrations).

6. Experimental results

This section assesses the impact of the described optimization techniques in the DE algorithm. In order to evaluate the efficiency

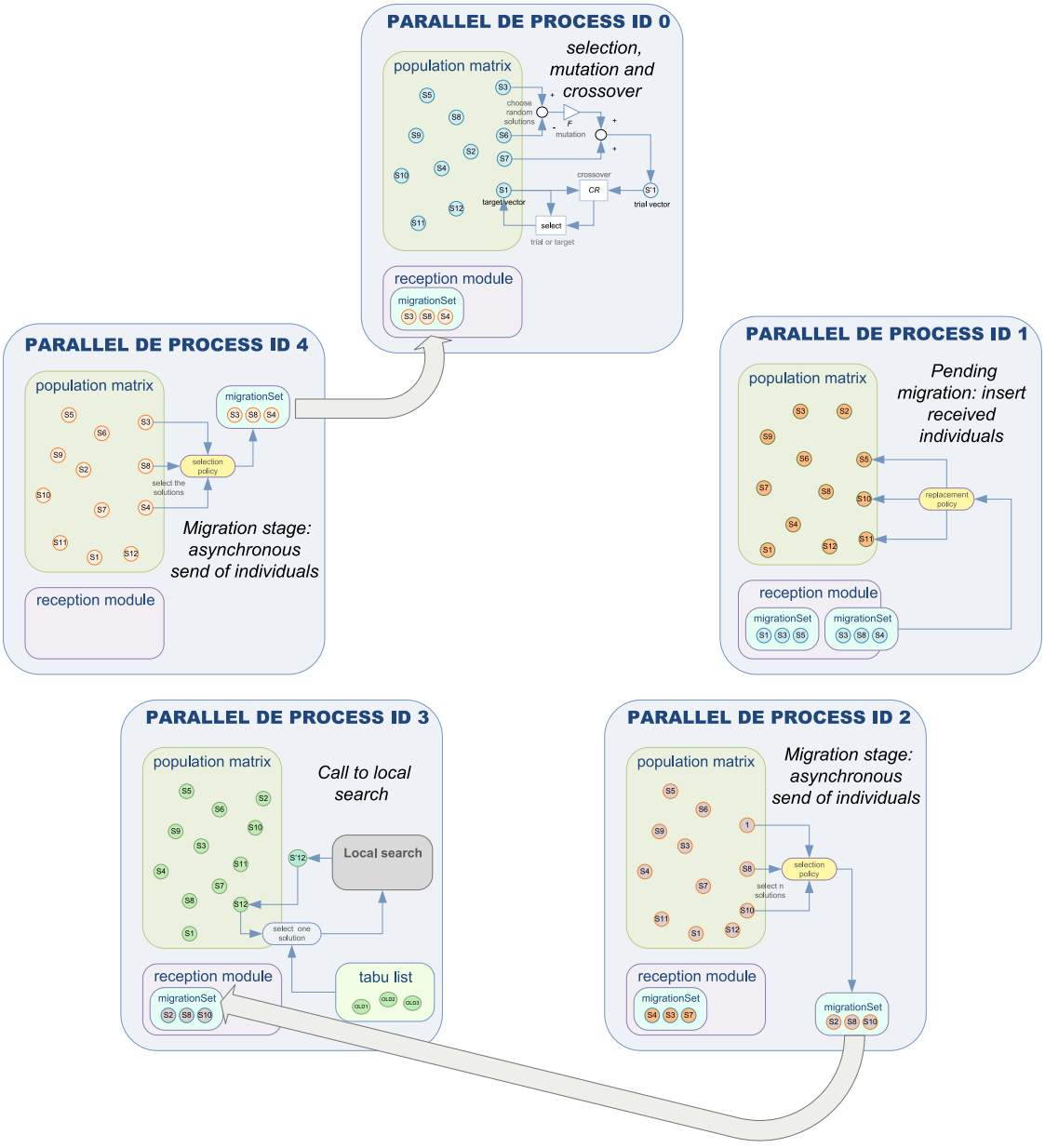

Fig. 1. Asynchronous parallel DE with improved heuristics (`asynPDE_IH`).

of the proposed cooperative asynchronous algorithm, different experiments have been carried out. Its behaviour, in terms of convergence and total execution time, was compared with alternative versions of DE. To simplify the understanding of this work, we use the following nomenclature in this section:

- `seqDE`: sequential classic version of DE (Algorithm 1)
- `asynPDE`: proposed asynchronous parallel version of the seqDE
- `synPDE`: synchronous parallel version of the seqDE (used for comparative purposes)
- `seqDE_IH`, `asynPDE_IH` and `synPDE_IH`: versions of the above algorithms with improved heuristics enabled (local search, tabu list and logarithmic search).

The experiments performed are presented in two subsections. The first one analyzes the performance of the asynchronous parallel cooperative strategy proposed considering an algebraic black-box optimization test-bed. The second one evaluates the overall improvement in a set of computational systems biology problems by combining diversification achieved by the parallel cooperative strategy with the intensification proposed for the local search.

In order to enable the reproducibility of these experiments and the comparison with other parallel approaches [53], sufficient information to allow the replication of the different experiments, such as all the configuration parameters used in each test, are provided in next subsections. Also, because of the stochastic properties inherent to these algorithms, several independent runs have been made for each experiment. The number of independent runs and statistical data corresponding to the obtained results are reported as well.

6.1. Performance evaluation of the asynchronous parallel strategy

The quality of the solution for the proposed parallelization has to be evaluated, since the proposal implements not only a variant of the sequential DE, the island-model, but also a modification

Table 1
Subset of BBOB-2009 benchmark functions. Parameters: dimension (D), population size (NP), crossover constant (*CR*), mutation factor (*F*), mutation strategy (MSt), value-to-reach/*ftarget* (VTR).

Number	Function	D	NP	*CR*	*F*	MSt	VTR
f8	Rosenbrock Function, original	12	150D	.8	.9	DE/rand/2	149.15
f15	Rastrigin Function	5	150D	.8	.9	DE/rand/2	1000
f17	Schaffers F7 Function	6	150D	.8	.9	DE/rand/2	−16.94
f19	Composite Griewank-Rosenbrock Function F8F2	4	3000D	.9	.9	DE/rand/1	−102.55
f20	Schwefel Function	6	150D	.8	.9	DE/rand/2	−546.5
f22	Gallagher's Gaussian 21-hi Peaks Function	10	150D	.8	.9	DE/rand/2	−1000

of the classical synchronous implementation of this model. Therefore, it is interesting to determine whether the proposed algorithm challenges the classical implementations in terms of the number of evaluations needed to achieve the required quality solution. In order to obtain a fair comparison between the proposal (`asynPDE`) and the serial classical DE (`seqDE`), the enhancements proposed in Section 5, that is, the logarithmic search, the local solver, and the tabu list, were disabled.

