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

Hybrid parallel multimethod hyperheuristic for mixed-integer dynamic optimization problems in computational systems biology

Patricia González · Pablo Argüeso-Alejandro · David R. Penas · Xoan C. Pardo · Julio Saez-Rodriguez · Julio R. Banga · Ramón Doallo

Received: date / Accepted: date

Abstract This paper describes and assesses a parallel multimethod hyperheuristic for the solution of complex global optimization problems. In a multimethod hyperheuristic, different metaheuristics cooperate to outperform the results obtained by any of them isolated. The results obtained show that the cooperation of individual parallel searches modifies the systemic properties of the hyperheuristic, achieving significant performance improvements versus the sequential and the non-cooperative parallel solutions. Here we present and evaluate a hybrid parallel scheme of the multimethod, using both message-passing (MPI) and shared memory (OpenMP) models. The hybrid parallelization allows to achieve a better trade-off between performance and computational resources, through a compromise between diversity (number of islands) and intensity (number of threads per island). For the performance evaluation we considered the general problem of reverse engineering nonlinear dynamic models in systems biology, which yields very large mixed-integer dynamic optimization (MIDO) problems. In particular, three very challenging problems from the domain of dynamic modelling of cell signaling were used as case studies. In addition, experiments have been carried out in a local cluster, a large supercomputer and a public cloud, to show the suitability of the proposed solution in different execution platforms.

P. González, P. Argüeso-Alejandro, X.C. Pardo and R. Doallo
Computer Architecture Group, CITIC, University of A Coruña (Spain)
E-mail: patricia.gonzalez@udc.es

David R. Penas
MODESTYA research group, IMAT, University of Santiago de Compostela (Spain)

Julio Saez-Rodriguez
Heidelberg University, Faculty of Medicine, Institute for Computational Biomedicine, Bioquant (Germany)

Julio R. Banga
BioProcess Engineering Group, IIM-CSIC, Spanish National Research Council (Spain)

Keywords reverse engineering · computational systems biology · mixed-integer optimization problems · parallel metaheuristics · global optimization · multimethod optimization

1 Introduction

Global optimization methods can be used to solve complex problems in many domains of science and engineering. They are attaining increasing popularity in many key areas of the life sciences, including bioinformatics and computational systems biology [7]. In the particular case of computational systems biology, there is a great interest in reverse engineering, i.e. modeling biosystems by means of dynamic models which are able to capture well their time-varying nature [49]. Many research efforts are focused in building and exploiting these dynamic models through mathematical optimization techniques. Here we consider one of the most general and difficult classes, mixed-integer dynamic optimization (MIDO), where part of the decision variables are discrete (binary or integer) [10].

Though there is a fair amount of different global optimization methods, metaheuristics have stood out as a competitive alternative for complex problems. Metaheuristics employ learning strategies to build the solution progress and find, in an efficient way, near-optimal solutions [28]. They consist of an iterative generation process that guides a heuristic through the connection of different ideas for exploration (global search) and exploitation (local search) of the search spaces.

Though it is granted that the use of metaheuristics often allows a significant reduction of the computational complexity of the search process, select and apply an appropriate metaheuristic for a given problem in an efficient way is a challenging exercise. First, it is difficult to know in advance which metaheuristic will be the most suitable for solving the problem at hand. Once selected, a metaheuristic algorithm started from different initial solutions will explore different regions of the solution space and return different optimized solutions. Moreover, using the same metaheuristic algorithm with different configuration settings will give different results, and there will even be large variations in performance between different instances of the same problem. Furthermore, metaheuristics remain time consuming for very hard complex problems. High performance computing (HPC) may stand for an effective means to overcome these issues.

In this work we propose a multimethod hyperheuristic that is suitable for the solution of very complex mixed-integer nonlinear programming (MINLP) problems. In our proposal, multiple different global search algorithms are executed concurrently and cooperate through the exchange of information. In [21] we have explored this idea in nonlinear programming (NLP) problems, demonstrating that, if we can devote a significant amount of resources to the search, an adaptive multimethod would achieve a more effective exploration of the search space. In [20] we have preliminary extended this idea to deal with

MINLP problems through the inclusion of an efficient local solver for this class of problems, changes to the self-adaptation mechanism to avoid premature stagnation of the convergence in this kind of problems, and the addition of new mechanisms to ensure diversity while keeping parallel cooperation. However, the preliminary evaluation performed in [20] already pointed out several further directions to be explored, that have been considered in this extended work, whose main contributions are:

- (1) a shared memory parallelization of the cost function evaluations using OpenMP, maintaining a loosely-coupled coarse-grained parallelization of the search diversification and the cooperation among the different methods using MPI. This release of the multimethod hyperheuristic for MINLP problems was pointed out as future work in [20]. The hybrid MPI+OpenMP implementation improves the scalability of the approach allowing to balance the computational resources between diversity and intensity in the search.
- (2) new adjustments in the specific mechanisms used to avoid premature convergence in the case of MINLP problems, in order to fix some of the problems that arose in the preliminary evaluation. The main one is the setting of the parameters used to trigger reconfiguration requests when a metaheuristic stagnates, which are now defined by the user. The values of these parameters were hard-coded in previous implementations of the method, preventing them from adapting to different metaheuristics and problems.
- (3) an exhaustive assess of the proposed implementation, comparing it with other single-method and non-cooperative parallel approaches. We are interested in applying this framework to the reverse engineering problem in computational systems biology, which yields very large MIDO-MINLP problems. Thus, in this work we have exhaustively evaluated the proposed multimethod using very challenging benchmarks in the field of cell signaling, that plays a key role in many application domains, such as the development of novel therapies in complex diseases such as cancer.
- (4) an evaluation in different infrastructures, from local clusters to supercomputers and public clouds. We believe that the results obtained could be particularly appealing for those interested in the potential of HPC infrastructures in general, and cloud-based platforms in particular, for developing metaheuristic methods in global optimization problems.

The organization of the paper is as follows. Section 2 presents a brief review of related work. Section 3 describes in depth the proposed multimethod approach. Section 4 covers an introduction to the reverse engineering of cell signaling phenomena, an explanation of the framework used in this work, and the description of the three challenging case studies used as benchmarks. An exhaustive performance evaluation of the multimethod proposed, attaining diversity in the different islands by means of two different metaheuristics and different configuration parameters, is reported in Section 5. Finally, conclusions and future work are detailed in Section 6.

2 Related Work

This section covers related work in the areas where this paper attempts to make contributions: on the one hand, in the development of multimethod hyperheuristics and, on the other hand, in the parallelization itself and its assessment in different HPC infrastructures, in particular, comparing the cloud with traditional architectures.

A large number of metaheuristics has been developed to address many different types of global optimization problems. However, it is difficult to predict which of the existing algorithms will be the most appropriate for a specific problem. The use of more than one algorithm becomes an attractive option in this context. A multimethod hyperheuristic consists of multiple algorithms that run and cooperate to achieve a better solution than the solution any of them could find separately. This has been shown up in different fields, including parallel cooperative search methods [12], memetic algorithms [11], algorithm ensembles [53], algorithm portfolios [37], and hyperheuristics [8]. A self-adaptive Differential Evolution (DE) algorithm presented in [39], that makes use of DE learning strategies that are weighted based on the algorithm success, can be considered one of the first attempts in this direction. Another example is the heterogeneous cooperative algorithm presented in [32], that employs different evolutionary algorithms to update the subpopulations in a cooperative algorithm framework. In [5] the use of a parameterized metaheuristic facilitates experimentation with different metaheuristics and hybridation/combinations to adapt them to the particular problem at hand. Also, the multimethod hyperheuristic algorithm that makes use of a number of common metaheuristics presented in [22] has obtained promising results in terms of solution quality and algorithm robustness. Finally, among all the examples found in the literature, AMALGAM-SO [50] is probably the most recognized one. It consists of a population-based genetic adaptive method that updates the allocation of algorithms during the optimization run. In this paper we describe and exhaustively evaluate a multimethod approach that incorporates a self-adaptive mechanism, so that the cooperation between different methods guides the behavior of the algorithm, dynamically changing the configuration parameters during the execution.

Current trends in computer architecture, with the proliferation of multicore systems and the incursion of the accelerators (mainly GPUs, but also others such as FPGAs), make the use of parallel computing increasingly common. Additionally, the cost/performance ratio of HPC architectures is continuously decreasing and, with the advent of cloud computing, it is now more feasible to access a large amount of distributed resources. Thus, it is not surprising that the parallelization of very time-consuming methods, such as metaheuristics, has received much attention. Many different parallel solutions have been proposed in the literature. Most of them are parallel implementations based on traditional parallel programming interfaces, such as MPI or OpenMP, executed in traditional parallel infrastructures, such as local clusters or supercomputers. A nice review can be found in [1] and, more recently, in [2]. In this paper we

propose a parallel implementation using an hybrid approach with MPI and OpenMP. This parallelization allows to achieve a better trade-off between diversity and intensity in the searches, which is specially effective in very hard problems, like those that are the objective of this work.

Recently, some attention has also been focused on cloud-based parallel metaheuristics [30,26,48,40,27,45,46] using MapReduce [13] and Spark [54]. The performance of MPI applications in the cloud has also received attention lately, though only few studies have been carried out in the specific field of parallel metaheuristics [19,43]. Most of the studies found in the literature use classical MPI benchmarks to compare the performance of MPI on public cloud platforms [15,31,33]. More recently, an extensive analysis to detect the more critical issues and bottlenecks of HPC applications in the cloud has been carried out in [18]. These works conclude that clouds were not designed for running tightly-coupled HPC workloads, such as MPI applications. The virtualization overhead, together with the lack of high-bandwidth and low-latency networks, degrade the performance of such applications in the cloud. However, the parallel implementation presented in this work is based on an island model, which drastically reduces the inter-process communications. This fact, together with the asynchronous communication protocol proposed and the star topology used in the cooperation stage, lead to a loosely-coupled parallel application, more suitable for the cloud, as it will be demonstrated in Section 5.

3 Self-adaptive cooperative multimethod

Metaheuristic algorithms are extraordinarily multifaceted. By varying the representation, the operators, the population size, the initialization, the selection mechanism, and other parameters, they lead to a varied collection of search procedures. Thus, it is not easy to know in advance which of the numerous existing choices will be the most suitable for solving a given problem. The self-adaptive cooperative multimethod (saCMM) aims to help overcome this issue by serving much like a swiss army knife: a handy set of tools that can be used to address a variety of problems. However, this versatility comes at a cost. For each application there may be probably a better metaheuristic devised specifically for it. Still, if you don't know exactly which metaheuristic will face your problem, the flexible nature of the multimethod provides with the ability to address effectively a wide variety of problems.

The multimethod strategy pursues the convergence improvement boosting the diversification in the search by means of an island-model approach. The global execution is divided into processes (islands) where single methods (different metaheuristics) are executed isolated, while sparse information exchanges are performed to link different islands, thus modifying the systemic properties of individual searches.

The proposed saCMM method is graphically illustrated in Figure 1. A loosely-coupled coarse-grained parallelization of the search diversification is

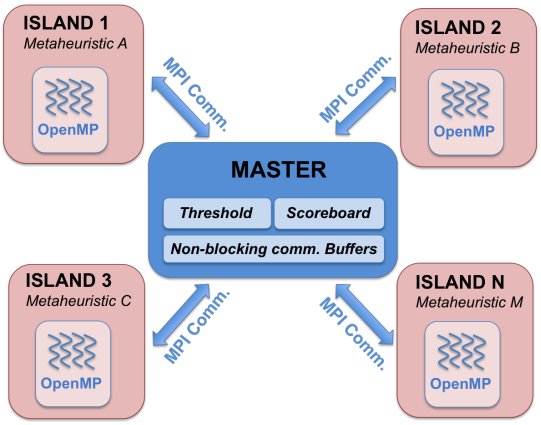


Fig. 1: Representation of the saCMM method: a master-slave approach with star topology and asynchronous communication protocol, that is implemented using MPI+OpenMP.

proposed, following a master-slave approach in which each slave is an island executing a different metaheuristic. The interconnection topology used is a star, in which the master process is in the center of the star and the islands exchange information through the master. The master process plays a key role in the saCMM method. First, it is in charge of the cooperation between islands. In order to achieve a more effective cooperation, in this proposal the exchange of information is driven by the quality of the solutions obtained, and not by the elapsed time as in most of the traditional implementations of island models. Second, it monitors the long-term behavior of the islands and adapts the configuration parameters during the execution to improve the search of the parallel hyperheuristic.

To efficiently handle MINLP problems, the *Mixed-Integer Sequential Quadratic Programming* (MISQP) [17,16] solver is used in each metaheuristic as a local solver. A different frequency is used to trigger it on each island in order to further increase diversity in the search. Also, to further improve the performance of the method, a hybrid MPI+OpenMP implementation is provided. The message passing paradigm, using MPI, is applied to implement an asynchronous communication protocol between the islands and the master process. The OpenMP framework is used to achieve a shared memory parallelization of the cost function evaluations within each island.

A flowchart of the steps followed by the master can be seen in Figure 2. In essence, the master is a loop waiting to receive communications from the islands. Note that the communication protocol implemented uses MPI asynchronous non-blocking operations in order to avoid stalling processes unnecessarily. The received communications at the master can be of three types: (1) a promising solution, (2) a reconfiguration request, or (3) a termination request.

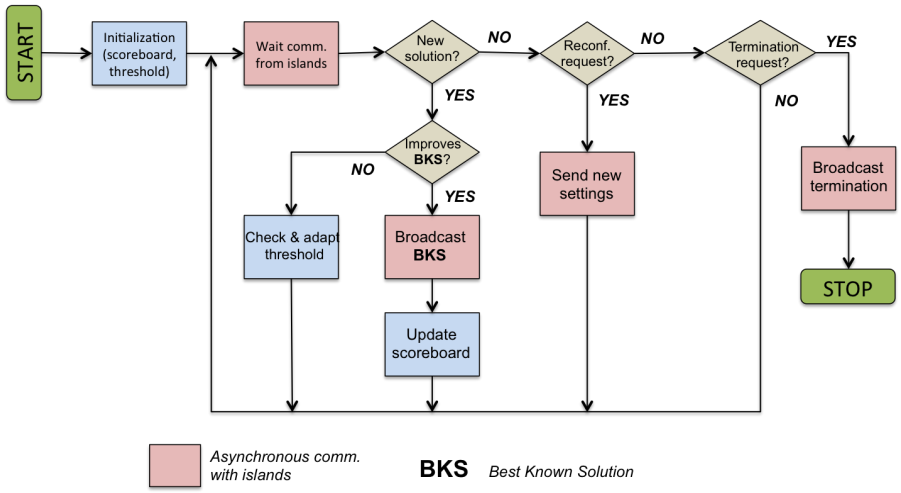


Fig. 2: Flowchart of the master process of saCMM.