The BBOB-2009 data set [18] has been used as a benchmarking testbed due to its popularity and accessibility. This data set is composed of 24 noiseless benchmark functions to be minimized. Although tests have been carried out using the whole BBOB-2009 data set, only five benchmark functions have been selected to illustrate the experimental results in this paper.

Table 1 shows the five selected benchmarks and some of the configuration parameters used in the following experiments. 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. 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.

For the selection of the settings in these experiments, in general, the suggestions in [10] have been followed. Regarding *F* and *CR*, the settings $F \in [0.5, 1]$ and $CR \in [0.8, 1]$ are recommended. Thus, in these experiments $F = 0.9$ and $CR = 0.8$ have been chosen. Concerning the mutation strategy, among those suggested in [10], namely DE/best/1, DE/best/2, DE/rand/1 and DE/rand/2, the last one has been chosen since some of these benchmarks are multimodal, where this MSt remained most competitive and slightly faster to converge to the value to reach [13]. For the population size, although in [10] a guideline is given where a setting of the DE population size to about ten times the dimensionality of the problem is proposed, this indication is not confirmed in recent studies, such as in [54] where it is shown that a population size lower than the dimensionality of the problem can be optimal in many cases. Since the parallel algorithm will scatter the population matrix between the number of processors, and taking into account that when the population size is small the probability of premature convergence and stagnation may be higher, a $NP = (5 \times D) \times 30 = 150 \times D$ has been chosen for these experiments, where the number of processors range between 5 and 60. The only exception was function *f19-Composite Griewank–Rosenbrock Function F8F2*, which is a highly multimodal function that, with the previous settings, frequently fall into an undesired stagnation condition. Therefore, to prevent stagnation in this function NP has been augmented to $3000 \times D$, *CR* has been set to 0.9, and MSt to DE/rand/1.

In parallel island DE algorithms, new parameters have to be also considered (see Table 2), such as migration frequency (μ), island size (λ), communication topology (Tp) between processors, or selection policy (SP) and replacement policy (RP) in the migration step. In SP and RP policy, there are two possible values: random mode (RR) when individuals are selected or replaced randomly, and better/worse mode (BW) when the better individuals are selected to be sent and the worst individuals are replaced.

In addition, the proposed parallel algorithms was tested using different combination of *CR* and *F* values in different processes, which enhances diversity as well [39,41]. Experiments combining these parameters have been thoroughly performed. Local configuration (LC) parameter manages this property: it can be homogeneous (Homo), when all processes have the same attributes, and heterogeneous (Hete) in the opposite case. In heterogeneous case, the values used for the results reported were $F = \{.9, .7\}$ and $CR = \{.9, .7, .2\}$. The combination policy is the following: the algorithm assigns to each process one pair *F-CR* belonging to the sorted list of all possible combinations of the above sets of *F* and *CR*. When all the pairs have been distributed, it restarts the previous sorted list and the algorithm continues the distribution.

In the experiments performed with BBOB-2009 benchmarks, it is important to note that the global population has been distributed among the cores. This assures that the improvement in the diversity comes from the asynchronous sparse individual exchanges in the parallel version, allowing to analyze the impact of this technique on the algorithm computational efficiency. So, the island size for these experiments is *NP*/*PROC*.

Experiments, consisting of 50 independent runs each, were carried out on Intel Sandy Bridge nodes of CESGA SVG Linux cluster [55]. Table 3 shows, for the *f-22 Gallagher's Gaussian 21-hi Peaks Function* of the BBOB-2009 data set, the percentage of executions that achieve the optimal value (% hit), the average value of the achieved tolerances (avg), the mean number of evaluations (# evals) and the mean execution time. Results for the serial DE (`seqDE`) and for the proposed parallel island-based algorithm (`asynPDE`) with different number of cores are shown. First, it can be observed that when the number of cores grows (P-5 to P-60 in Table 3), the number of the executions that achieve the quality

Table 2
Parallel implementation parameters.

Parameter	Value	Description
Island size (λ)	n/nproc	Size of population matrix for each parallel process.
Migration frequency (μ)	12.5% (every 8 iterations)	Number of iterations to enter in migration stage.
Topology (Tp)	ring/star	Communication topology between processors
Selection policy (SP)	random (RR) / best (RB)	It selects a set of elements to send in migration stage.
Replacement policy (RP)	random (RR) / worst (RW)	It replaces a set of elements in subpopulation matrix.
Local configuration (LC)	Homogeneous/Heterogeneous	It explains how is the value of *CR* and *F* for each processor.

Table 3
AsynPDE vs SeqDE. Function: *f-22*. Values in the table in order of appearance: percentage of success (hits), average of the best value for each run, mean of the number of consumed evaluations, mean and standard deviation of the execution time, and speedup calculated as sp = time(seqDE)/time(asynPDE). Parallel implementation parameters: Tp = ring, SP/RP = RR, μ = 12,5%, and LC = Heterogeneous. Stopping criteria: maximum number of evaluations = 3,000,000 or achieve VTR = −1000 with an absolute tolerance TOL = 1e−3.

Algorithm	Processors	% hit	avg(bestx)	mean # evals	mean time ±std (s)	mean speedup
seqDE	1	4	−999.4315	3001950	10.199 ± 0.037	–
	5	20	−999.5550	2937648	1.688 ± 0.086	6.04
	10	44	−999.7322	2841867	0.979 ± 0.097	10.42
asynPDE	20	60	−999.8317	2744889	0.481 ± 0.055	21.21
	40	78	−999.9434	2576759	0.246 ± 0.039	41.36
	60	92	−999.9987	2537147	0.187 ± 0.023	54.34

solution increases. It can also be observed that the number of evaluations needed to achieve the optimal solution decreases when the number of processors grows. In short, these results show the effectiveness of the parallel algorithm proposed in terms of quality of the solution, where less evaluations are needed to come to the required tolerance when more processors take part in the search. Table 3 shows also speedups for the proposed algorithm asynPDE versus the seqDE with different number of processors. At first glance, it can be observed that the speedups obtained are superlinear. That is due to the diversity introduced by the migration phase in the parallel execution, that actually improves the effectiveness of the DE algorithm.