Each island sends the solution that locally improve the *best known solution* at that moment to the master. However, it is well known that an excess of cooperation between the islands can have a counterproductive effect in the parallel search, since it could harm the diversity and lead to what is called *premature convergence* to local optima, from which it could be hard to get out. This effect, common in many problems, has already arisen in the evaluation of the previous version of saCMM for MINLP problems [20]: a promising incoming solution in an island acts as an attractor for the members of the population, bringing them fast to the vicinity of this new value. To avoid this effect several new mechanisms have been added to this version of saCMM, both in the master and in the islands. For instance, the master includes a threshold to decide which of the solutions coming from the islands deserves to be considered as a cooperative solution and be distributed to the rest. In the case of MINLP problems, the threshold starts with a cautiously chosen high value, so that most of the solutions that arrive are discarded and only those that significantly improve the global best known solution so far are propagated. However, as the search progresses, the ratio of improvement of the new solutions also decreases, especially as they approach the global optimum. Thus, the threshold is adapted dynamically during the search, decreasing according to the ratio of improvement of the solutions.

The master uses a scoreboard to keep up-to-date information on the punctuation of each island based on the success of its search, thus, the scoreboard is updated when an island cooperates with a promising solution. The information maintained in the scoreboard is used when a reconfiguration request arrives from an island. An island requests a reconfiguration when it is not able to improve its local best known solution but it frequently receives promising cooperative solutions from other searches. That is, each island decides whether

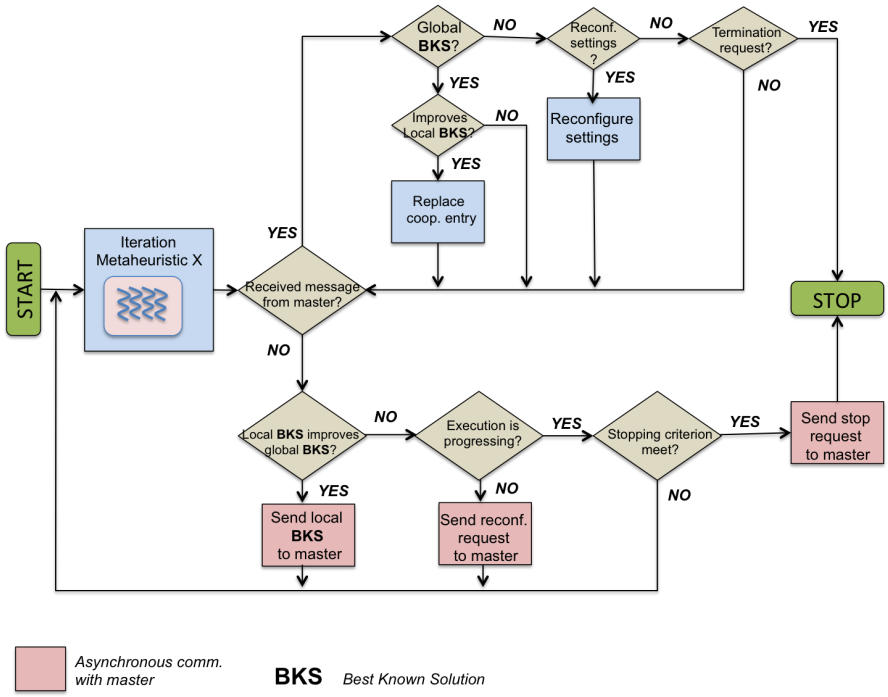


Fig. 3: Flowchart of the islands of saCMM.

it is stagnated in the search, and requests the master to send new configuration parameters to start a different metaheuristic. The master will send the parameters of one of the searches in the upper part of the scoreboard ordered by punctuation.

Finally, when the master receives a termination request from an island is because the stopping criterion was met, so the master sends a termination message to the rest of the islands to finish the execution.

Figure 3 summarizes the steps followed by the islands. First and foremost, the islands execute a sequential metaheuristic (single method) using certain configuration parameters that are different for each of them to improve diversification in the search. Additionally, since the evaluation of the cost function is one of the most time-consuming tasks in this kind of problems, an intra-node parallelization is used in each island to perform the evaluations of the cost function in parallel. Algorithm 1 shows the pseudocode of the evaluation step that has been implemented as an OMP parallel loop. Note that a dynamic schedule is used so that the evaluations are handed out to threads as they complete their previously assigned evaluation.

Since the communication protocol implemented is asynchronous, during the progress of the search in each island, a reception memory buffer keeps the messages that arrive from the master to process them afterwards. Once

Algorithm 1: Parallel evaluation of the cost function

```
neval = 0;
$$ parallel do (dynamic schedule, private(eval,newsol,i)
  reduction(+:neval));
for  i=1 to numSolutions do
    newsol = solutions(:,i);
    eval = f_eval(newsol);
    neval ++;
end
$$ end parallel do;
```

one iteration of the metaheuristic is finished, it is followed by a reception step where each island inspects its reception buffer. There are three types of messages that can be received from the master: (1) a promising solution, (2) a reconfiguration message, and (3) a termination request.

First, when an island receives a new promising solution that improves the local best known solution, it replaces the so-called *cooperative entry* of the island population. Note that only one entry in the population is labeled as the cooperative entry, so that promising solutions from other islands can not overwhelm the island population. This allows to preserve the diversity in the islands, to try to avoid the premature convergence in MINLP problems. Second, the island may also receive a message with the new configuration settings it had previously requested. Note that the request for a reconfiguration is also an asynchronous non-blocking operation, thus, the execution proceeds until the message with the new configuration settings arrives. Again, in the case of difficult problems, and in particular in the case of MINLP problems, islands easily stagnate due to a quick loss of diversity when the members of the population start to converge to the same local optimum. Thus, an additional mechanism is used to inject diversity into those situations, besides changing the configuration settings. Most of the members of the population (all of them except the best known solution and another random entry) are reinitialized randomly to inject more randomness into the reconfigured islands. Third, a termination message may also arrive to request that the execution on the island be stopped.

Once the reception step is over, each island sends its best known solution to the master as a promising solution, only if it improves the global best known solution. Then, an adaptive step is executed to decide if the island search is progressing or not in order to request new configuration settings from the master. The reconfiguration request is based on both the number of evaluations performed since the last time the island sent a promising solution to the master (i.e. whether the island is performing an unproductive effort), and the trade-off between received and sent solutions (i.e. whether the other islands are being more successful). The parameters used by the islands to decide if they are progressing were hard-coded in the previous implementations of saCMM. However, preliminary results showed that, based on the behavior of differ-

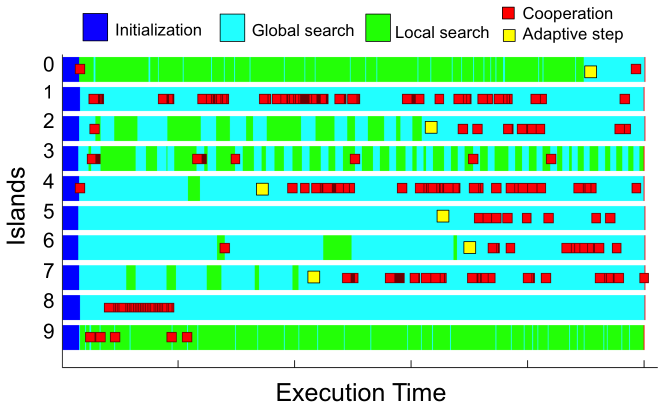


Fig. 4: Gantt diagram showing the execution of different islands and the cooperation between them. Light blue areas represent global search steps, green areas represent local search steps, red dots represent asynchronous cooperation and yellow dots represent reconfiguration steps.

ent searches (i.e. different metaheuristics, different configurations of the same metaheuristic, or even the same configuration applied to different problems), the trigger of a reconfiguration request is problem-dependent. Thus, in the new version of the saCMM implementation, the trigger of the reconfiguration request can be defined by the user. Finally, the stopping criterion is checked, starting a new iteration of the metaheuristic if it is not fulfilled.

In order to illustrate how the saCMM works, Figure 4 shows a Gantt diagram of an execution using 10 islands. Each island starts a metaheuristic with a different trade-off between global and local searches, ranging from aggressive ones, which perform frequent local searches, to conservative ones, which perform local searches only sporadically. It can be observed that conservative configurations, with few local searches, outperform aggressive configurations. During the execution, unsuccessful islands detect stagnation and request a reconfiguration from the master. Then, the master sends the configuration settings of the most promising islands to those that made the request. That is the case of islands 0, 2, 4, 5, 6 and 7 that, at a certain point during their execution, request a reconfiguration and change from an aggressive to a conservative configuration. It can be seen in the figure that, after the reconfiguration, they begin to cooperate again. Note that, since the most promising configuration can change during the progress of the search, not all the islands are allowed to be reconfigured. In the example, the islands 8 (a conservative one) and 9 (an aggressive one) are not reconfigured. This prevents the loss of diversity in the group of metaheuristics being explored, and avoids that all the islands end up executing the same metaheuristic in the long term.

4 Cell signaling: framework and case studies

Reverse engineering combines mathematical modeling with experimental data in order to infer, analyze and understand the functional and regulatory procedures that control the behavior of biological systems. Reverse engineering of cell signaling constitutes a major area in systems biology [3]. In this work, we have considered three case studies involved in cell signaling processes.

Most of these models have to cope with many trials, such as dealing with dynamic behavior [49]. The logic-based ordinary differential equations (ODE) framework has been found particularly useful in this kind of problems [52]. In particular, we have applied the mixed-integer global optimization approach proposed in [24]. In this approach, the problem of identifying the logic gates is formulated as a simultaneous model selection and parameter identification problem, which corresponds to a mixed-integer dynamic optimization (MIDO) problem.

The general MIDO problem is usually formulated as finding the set of discrete, time-dependent (stimuli or controls) and time-independent parameters, to optimize a predefined cost function, while satisfying a set of dynamic and algebraic constraints. The MIDO problem is commonly formulated in mathematical form as:

Find $\mathbf{u}(t)$, $\mathbf{i}(t)$, $\mathbf{p}$ and t_f so as to minimize (or maximize):

$$J = G_{t_f}(\mathbf{x}, \mathbf{u}, \mathbf{i}, \mathbf{p}, t_f) + \int_{t_0}^{t_f} F(\mathbf{x}(t), \mathbf{u}(t), \mathbf{i}(t), \mathbf{p}, t)dt \tag{1}$$

subject to:

$$\mathbf{f}(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{u}(t), \mathbf{i}(t), \mathbf{p}, t) = 0, \quad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{2}$$

$$\mathbf{g}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{i}(t), \mathbf{p}, t) \leq 0, \quad l = \overline{1, m_e + m_i} \tag{3}$$

$$\mathbf{u}_L \leq \mathbf{u}(t) \leq \mathbf{u}_U, \tag{4}$$

$$\mathbf{i}_L \leq \mathbf{i}(t) \leq \mathbf{i}_U, \tag{5}$$

$$\mathbf{p}_L \leq \mathbf{p} \leq \mathbf{p}_U, \tag{6}$$

where $\mathbf{x}(t) \in X \subseteq R^{n_x}$ is the vector of state variables, $\dot{\mathbf{x}}(t)$ is its time derivative, $\mathbf{u}(t) \in U \subseteq R^{n_u}$ is the vector of real valued control variables, $\mathbf{i}(t) \in I \in Z^{n_i}$ is the vector of integer control variables, $\mathbf{p} \in P \subseteq R^{n_p}$ is the vector of time-independent parameters, t_f is the final time of the process, m_e, m_i represent the number of equality and inequality constraints, $\mathbf{f}$ is the set of ordinary differential equations describing the dynamics of the system (plus the corresponding initial conditions), $\mathbf{g}$ is the set of state constraints (path, pointwise and final time constraints), and $\mathbf{u}_L$, $\mathbf{i}_L$, $\mathbf{p}_L$, $\mathbf{u}_U$, $\mathbf{i}_U$, $\mathbf{p}_U$ correspond to the lower and upper bounds for the control variables and the time-independent parameters.

In this work, the above formulation is used to identify at the same time the underlying network topology, its regulatory structure, the time-dependent controls and the time-invariant model parameters, compatible with existing

experimental data. From the variety of methods for the numerical solution of dynamic optimization problems, we have followed the direct approach described in [24] that consists of two steps. First, the original MIDO problem is translated into a mixed-integer nonlinear programming (MINLP) problem. Then, a numerical solution step is applied, in which the solution of the MINLP problem is actually obtained by means of the multimethod proposed.

The next subsections briefly describe the three case studies used in this work as benchmarks to assess the performance of the saCMM multimethod for solving complex MINLP problems.

4.1 Synthetic signaling pathway (SSP)

The first case study considered in this work is a synthetic signaling pathway described in [29]. This benchmark, from now on SSP, considers a dynamic model composed of 26 ordinary differential equations and 86 continuous parameters. Despite being a synthetic problem, it is considered a reasonable representation of a signaling pathway. In fact, it was originally used to illustrate the capabilities and limitations of different formalisms related to logic-based models.

An expanded version of this model, containing every possible AND/OR logic gate given the initial graph structure, is obtained using the methodology described in [41]. The final optimization problem contains 120 parameters, being 86 continuous and 34 binaries. The model and experimental setup were implemented using AMIGO [6], and then exported to C code in order to apply the saCMM method presented here.

4.2 Signaling pathway applied to liver cancer (HepG2)

The reverse engineering of a logic-based ODE model using liver cancer data (a subset of the data generated by [4]) is used as the second case study, called HepG2 for short. The dataset consists of phosphorylation measurements of proteins (used as proxy of their activation), from a hepatocellular carcinoma cell line (HepG2) at 0, 30 and 180 minutes after perturbation.