Two different stopping criteria for the algorithms were considered in these experiments: solution quality, for an horizontal view, and maximum effort, for a vertical approach. Thus, two different speedup definitions were used to compare the sequential and the parallel algorithm: a speedup with solution quality stop, and a speedup with predefined effort. The first one shows how fast the parallel algorithm reaches DE solution versus the sequential algorithm. This is the case of the results shown in Table 3. This speedup is due both to the distribution of computations among the processors, and to the effectiveness of the parallel algorithm, which requires fewer iterations as the number of processors grows. However, in most of the cases, the sequential algorithm and the parallel one running on few processors do not reach the quality solution before the maximum allowed effort. In those cases, the best way to fairly compare sequential and parallel executions is to stop all of them at a predefined effort, that is, for a predefined number of evaluations. Since the number of iterations performed are, in this case, the same, these speedup results help to analyze how faster are the iterations of the parallel algorithm versus the classical iterations. Table 4 shows the speedup with quality solution for those configurations that reach the required tolerance in a reasonable amount of time and eventually obtain the best speedups. The value-to-reach (VTR) in each experiment, generated by the BBOB benchmark software, is shown in Table 1. It is noticeable that, in most cases, the heterogeneous strategies are superior to homogeneous ones, since they benefit from search diversification.

Fig. 2 shows the speedup from a vertical view, that is, for a predefined effort. Results for both the proposed asynchronous algorithm (asynPDE) and for a synchronous version of the classical DE (synPDE) are shown for comparison purposes.

It can be seen that, for a small number of processors, the speedup achieved when using the stopping criterion of predefined effort (Fig. 2) is similar in synchronous and asynchronous strategies, indicating that the evaluation time is similar in both. For a small number of processors, the synchronous version obtains higher speedups from the horizontal view, that is, using the solution quality as stopping criterion (data shown in Table 4). This is because fewer evaluations are needed to reach the solution in the synchronous version. The reason is that, when the processes are synchronized in the migration phase, the new evaluations are performed after exchanging solutions between processors. In the asynchronous version, however, the new evaluations after the migration point can start before the complete exchange of solutions, since no synchronization is forced between processes.

As the number of processors increases, it can be observed that the scalability of asynchronous proposal improves versus the synchronous one. This is because the synchronization slows down the processes, since it implies more processes' stalls while waiting for data. Thus, the asynchronous version, in spite of requiring further evaluations, reaches the solution in a shorter time.

In summary, these results show that the proposed asynchronous island-based algorithm achieves good speedup, and it scales better than the synchronous algorithm for a large number of processors.

6.2. Result for parameter estimation problems in systems biology

In computational systems biology, parameter estimation in models of nonlinear dynamic systems seeks to obtain a decision vector **p**, that optimizes the cost function in order to obtain quantitative predictions which match a given set of experimental

Table 4
Quality value test in BBOB benchmarks. Values in the table: mean and standard deviation of seqDE runtimes; parallel parameters, LC and Tp, that obtain the best results for each parallel algorithm (asynPDE or synPDE); speedup results calculated as sp = time(seqDE)/time((a)synPDE). Common parallel implementation parameters: SP/RP = RR. Stopping criteria: achieve VTR with an absolute tolerance TOL = 1e−3.

Function	seqDE: mean time±std (s)	Algorithm	LC	Tp	Mean speedup				
					Number of processors				
					5	10	20	40	60
f8 - Rosenbrock, original	13.55 ± 1.17	asynPDE	Hete	Ring	5.2	10.8	21.1	34.0	43.8
		synPDE	Hete	Ring	5.0	9.5	16.9	15.4	13.2
f15 - Rastrigin	20.41 ± 0.73	asynPDE	Hete	Ring	4.3	8.7	43.4	171.9	221.9
		synPDE	Homo	Ring	9.8	22.3	69.0	123.1	136.4
f17 - Schaffers	7.26 ± 0,35	asynPDE	Hete	Star	6.7	12.6	26.1	70.7	120.8
		synPDE	Hete	Star	7.6	15.9	29.2	56.4	68.3
f19 - Composite Griewank-Rosenbrock	26.93 ± 10.39	asynPDE	Homo	Ring	7.3	11.2	40.9	124.8	155.4
		synPDE	Hete	Ring	9.0	18.6	45.3	95.4	128.0
f20 - Schwefel	18.12 ± 0,88	asynPDE	Hete	Ring	21.3	31.9	113.3	318.6	447.2
		synPDE	Hete	Ring	21.8	50.1	131.7	235.2	252.0

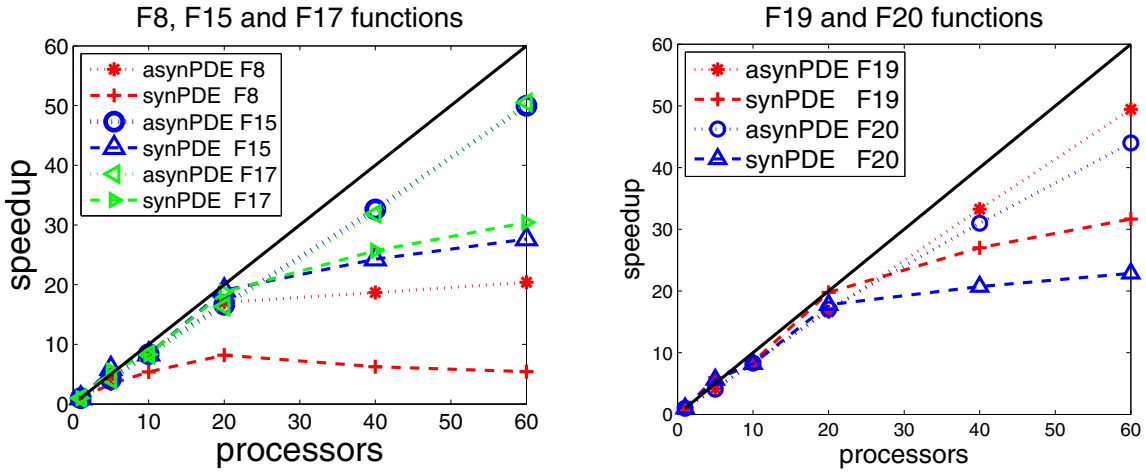

Fig. 2. Predefined effort test in BBOB benchmarks. Parallel implementation parameters: LC = Heterogeneous, Tp = ring, SP/RP = RR and μ = 12,5%. Stopping criteria: maximum number of evaluations = 1,000,200.