The R version of CellNOpt [47] is used to preprocess the network. It is compressed to remove as many non-observable/non-controllable species as possible. All gates that are consistent with the network are generated, adding hyperedges (AND gates) from all pair of inputs (the OR gates are implicit). The expanded network has 109 hyperedges and 135 continuous parameters. A parser that generates a C model file and Matlab scripts compatible with the AMIGO toolbox [6] is used to obtain the logic-based ODE model.

Finally, the optimization problem to solve contains a total of 244 parameters, being 135 continuous and 109 binaries. The time-series dataset is quite rich from the point of view of information content: it includes 64 perturbations comprising 7 ligands and 7 small-molecule inhibitors. The ligands were chosen to activate inflammation and proliferation pathways, and the inhibitors to

block the activity of specific proteins. Data are normalized by rescaling so that they are between 0 and 1. The model has a total of 25 states, 16 of them corresponding to observed species. The initial conditions for the other 9 species have to be estimated. To avoid increasing the problem size and multi-modality unnecessarily, the estimated initial conditions were assumed the same for each experiment reported in Section 5.

4.3 Breast cancer network inference challenge (HPN-DREAM)

As the third case study we consider an extremely difficult problem which has been made publicly available in the context of the DREAM challenges [1]. Presented as collaborative competitions, the DREAM challenges brings forth a forum to crowdsource fundamental problems in systems biology and medicine[42]. The inference of signaling networks [38, 25] is an example.

The problem used here as a benchmark is obtained from the HPN-DREAM breast cancer challenge. The data-set incorporates time-series acquired under eight extracellular stimuli, under four different protein inhibitors and a control, in four breast cancer cell lines [25]. Overall, the problem contains a total of 828 decision variables (690 continuous and 138 binaries). Furthermore, the HPN-DREAM is an extraordinary defiant problem even from a computational perspective, with a vast execution time and an unknown final target value.

5 Experimental results

The aim of this section is to evaluate, as a proof of concept, the proposed self-adaptive cooperative multimethod (saCMM) to solve complex MINLP problems. Currently, two metaheuristics are implemented in the saCMM method proposed, the Differential Evolution (DE) [44], with the enhancements described in [34], and the enhanced Scatter Search (eSS) [14], using the implementation outlined in [35]. However, in saCMM each island performs one of these metaheuristics with different configuration parameters, which leads to completely different searches. At the start of every execution of saCMM, half of the islands perform DE and the other half eSS with different initial populations and configuration parameters. For the experiments reported in this section, Table 1 shows the initial configuration of the different islands. Some parameters are common to both metaheuristics: the factor used to calculate the reference set size or population size (*size factor*) for each island, the parameter to balance the weight between quality and diversity when choosing a candidate as the initial point for the local search (*balance*), and the parameter that sets how often (in number of iterations) a local solver is performed (*local.n*). Other parameters are specific of the DE method: the mutation factor (F), the crossover constant (CR), and the mutation strategy (*MSt.*). As

[1] www.dreamchallenges.org

Table 1: Initial configuration of the different islands for the experiments in this section. Note that the islands could be reconfigured during runtime.

				DE islands		
#Island	**size factor** (*)	**balance**	**local.n**	**F**	**CR**	**MSt.**
1	1	0.0	1	0.9	0.8	best/2
2	3	0.0	1000	0.7	0.9	best/1
3	5	0.25	10	0.5	0.9	best/2
4	10	0.5	20	0.9	0.9	best/1
5	15	0.25	100	0.7	0.7	best/2
6	1	0.25	1000	-	-	-
7	3	0.25	15	-	-	-
8	5	0.25	7	-	-	-
9	10	0.0	1000	-	-	-
10	15	0.0	1	-	-	-

(*) For DE islands the population size is computed as $50 * factor$, while for eSS islands the reference set size is calculated by means of the polinomial: $x^2 - x - factor * Nparameters$

the execution proceeds, the islands reconfigurate themselves according to the execution progress and the successful configurations.

Using the three different case studies described in Section 4, different experiments have been orchestrated to assess: (1) the impact of the cooperation and self-adaptive mechanisms included in saCMM versus a single-method and other parallel non-cooperative approaches, (2) the impact of the diversification comparing the performance of the proposal with other parallel cooperative approaches, (3) the scalability of the coarse-grained parallelization, (4) the performance of the hybrid approach and its impact in the scalability, (5) the performance obtained in different infrastructures and its significance, and (6) the impact of the proposal on the solution of very complex MIDO/MINLP problems.

It should be noted that, for a given model structure in a reverse engineering problem, datasets of different sizes may result in global optimization problems of different difficulty due to the different associated ill-conditionings. In this respect, note that more important than the size of the dataset is the information it contains. Further, how flexible the dynamics are also can play a major role in the resulting multimodality of the problem. For instance, systems exhibiting oscillatory dynamics can result in extremely hard problems even for small number of parameters. For an in-depth discussion about the impact of these issues in reverse engineering problems in system biology the reader is referred to [49,24].

Most of the experiments in this section were carried out in a local cluster named Pluton. Each compute node in Pluton consists of 2x8 cores Intel Xeon E5-2660 @2.60 GHz processors and 64 GB of RAM.

Assessing the performance of metaheuristics in general, and their parallel implementations in particular, is not an easy task mainly due to their variability and stochastic nature. Comparing the performance of different proposals is

therefore even more complex. Experiments were designed to perform analyses from both a horizontal view [23], that is, assessing the performance by measuring the time needed to reach a given target value (*value-to-reach*, or VTR from now on), and from a vertical view, that is, evaluating the performance for a predefined effort that for these experiments is the maximum execution time. There are also some experiments that combine the two stopping criteria, a VTR and a maximum execution time. Due to the substantial dispersion of the results, 20 executions have been performed for each experiment.

5.1 Performance of the coarse-grained parallelization

5.1.1 Impact of the cooperation and self-adaptive mechanisms

In these experiments, the saCMM method presented has been compared with the following different methods in order to illustrate the impact of the cooperation and self-adaptive mechanisms in the proposal success:

- *Single metaheuristics (Mx)*: the same different metaheuristics that are used in each island of the saCMM method were executed in isolation and sequentially. This experiment allows to compare the performance of the single sequential methods with the impact of the parallelization, in which multiple executions can be done concurrently.
- An *embarrassingly parallel non-cooperative multimethod (MM)*: that consists of the same different metaheuristics that are used in the saCMM method being executed in parallel without cooperation between them. The reported result would be the best value achieved by any of them. This experiment measures the impact of the cooperation and self-adaptation with respect to a parallel solution without such mechanisms.
- A *sequential saCMM (seq_saCMM)*: that is, a saCMM execution with multiple searches in different islands but using only one core. Note that seq_saCMM is not a serial implementation of the saCMM method, but the saCMM method executed on a single core. The purpose of this experiment is to show that the multimethod is also suitable in a sequential environment to improve the solution of a problem when it is not known a priori which of the different metaheuristics is the most appropriate.

Figure 5 shows the results of the best value obtained (*fbest*) in these experiments for both the SSP and HepG2 minimization problems. For SSP, a maximum execution time of 1 hour was used and for HepG2, due to its larger computational complexity, the maximum execution time was set to 3 hours. In both cases a configuration with 10 islands was used. For each experiment, the best value achieved is plotted and *violin plots* are used to also show the statistical distribution of the results. Every *single* metaheuristic (that is, any of the islands of the saCMM isolately) is represented by the letter 'M' followed by a number from 1 to 10 (*Mx*). It can be seen in the figure that the dispersion is much larger in any of these methods than in the last three. The sequential saCMM (*seq_saCMM*) improves the results of the single metaheuristics in

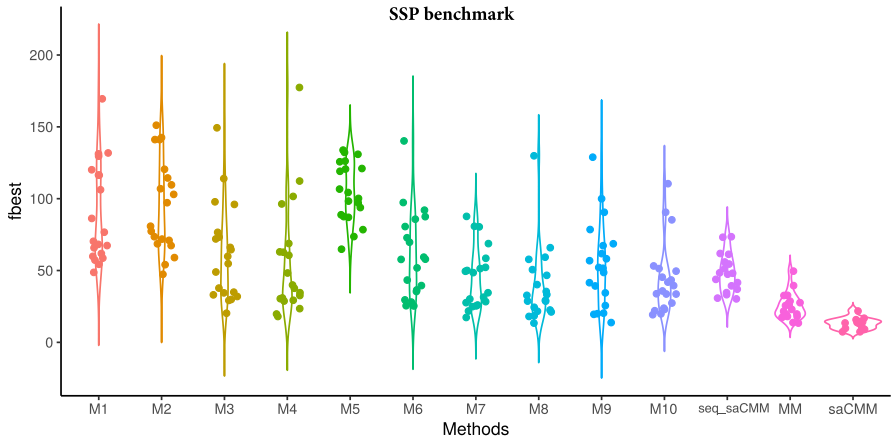
terms of dispersion; however, as expected, the parallel versions are the ones that obtain the best results: the non-cooperative multimethod (*MM*) obtains encouraging results using an embarrassingly parallel solution, and the saCMM outperforms them by including cooperation and adaptation. In these experiments seq_saCMM achieves an average best value of 50±16 for SSP in 1 hour and 88±7 for HepG2 in 3 hours, while saCMM shows a speedup of 7.6x and 12.1x respectively to reach the same values. Likewise, MM achieves an average best value of 25±9 for SSP in 1 hour and 73±12 for HepG2 in 3 hours, and saCMM, compared to MM, achieves a speedup of 2.9x and 3.9x respectively. Thus, saCMM demonstrates its performance impact when compared to the same parallel islands running isolated without cooperation between them or with self-adaptation during run-time.

5.1.2 Impact of the diversification

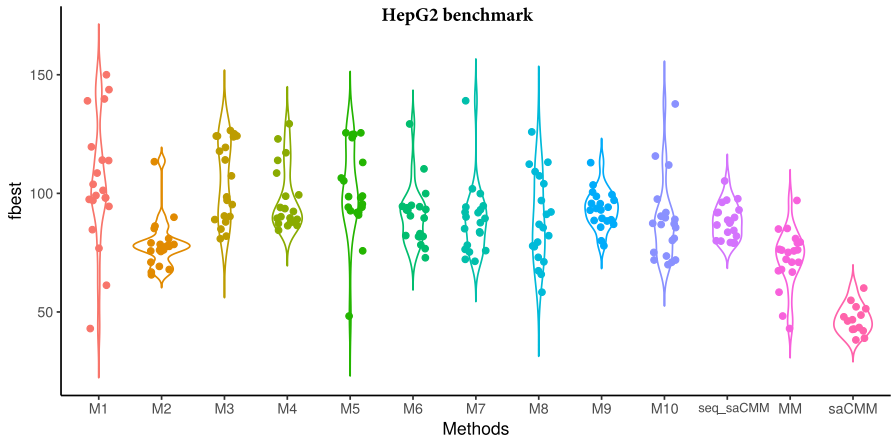
Previous results illustrate the improvement achieved through the cooperation between different islands. It should be noted that different metaheuristics can have different performance as the execution progresses, that is, a method can be very promising at the beginning of execution and stagnate before reaching the global optimum, and conversely, it can start slowly and end up being the most promising. This fact is illustrated in figure 6 that represents the progression of the search at different times for saCMM using the SSP benchmark. As it can be seen, after one hour of execution, most of the best values obtained in different runs of the experiment were obtained by eSS islands, while after 3 hours, and specially after 6 hours, most of the best values were achieved by DE islands.

In order to evaluate the impact of diversification on the islands, a series of experiments have been carried out to compare the saCMM method with other cooperative and adaptive parallel methods. These methods include the same enhancements as saCMM but don't perform different metaheuristics on each island. Two parallel methods were considered: (1) saCDE, that only uses the DE metaheuristic, and (2) saCeSS, that only uses the eSS metaheuristic. Table 2 summarises the results obtained for the SSP and HepG2 benchmarks both for a horizontal view, using as stopping criterion a specific VTR, and for a vertical view, using as a stopping criterion a maximum time. In the table the average and median execution time for the horizontal view and the average and median best value for the vertical view, are reported. Since the dispersion of the results is also significant to assess the robustness of the method, Figure 7 also shows the violin plots of the results of the horizontal view. These results show that, for the SSP benchmark, the saCeSS method outperforms saCDE. However, for the HepG2 benchmark, saCDE outperforms saCeSS. For both benchmarks, either saCMM performs close to the best of the other two methods or outperforms them both thanks to the cooperation between the single-method metaheuristics.

To test the statistical significance of the results, the Wilcoxon Rank-Sum test [51] was used, which verifies the null hypothesis that two populations



(a) Results for SSP using as stopping criterion a predefined effort of 1 hour.



(b) Results for HepG2 using as stopping criterion a predefined effort of 3 hours.

Fig. 5: Violin plots comparing the dispersion of the best values obtained by 10 single metaheuristics (Mx), the saCMM method executed in 1 core (*seq_saCMM*), and the parallel embarrassingly non-cooperative method (MM) and saCMM, both executed in 10 cores.

have the same continuous distribution. Table 3 shows the results of the test for the best values of the vertical view reported in Table 2 with a confidence level of 0.95. As it can be seen, for the SSP benchmark the p-value is smaller than the significance (0.05) when comparing the saCeSS and saCDE methods. Thus, saCeSS clearly outperforms saCDE for this benchmark. As expected, saCMM also achieves a smaller p-value than the significance when compared to saCDE. But when compared to saCeSS, the p-value obtained is greater than the significance, because for this benchmark the saCMM and saCeSS methods