time-series data, satisfying other possible constraints. Finding the optimal values of this decision vector $\mathbf{p}$ can be represented as a nonlinear programming problem (NLP), where the objective is to minimize the cost function:

$$J = \sum_{\varepsilon=1}^{n_\varepsilon} \sum_{o=1}^{n_o^\varepsilon} \sum_{s=1}^{n_s^{\varepsilon,o}} (ym_s^{\varepsilon,o} - y_s^{\varepsilon,o}(\mathbf{p}))^T W (ym_s^{\varepsilon,o} - y_s^{\varepsilon,o}(\mathbf{p})) \quad (5)$$

where n_ε is the number of experiments, n_ε^ϵ is the number of the observables (state variables measured experimentally), $n_s^{\epsilon,o}$ corresponds with the number of the samples per observable per experiment, $ym_s^{\varepsilon,o}$ are the measured data, $y_s^{\varepsilon,o}(\mathbf{p})$ are the model predictions, and W is a scaling matrix that balances the results of the observables. This optimization is also subject to the following constraints:

$$\dot{x} = f(x, \mathbf{p}, t) \quad (6)$$

$$x(t_o) = x_o \quad (7)$$

$$y = g(x, \mathbf{p}, t) \quad (8)$$

$$h_{eq}(x, y, \mathbf{p}) = 0 \quad (9)$$

$$h_{in}(x, y, \mathbf{p}) \leq 0 \quad (10)$$

$$\mathbf{p}^L \leq \mathbf{p} \leq \mathbf{p}^U \quad (11)$$

where x is the vector of state variables, x_o are the initial conditions, f is the nonlinear dynamic problem with the differential-algebraic constraints (DAEs), g corresponds with the observation function, h_{eq} and h_{in} are equality and inequality constraints, and p^L and p^U are lower and upper bounds for the decision vector $\mathbf{p}$. The equality dynamic constraints, Eqs. (6)–(7), are solved as an inner initial value problem for each decision vector. Note that in our case studies constraints Eqs. (9)–(10) were not present, but could be handled using differential-algebraic solvers and suitable penalty functions, as described in [56]. Upper and lower bounds for the parameters Eq. (11) were handled by a reflection strategy [11] during the global phase and automatically by the solver NL2SOL during the local searches.

In order to evaluate the global algorithm proposed in this paper (`asynPDE_IH`, that combines the diversification introduced by the parallelization evaluated previously and the intensification of the local search) three challenging parameter estimation problems from the domain of computational system biology were considered. These problems are known to be particularly hard due to their ill-conditioning and non-convexity [57,4]:

- *Circadian* model: parameter estimation in a dynamic model of the circadian clock in the plant *Arabidopsis thaliana*, as presented in [58]. 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 [59] 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 [57].

The aim of these experiments is to demonstrate the potential of the proposed techniques for improving the convergence and execution time of very hard problems. Since the goal of including the enhancements proposed in Section 5 in the DE algorithm is to improve the effectiveness of each local evaluation, it is desirable to enhance the exploration in the solution space by means of increasing the diversification in the parallel threads. Therefore, for these experiments, the heterogenous LC was used in all the cases.

A multicore cluster was used to carry out these experiments. It consists of 16 nodes powered by two octa-core Intel Xeon E5-2660 CPUs with 64 GB of RAM. The cluster nodes are connected through an InfiniBand FDR network. OpenMPI library version 1.6.2 has been used to compile the parallel implementations, and the experiments were carried out using from 1 to 30 processors. In these benchmarks, the execution of only one test of `seqDE` could take hours or even days to complete. For each experiment 30 independent runs have been performed.

Fig. 3 shows the best convergence curve for these three parameter estimation problems. Results show that, on the one hand, the logarithmic search, the local search, and the tabu list, on `seqDE_IH` achieve by themselves a drastically reduction in the execution time required for convergence, due to a reduction in the number of evaluations needed; on the other hand, the parallelization on `asynPDE_IH` also improves quality of mean solution since more evaluations are performed in the same amount of time.

Fig. 4 illustrates how the proposed `asynPDE_IH` reduces the big variability of execution time in the sequential version of the algorithm. It demonstrates that when the number of processors grows up in the asynchronous method, the outliers of the execution time decrease. This is a important feature in the performance of this solver, because it reduces the average execution time for each benchmark.

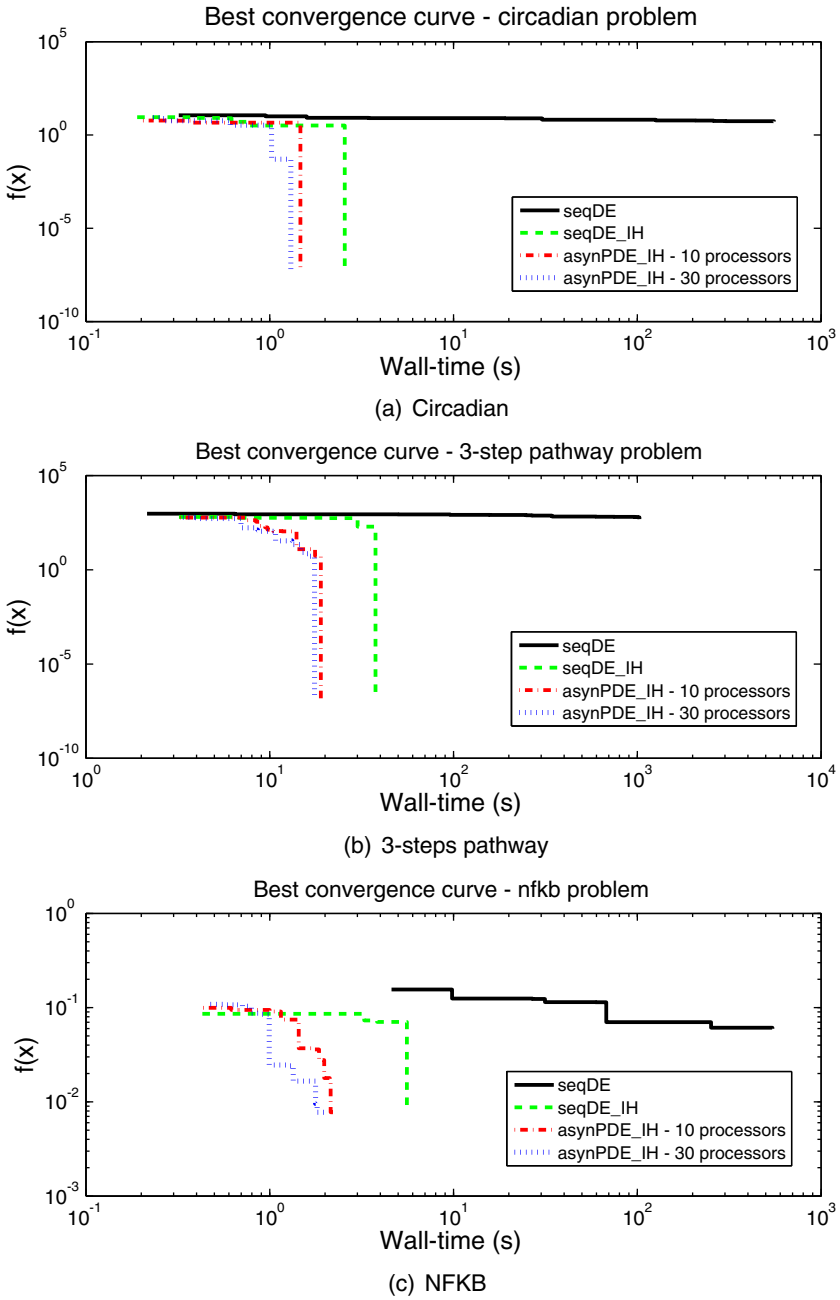

Fig. 3. Best convergence curves for parameter estimation problems. Configuration DE parameters: NP = 10×D, MSt = DE/rand/2. Parallel implementation parameters: LC = Heterogeneous, Tp = ring, SP = RB, RP = RW, μ = 33% (m = 3), and λ = NP/NPROC. Local solver condition: L = 6. Stopping criteria: achieve (a and b) VTR (value to reach) = 1e−5 (c) VTR = 1e−2.