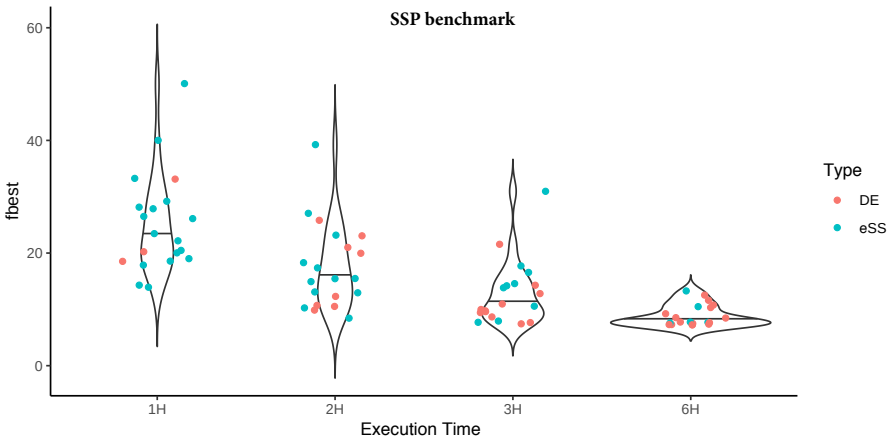
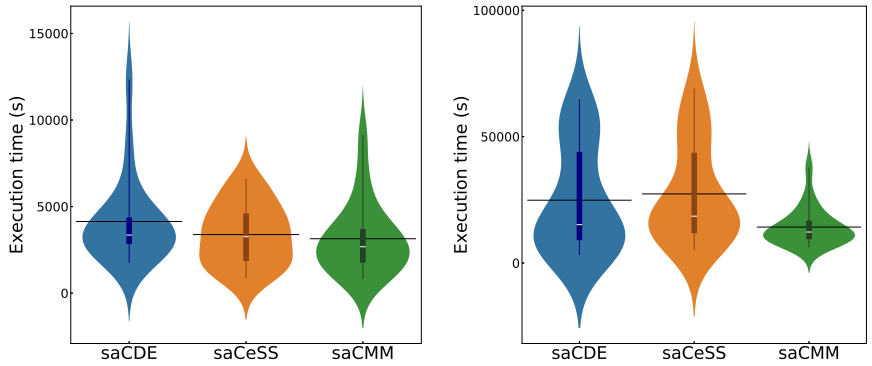


Fig. 6: Violin plots comparing the dispersion of the best values obtained by the saCMM method during the progression of the search. The dots represent the best value for each experiment out of a total of 20, colored according to the metaheuristic used on the island that obtained the best solution

Table 2: Performance of the proposed self-adaptive multimethod compared with other parallel but single-method approaches. Results were obtained using 10 islands for both horizontal and vertical views.

			Horizontal view	
	method	**VTR**	**avg. time (s)**	**median time (s)**
SSP	saCDE	15	4138±2466	3355
	saCeSS	15	3384±1765	3362
	saCMM	15	3143±2072	2679
HepG2	saCDE	45	24844±21233	15172
	saCeSS	45	27336±19483	18526
	saCMM	45	14216±7507	12260
			Vertical view	
	method	**effort (s)**	**avg. fbest**	**median fbest**
SSP	saCDE	1500	38.27±12.44	35.58
	saCeSS	1500	20.42±3.50	20.77
	saCMM	1500	19.92±5.38	18.47
HepG2	saCDE	10800	57.81±9.85	56.11
	saCeSS	10800	56.51±9.06	57.92
	saCMM	10800	49.87±8.81	47.64

have similar performances. The same could be said about the saCDE and saCeSS performances with the HepG2 benchmark, the p-value is greater than the significance and the null hypothesis is accepted. However, the p-value is less than the significance when comparing saCMM with the saCDE and saCeSS methods, because saCMM outperforms both in this benchmark.



(a) SSP benchmark (b) HepG2 benchmark

Fig. 7: Violin plots of the execution times of the horizontal view reported in Table 2.

Table 3: Wilcoxon rank-sum test with a significance level α=0.05.

benchmark	**tests**	W	p**-value**
SSP	saCeSS vs saCDE	391	2.563E-07
	saCMM vs saCDE	385	6.015E-07
	saCMM vs saCeSS	231	0.409
HepG2	saCeSS vs saCDE	215	0.6949
	saCMM vs saCDE	293	0.01234
	saCMM vs saCeSS	283	0.02564

In light of these results, we can conclude that when a single-method metaheuristic performs better than others for a problem, saCMM will perform similarly to that metaheuristic, because the adaptive procedure will shove most of the islands to the most successful configuration. Besides, if several single-method metaheuristics show similar performances or are successful at different times of the search, the multimethod will have a better performance than them due to cooperation.

5.1.3 Scalability of the coarse-grained parallelization

All previous experiments were performed using 10 islands plus the master process. To evaluate the scalability of the saCMM method, new experiments were conducted with 10, 20 and 40 islands. Increasing the number of islands increases diversification in the search and, therefore, allows us to study scalability. Table 4 summarizes the results of these experiments for the SSP and HepG2 benchmarks for both horizontal and vertical views.

Table 4: Performance of the proposed self-adaptive multimethod with respect to the number of islands.

	Horizontal view				
	#Islands	**VTR**	**avg. time(s)**	**median(s)**	**sp(%eff)**
SSP	10	15	3143±2072	2679	-
	20	15	2611±2337	1821	1.20 (60%)
	40	15	1897±1253	1518	1.66 (42%)
HepG2	10	45	14216±7507	12260	-
	20	45	13803±9764	11087	1.03 (51%)
	40	45	8911±3099	9243	1.60 (40%)
	Vertical view				
	#Islands	**effort(s)**	**avg. fbest**	**median fbest**	**#hits**
SSP	10	1500	19.92±5.38	18.47	3
	20	1500	18.81±4.71	18.32	7
	40	1500	17.46±3.67	16.03	9
HepG2	10	10800	49.87±8.81	47.64	8
	20	10800	48.78±8.63	46.78	8
	40	10800	44.82±7.81	43.20	13

For the horizontal view, the table includes the speedup (*sp*) for 20 and 40 islands calculated based on the execution on 10 islands. It can be seen that, for both benchmarks, the performance of the method improves with the number of islands. Note that the original workload is not distributed among the islands, since the population is not distributed among them. Therefore, the speedup achieved is due to the impact of diversification in the search. The more islands, the more regions of the solution space are explored using different metaheuristics. However, the efficiency (*%eff*) achieved, also reported in this table, shows a moderate outcome when the number of islands increases. Hence, a new proposal enters in discussion: the implementation of hybrid parallelization using MPI and OpenMP. This will help balancing resources between the cost function evaluation and the diversification in the search, as shown in next section.

For the vertical view, the table includes the number of executions that achieve the VTR before the maximum allowed time expires (#hits). Note that, although the best value achieved does not improve substantially with the number of islands, the number of hits improves significantly.

5.2 Performance of the hybrid implementation

The performance of the hybrid implementation has been evaluated by means of two set of experiments. First, the impact of the shared memory parallelization was evaluated using 10 islands (MPI processes) and 1, 2 and 4 OpenMP threads on each island. The results of these experiments for the SSP and HepG2 benchmarks are reported in Table 5 for both horizontal and vertical views. As expected, for the same number of islands, increasing the number of threads in each island improves the execution time. However, the efficiency is

Table 5: Performance of the shared memory parallelization in the proposed self-adaptive multimethod.