Fig. 5 shows the speedup for a quality value test in these three parameter estimation problems. These speedups were calculated comparing execution times of `synPDE_IH` and `asynPDE_IH` with `seqDE_IH`. Note that the `seqDE` execution time can not be used to compare with, because of its unreasonable amount of time to converge (more than 48 h for each 3-step and NFKB experiment, when they do not get stuck in local optima). These figures show that the proposed `asynPDE_IH` reduces significantly execution time, and scales better than the `synPDE_IH` algorithm for a large number of processors.

The speedup results in these figures may appear to be modest, since they have been calculated related to the `seqDE_IH` execution time instead of to the `seqDE` execution time. Table 5 shows results for both the execution time and the number of evaluations performed in these experiments. Note that the number of evaluations reported in this table includes those performed by the local solver. Although each external iteration of the algorithm involves more evaluations when the local solver is enabled, the overall result shows that the enhanced algorithm needs less total number of evaluations to reach the optimum value.

In these problems, the improvement introduced by the local solver achieves convergence with very few external iterations of the algorithm, that is, with very few migration phases between islands. Therefore, adding more processes does not significantly improve the execution time. In the NFKB problem, the convergence improvement achieved by the local solver is not that noticeable, thus, the parallelization continues introducing diversity when the number of processes increases so performance

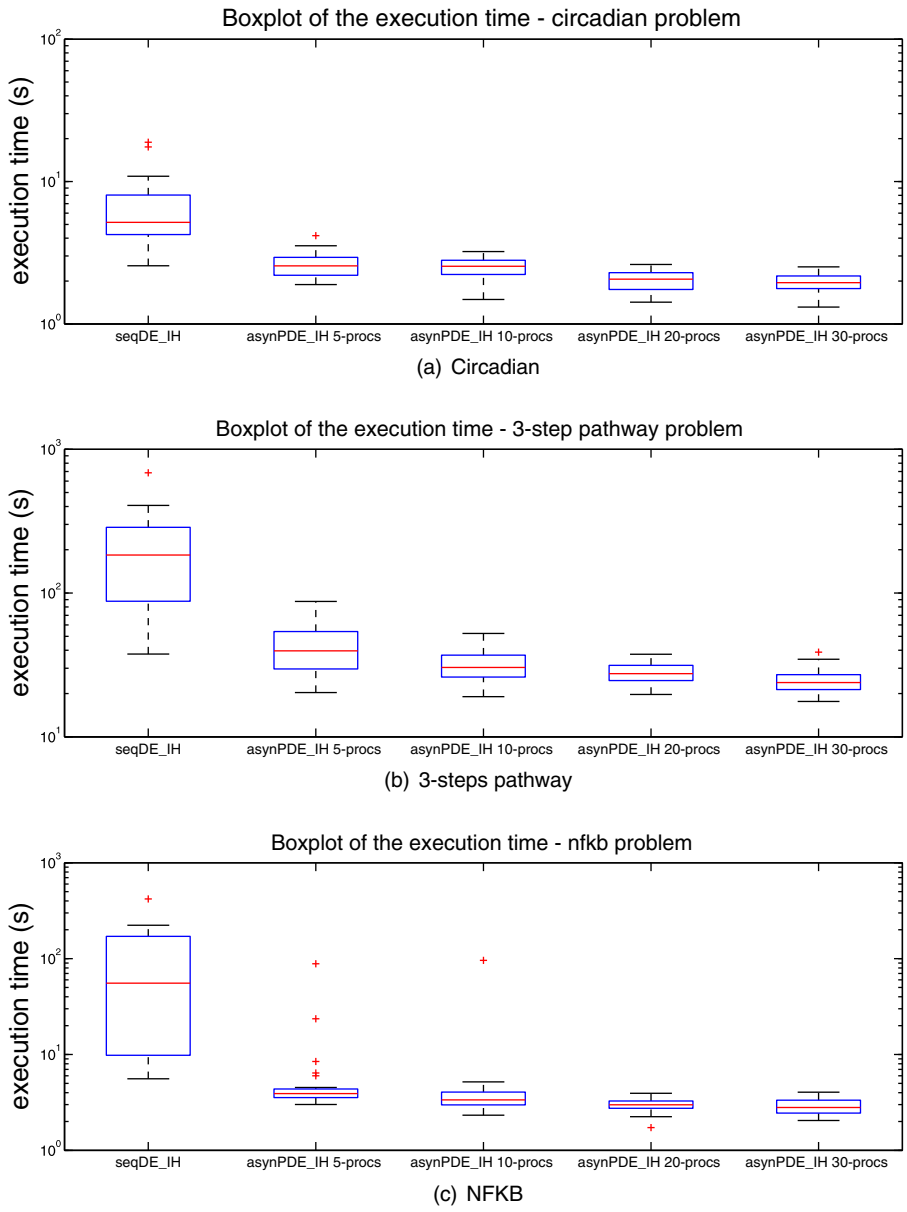

Fig. 4. Boxplot of the execution time for parameter estimation problems with asynPDE_IH. Configuration: same configuration as in Fig. 3.

Table 5
Execution time table of the experiments. Configuration: same configuration as in Fig. 3.

SB problem	Algorithms	Processors	Mean # evals	Mean # evals/process	Mean # iterations	Mean time±std (s)
circadian	seqDE	1	11,521,835	11,521,835	88,627.7	26,856.12 ± 3424.14
	seqDE_IH	1	3534.37	3534.37	11.20	6.46 ± 3.81
	asynPDE_IH	10	10,382.37	1038.23	7.10	2.50 ± 0.27
	asynPDE_IH	20	17,398.73	869.93	6.70	2.02 ± 0.32
	asynPDE_IH	30	23,935.47	797.85	6.58	1.96 ± 0.43
3-step pathway	seqDE	1	–	–	–	–
	seqDE_IH	1	30,580.97	30,580.97	32.60	202.40 ± 138.53
	asynPDE_IH	10	43,802.94	4380.29	11.03	32.01 ± 8.77
	asynPDE_IH	20	73,970.33	3698.57	9.28	27.79 ± 4.74
	asynPDE_IH	30	95,001.10	3166.70	8.20	24.67 ± 5.06
NFKB	seqDE	1	–	–	–	–
	seqDE_IH	1	11,253.90	11,253.90	34.00	91.37 ± 98.68
	asynPDE_IH	10	4370.10	437.01	8.30	6.55 ± 16.90
	asynPDE_IH	20	4688.27	234.41	7.31	2.98 ± 0.52
	asynPDE_IH	30	5839.67	194.66	6.76	2.88 ± 0.56

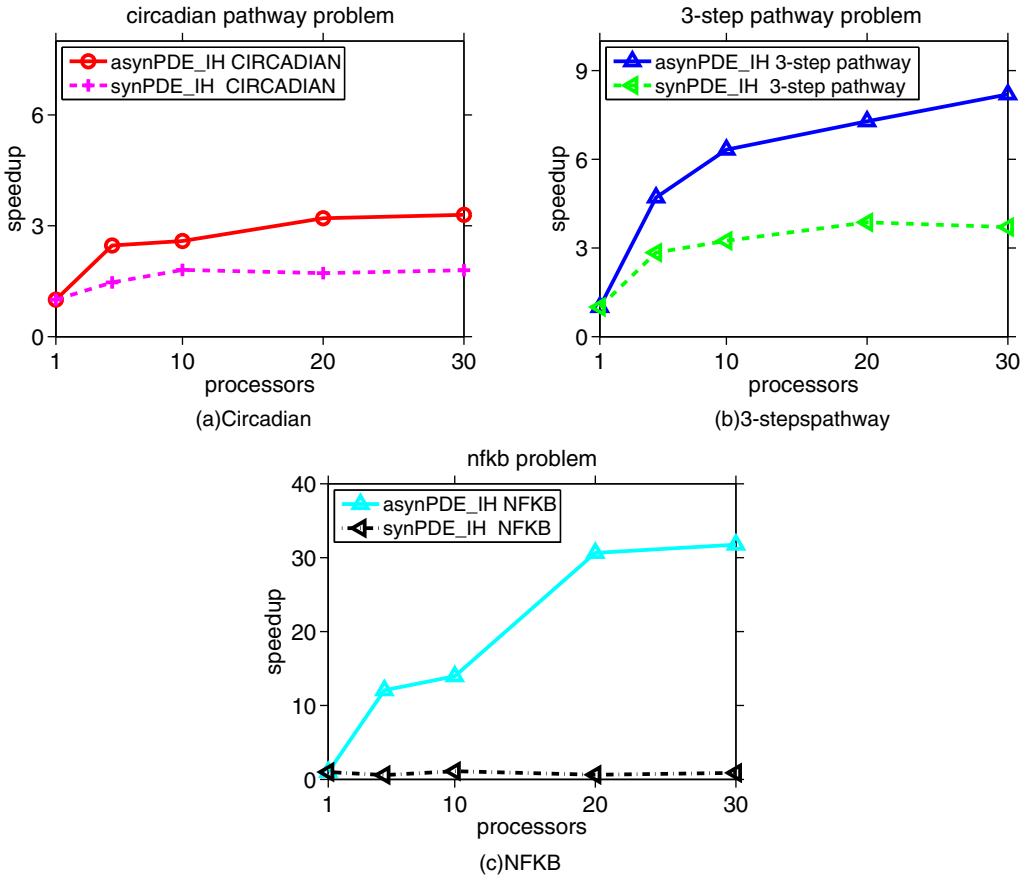

Fig. 5. Speedups calculated with respect to the execution time of seqDE_IH. Configuration: same configuration as in Fig. 3.

enhancement is achieved. In all the cases, when the number of external iterations are drastically diminished, there is no room for further improvement despite the increased number of processors.

It is important to note that the local solver introduces a great overhead on the execution of each evaluation. Moreover, it is responsible for the lack of synchronization between processes at the migration step, since it leads processes into a more computationally unbalanced scenario, thus, giving advantage to the proposed asynchronous solution. When more evaluations are required, the synchronization between processes will decrease, thus, better performance will be achieved by the asynchronous proposal. This is the case of the NFKB problem, that requires more evaluations to converge than the other two benchmarks.

To prove the statistical significance of the results, the Wilcoxon Rank-sum test has been applied for a confidence level of 0.95. Table 6 shows the results of the test, using the runtimes obtained in the experiments. As it can be seen, the *p*-value is always smaller than the significance, thus, `asynPDE_IH` outperforms both the `seqDE_IH` and the `synPDE_IH`. Note also that, when comparing the asynchronous proposal with the synchronous one, the values of *p*-value are higher when using few processors and they decrease when the number of processors grows. This demonstrates the scalability of the proposal over the synchronous version.

Compared to the serial DE (`seqDE`), the proposed enhancements improve the execution time in several orders of magnitude. For the circadian problem, the execution of the `seqDE` lasts more than 7 h while the proposed `seqDE_IH` requires only a few seconds, which

Table 6
Wilcoxon signed ranks test with a significance level $\alpha = 0.05$. The parameters in the table R^+ and R^- are the sum of the positive/negative ranks. Configuration: same configuration as in Fig. 3.

SB problem	Algorithms	R^+	R^-	*p*-value
circadian	asynPDE_IH 10-procs versus seqDE_IH	465	0	1.73E−06
	asynPDE_IH 30-procs versus asynPDE_IH 10-procs	424	41	8.19E−05
	asynPDE_IH 10-procs versus synPDE_IH 10-procs	446	19	1.13E−05
	asynPDE_IH 30-procs versus synPDE_IH 30-procs	464	1	1.92E−06
3-step pathway	asynPDE_IH 10-procs versus seqDE_IH	465	0	1.73E−06
	asynPDE_IH 30-procs versus asynPDE_IH 10-procs	399	66	6.16E−04
	asynPDE_IH 10-procs versus synPDE_IH 10-procs	443	22	1.49E−05
	asynPDE_IH 30-procs versus synPDE_IH 30-procs	465	0	1.73E−06
NFKB	asynPDE_IH 10-procs versus seqDE_IH	465	0	1.73E−06
	asynPDE_IH 30-procs versus asynPDE_IH 10-procs	370	95	4.70E−03
	asynPDE_IH 10-procs versus synPDE_IH 10-procs	441	24	1.80E−05
	asynPDE_IH 30-procs versus synPDE_IH 30-procs	465	0	1.73E−06

demonstrates the potential of the proposed heuristics in the solution on these problems.

7. Conclusions

In this paper, we presented an improved Differential Evolution algorithm designed to solve complex problems in computational systems biology. The key idea is to achieve a proper balance of the exploration abilities of DE and the exploitation abilities of efficient local search. The method improves global search through an asynchronous parallel implementation based on a cooperative island-model. The improved local search is implemented by means of several heuristics (efficient local solver, tabu list, logarithmic search) which exploit the structure of parameter estimation problems in systems biology, the main application area considered here.

It should be noted that, although the method presented here is based on a parsimonious hybrid (global–local) design, the three heuristic enhancements introduced in the local search are fundamental to successfully exploit the special characteristics of these systems biology problems, which are typically very ill-conditioned and highly multimodal, as reviewed in [4]. The results obtained show that this improved local search mechanism, combined with the parallel cooperation scheme, allow an adequate balance between exploration and exploitation for the class of problems considered.

The experimental results show that (i) convergence time can be reduced by several orders of magnitude when the local search heuristics are included in the DE algorithm, and (ii) the asynchronous parallel strategy proposed attains a further reduction in the convergence time through collaboration of the parallel processes, demonstrating also a competitive speedup against the synchronous approaches.

As an example of the practical significance of this work, one of the systems biology benchmarks considered, the 3-steps pathway problem, typically requires more than 3 days of computation time using the classical version of DE executed in one of the cores of our test machine, but it can be solved in less than 1 min with the novel asynchronous parallel method presented.

Although the improved DE 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.

Acknowledgements

This research received financial support from the Spanish Ministerio de Economía y Competitividad (and the FEDER) through the project “MultiScales” (DPI2011-28112-C04-03), and from the CSIC intramural project “BioREDES” (PIE-201170E018). It has been also partially supported by the Spanish Ministerio de Ciencia e Innovación (Project TIN2013-42148-P) and by the Galician Government (consolidation program of competitive reference groups GRC2013/055). We acknowledge the computational resources provided by CESGA. D. R. Penas acknowledges financial support from the MICINN-FPI programme.

References

[1] H. Greenberg, W. Hart, G. Lancia, Opportunities for combinatorial optimization in computational biology, INFORMS J. Comput. 16 (3) (2004) 211–231.
[2] P. Larrañaga, B. Calvo, R. Santana, C. Bielza, J. Galdiano, I. Inza, J. Lozano, R. Armañanzas, G. Santafé, A. Pérez, V. Robles, Machine learning in bioinformatics, Brief. Bioinform. 7 (1) (2006) 86–112.
[3] J.R. Banga, Optimization in computational systems biology, BMC Syst. Biol. 2 (2008), art. no. 47.
[4] A. Villaverde, J.R. Banga, Reverse engineering and identification in systems biology: strategies, perspectives and challenges, J. R. Soc. Interface 11 (91) (2014), art. no. 20130505.
[5] T. Crainic, M. Toulouse, Parallel strategies for meta-heuristics, in: Handbook of Metaheuristics, vol. 57 of International Series in Operations Research & Management Science, Springer, US, 2003, pp. 475–513.
[6] E. Alba, Parallel Metaheuristics: A New Class of Algorithms, Wiley-Interscience 47 (2005).
[7] T. Perkins, J. Jaeger, J. Reinitz, L. Glass, Reverse engineering the gap gene network of drosophila melanogaster, PLoS Comput. Biol. 2 (5) (2006) 417–428.
[8] L. Jostins, J. Jaeger, Reverse engineering a gene network using an asynchronous parallel evolution strategy, BMC Syst. Biol. 4 (2010), art. no. 17.
[9] A. Villaverde, J. Egea, J.R. Banga, A cooperative strategy for parameter estimation in large scale systems biology models, BMC Syst. Biol. 6 (2012), art. no. 75.
[10] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, J. Glob. Optim. 11 (4) (1997) 341–359.
[11] K. Price, R.M. Storn, J.A. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, Springer Science & Business Media, 2006.
[12] U.K. Chakraborty, Advances in Differential Evolution, Springer Publishing Company, 2008.
[13] S. Das, P. Suganthan, Differential Evolution: A Survey of the State-of-the-Art, IEEE Trans. Evolut. Comput. 15 (1) (2011) 4–31.
[14] G. Quaranta, G. Marano, R. Greco, G. Monti, Parametric identification of seismic isolators using differential evolution and particle swarm optimization, Appl. Soft Comput. J. 22 (2014) 458–464.
[15] E.C.T. Zúñiga, I.L.L. Cruz, A.R. García, Parameter estimation for crop growth model using evolutionary and bio-inspired algorithms, Appl. Soft Comput. J. 23 (2014) 474–482.
[16] S. Da Ros, G. Colusso, T. Weschenfelder, L. De Marsillac Terra, F. De Castilhos, M. Corazza, M. Schwaab, A comparison among stochastic optimization algorithms for parameter estimation of biochemical kinetic models, Appl. Soft Comput. J. 13 (5) (2013) 2205–2214.
[17] D. Penas, J.R. Banga, P. González, R. Doallo, A parallel differential evolution algorithm for parameter estimation in dynamic models of biological systems, in: 8th International Conference on Practical Applications of Computational Biology & Bioinformatics (PACBB 2014), vol. 294 of Advances in Intelligent Systems and Computing, Springer International Publishing, 2014, pp. 173–181.
[18] N. Hansen, A. Auger, S. Finck, R. Ros, Real-Parameter Black-Box Optimization Benchmarking 2010: Experimental setup, Tech. Rep. Rapports de Recherche RR-6828, Institut National de Recherche en Informatique et en Automatique (INRIA), 2009.
[19] F. Neri, V. Tirronen, Recent advances in differential evolution: a survey and experimental analysis, Artif. Intell. Rev. 33 (1–2) (2010) 61–106.
[20] L. Weihmann, D. Martins, L. dos Santos Coelho, Modified differential evolution approach for optimization of planar parallel manipulators force capabilities, Expert Syst. Appl. 39 (6) (2012) 6150–6156.
[21] N. Noman, H. Iba, Enhancing differential evolution performance with local search for high dimensional function optimization, in: GECCO 2005 – Genetic and Evolutionary Computation Conference, 2005, pp. 967–974.
[22] N. Noman, H. Iba, Accelerating differential evolution using an adaptive local search, IEEE Trans. Evolut. Comput. 12 (1) (2008) 107–125.
[23] V. Tirronen, F. Neri, T. Rossi, Enhancing differential evolution frameworks by scale factor local search – part I, in: 2009 IEEE Congress on Evolutionary Computation, CEC 2009, 2009, pp. 94–101.
[24] F. Neri, V. Tirronen, T. Kärkkäinen, Enhancing differential evolution frameworks by scale factor local search – part II, in: IEEE Congress on Evolutionary Computation, CEC 2009, 2009, pp. 118–125.
[25] M. Srinivas, G. Rangaiah, Differential evolution with tabu list for global optimization and its application to phase equilibrium and parameter estimation problems, Ind. Eng. Chem. Res. 46 (10) (2007) 3410–3421.
[26] J.I. Kushida, K. Oba, A. Hara, T. Takahama, 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, pp. 639–644.
[27] E. Schneider, R.A. Krohling, Differential evolution and tabu search to find multiple solutions of multimodal optimization problems, in: Soft Computing in Industrial Applications, vol. 223 of Advances in Intelligent Systems and Computing, Springer International Publishing, 2014, pp. 61–69.
[28] D. Zaharie, D. Petcu, Parallel implementation of multi-population differential evolution, in: Proceedings of the NATO Advanced Research Workshop on Concurrent Information Processing and Computing, 2003, pp. 223–232.
[29] D. Tasoulis, N. Pavlidis, V. Plagianakos, M. Vrahatis, Parallel differential evolution, in: Proceedings of the 2004 Congress on Evolutionary Computation, CEC2004, vol. 2, 2004, pp. 2023–2029.
[30] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, PVM: Parallel Virtual Machine, in: A Users' Guide and Tutorial for Networked Parallel Computing, MIT press, 1994.
[31] M.S. Ntipteni, I.M. Valakos, I.K. Nikolos, An asynchronous parallel differential evolution algorithm, in: Proceedings of the ERCOFTAC Conference on Design Optimisation: Methods and Application, 2006.
[32] J. Olenšek, T. Tuma, J. Puhan, Árpád Burmen, A new asynchronous parallel global optimization method based on simulated annealing and differential evolution, Appl. Soft Comput. 11 (1) (2011) 1481–1489.
[33] D. Izzo, M. Rucinski, C. Ampatzis, Parallel global optimisation meta-heuristics using an asynchronous island-model, in: Proceedings of the Eleventh Conference on Congress on Evolutionary Computation, CEC'09, 2009, pp. 2301–2308.

[34] J. Apolloni, J. García-Nieto, E. Alba, G. Leguizamón, Empirical evaluation of distributed differential evolution on standard benchmarks, Appl. Math. Comput. 236 (2014) 351–366.
[35] M. Rucinski, D. Izzo, F. Biscani, On the impact of the migration topology on the island model, Parallel Comput. 36 (10–11) (2010) 555–571.
[36] M. Weber, F. Neri, V. Tirronen, Distributed differential evolution with explorative-exploitative population families, Genet. Program. Evol. Mach. 10 (4) (2009) 343–371.
[37] J. Brest, S. Greiner, B. Boskovic, M. Mernik, V. Zumer, Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems, IEEE Trans. Evolut. Comput. 10 (6) (2006) 646–657.
[38] M. Weber, V. Tirronen, F. Neri, Scale factor inheritance mechanism in distributed differential evolution, Soft Comput. 14 (11) (2010) 1187–1207.
[39] M. Weber, F. Neri, V. Tirronen, A study on scale factor in distributed differential evolution, Inf. Sci. 181 (12) (2011) 2488–2511.
[40] S.-Z. Zhao, P.N. Suganthan, S. Das, Self-adaptive differential evolution with multi-trajectory search for large-scale optimization, Soft Comput. 15 (11) (2011) 2175–2185.
[41] M. Weber, F. Neri, V. Tirronen, A study on scale factor/crossover interaction in distributed differential evolution, Artif. Intell. Rev. 39 (3) (2013) 195–224.
[42] M. Weber, F. Neri, V. Tirronen, Shuffle or update parallel differential evolution for large-scale optimization, Soft Comput. 15 (11) (2011) 2089–2107.
[43] W. Zhu, Massively parallel differential evolution-pattern search optimization with graphics hardware acceleration: an investigation on bound constrained optimization problems, J. Glob. Optim. 50 (3) (2011) 417–437.
[44] H. Wang, S. Rahnamayan, Z. Wu, Parallel differential evolution with self-adapting control parameters and generalized opposition-based learning for solving high-dimensional optimization problems, J. Parallel Distrib. Comput. 73 (1) (2013) 62–73.
[45] F. Aydemir, A. Günay, F. Öztoprak, S. Birbil, P. Yolum, Multiagent cooperation for solving global optimization problems: an extendible framework with example cooperation strategies, J. Glob. Optim. 57 (2) (2013) 499–519.
[46] M. Črepinšek, S.-H. Liu, M. Mernik, Exploration and exploitation in evolutionary algorithms: a survey, ACM Comput. Surv. 45 (3) (2013) 35:1–35:33.
[47] E. Alba, G. Luque, S. Nesmachnow, Parallel metaheuristics: recent advances and new trends, Int. Trans. Oper. Res. 20 (1) (2013) 1–48.
[48] J. Egea, M. Rodríguez-Fernández, J.R. Banga, R. Martí, Scatter search for chemical and bio-process optimization, J. Glob. Optim. 37 (3) (2007) 481–503.
[49] J.E. Dennis Jr., D.M. Gay, R.E. Welsch, Algorithm 573: Nl2sol – an adaptive nonlinear least-squares algorithm [e4], ACM Trans. Math. Softw. 7 (3) (1981) 369–383.
[50] M. Powell, Convergence properties of a class of minimization algorithms, Nonlinear Program. 2 (0) (1975) 1–27.
[51] F. Neri, C. Cotta, P. Moscato, Handbook of Memetic Algorithms, vol. 379, Springer, 2012.
[52] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, S. Schulenburg, Hyper-heuristics: an emerging direction in modern search technology, in: Handbook of Metaheuristics, Springer, 2003, pp. 457–474.
[53] M. Črepinšek, S.-H. Liu, M. Mernik, Replication and comparison of computational experiments in applied evolutionary computing: common pitfalls and guidelines to avoid them, Appl. Soft Comput. 19 (0) (2014) 161–170.
[54] F. Neri, V. Tirronen, On memetic differential evolution frameworks: a study of advantages and limitations in hybridization, in: IEEE Congress on Evolutionary Computation, 2008. CEC 2008, 2008, pp. 2135–2142.
[55] CESGA, Svg specifications, https://www.cesga.es/gl/infraestructuras/computacion/svg
[56] J.R. Banga, E. Balsa-Canto, C.G. Moles, A.A. Alonso, Dynamic optimization of bioprocesses: efficient and robust numerical strategies, J. Biotechnol. 117 (4) (2005) 407–419.
[57] C. Moles, P. Mendes, J.R. Banga, Parameter estimation in biochemical pathways: a comparison of global optimization methods, Genome Res. 13 (11) (2003) 2467–2474.
[58] J. Locke, A. Millar, M. Turner, Modelling genetic networks with noisy and varied experimental data: the circadian clock in *Arabidopsis thaliana*, J. Theor. Biol. 234 (3) (2005) 383–393.
[59] T. Lipniacki, P. Paszek, A. Brasier, B. Luxon, M. Kimmel, Mathematical model of nf-κb regulatory module, J. Theor. Biol. 228 (2) (2004) 195–215.