			Horizontal view		
	config.	**VTR**	**avg. time(s)**	**median time(s)**	**sp(%eff)**
SSP	10x1	15	3143±2072	2679	-
	10x2	15	1959±1007	1772	1.60 (80%)
	10x4	15	1349±889	963	2.33 (58%)
HepG2	10x1	45	14216±7507	12260	-
	10x2	45	10932±4744	9811	1.30 (65%)
	10x4	45	10326±5909	8429	1.38 (35%)
			Vertical view		
	conf.	**effort(s)**	**avg. fbest**	**median fbest**	**#hits**
SSP	10x1	1500	19.92±5.38	18.47	3
	10x2	1500	18.54±5.09	18.93	6
	10x4	1500	13.30±5.11	12.03	14
HepG2	10x1	10800	49.87±8.81	47.63	8
	10x2	10800	49.09±10.35	44.51	10
	10x4	10800	43.74±7.01	43.79	12

below ideal. For instance, for the horizontal view of the SSP benchmark, the speedup of the 10x2 configuration with respect to the 10x1 configuration is 1.60, while the ideal would be 2. This is because in the shared memory parallelization, only the cost function is evaluated in parallel, and although it is the most time-consuming operation, it is not the only one. Nevertheless, for the same number of islands, the results of the shared memory parallelization outperform those of the coarse-grained parallelization in Table 4.

Then, the impact of the balance between diversification (number of islands) and intensification (number of threads) in the hybrid approach was evaluated using different configurations of number of islands and threads per island (40x1, 20x2 and 10x4) in a fixed number of 40 cores. The results of these experiments for the SSP and HepG2 benchmarks are summarized in Table 6 for both horizontal and vertical views. For the horizontal view, the speedup was calculated with respect to the 40x1 configuration and efficiency is not provided, since the number of cores is the same for all experiments and there is no ideal speedup with which to compare. As expected, a good trade-off between diversification and intensification is essential to efficiently exploit the hybrid implementation, however, it is difficult to know in advance which configuration benefits the problem at hand. For instance, Table 6 shows that intensification benefits the SSP benchmark while diversification benefits the HepG2 benchmark.

5.3 Performance in different infrastructures

Traditionally, HPC applications are executed on special-purpose hardware, often located in specific facilities, such as supercomputing centers, and managed by staff with specialized skills. With Cloud Computing gaining popularity, ef-

Table 6: Performance of saCMM with different configurations of islands and threads per island using a fixed number of 40 cores.

			Horizontal view		
	config.	**VTR**	**avg. time (s)**	**median time (s)**	**speedup**
SSP	40x1	15	1897±1253	1518	-
	20x2	15	1716±742	1760	1.11
	10x4	15	1349±889	963	1.41
HepG2	40x1	45	8911±3099	9243	-
	20x2	45	9764±5181	7960	0.91
	10x4	45	10326±5909	8429	0.86
			Vertical view		
	config.	**effort (s)**	**avg. fbest**	**median fbest**	**#hits**
SSP	40x1	1500	17.46±3.67	16.03	9
	20x2	1500	16.82±5.51	18.12	6
	10x4	1500	13.30±5.11	12.03	14
HepG2	40x1	10800	44.82±7.81	43.20	13
	20x2	10800	45.39±9.84	42.95	12
	10x4	10800	43.74±7.01	43.79	12

fortless access to a large amount of distributed resources has become more feasible. However, its adoption by the HPC community is still limited. Firstly, because traditional parallel programming models and tools in the HPC community are not easily applicable to cloud platforms, and the learning curve to understand the different available architectures and run-time environments dampens from adopting it as an alternative. Secondly, because clouds also pose important challenges regarding performance aspects. As pointed out in section 2, there have been many research works evaluating the promise of cloud platforms for HPC computing, most of them concluding that cloud-based clusters need significant improvement in performance to be competitive for HPC applications. Thus, we were interested in assessing the performance of the saCMM method in different infrastructures, including the Cloud, to shed light on their potential for this kind of problems.

Two other platforms, besides our local cluster, have been used for these experiments: (1) the FinisTerrae-II supercomputer, located at the Galician Supercomputing Center (CESGA) [9], composed of 306 nodes interconnected via InfiniBand FDR 56Gb/s. Each node has two Intel Xeon E5-2680 v3 @2.5GHz processors with 12 cores and 128GB of RAM per processor; and (2) the Microsoft Azure public cloud, in which two different virtual clusters have been used, one with low-cost instances and the other with HPC instances, to compare their performance and performance/cost ratio. The same virtual clusters were used for all the experiments to avoid differences in the results due to differences in the underlying hardware or the variability in the latency between the nodes. Both clusters were deployed in the North Europe region, using A3 instances (4 cores with 7GB of RAM on Intel Xeon E5-2673 @2.40GHz) with canonical Ubuntu Server for the low-cost cluster, and A9 compute-intensive instances (16 cores with 112GB of RAM on Intel Xeon E5-2670 @2.6GHz) with HPC-CentOS for the HPC cluster.

Table 7: Horizontal analysis of the SSP benchmark with VTR=15 on different platforms.

Infrastructure	**mean time(s)**	**median(s)**	**price/hour**	**avg. cost/exp.**
Pluton	3143±2072	2679	–	–
FinisTerrae-II	1727±658	1539	0,100 €/core	0,53 €
Azure low-cost	18322±7582	18308	0,155 €/inst.	2,37 €
Azure HPC	3543±1830	3383	1,742 €/inst.	1,72 €

A series of experiments were carried out on the platforms described above using the SSP benchmark. The results for the horizontal analysis with a stopping criterion of VTR=15 are reported in Table 7. It also includes the price per hour on each platform and the average cost per experiment. As expected, the FinisTerrae-II supercomputer outperforms the other platforms, obtaining execution times between 1.8 and 2.0 times better than the local cluster Pluton or the Azure HPC virtual cluster. Also, it's not surprising that the results of the Azure low-cost virtual cluster are the worst of all. Both the underlying processor and network explain these results. Besides, it can be seen that the execution times of the Azure HPC virtual cluster are competitive with those of the local cluster.

It would be of notable interest to accomplish a cost analysis comparing the cost of using resources in the cloud with the cost of using resources in a local cluster, however, this is a very difficult task [55]. The actual cost of a local cluster is related to its utilization level. For a local cluster maintained over several years, such as Pluton, the higher the utilization level, the lower the effective cost rate. Moreover, when estimating the cost of a local cluster, not only the acquisition but also the operational expenses have to be taken into account. Therefore, an accurate estimate of the price per hour of our local cluster was unfeasible. The price per hour of the FinisTerrae-II supercomputer was obtained from the CESGA website [9], which states that this cost should be taken as a guideline and may vary if other services, beyond the computational nodes, are needed or technical support is required. Note also that, although the cost of these experiments in FinisTerrae-II is very low, it is not always easy to obtain access to this kind of infrastructures. In Azure, in November 2018 in the North Europe region, the price of an A3 instance was 0.155€/hour and the price of an A9 instance was 1.742€/hour. We needed three A3 instances and only one A9 instance to run the experiments. The average cost per experiment is shown in Table 7, in which it can be seen that A9 instances are cost-effective for this kind of experiments.

To further assess the scalability of the proposed multimethod on the Azure cloud, additional experiments were performed increasing the number of islands (cores) used. A vertical analysis was carried out using a maximum execution time of 1500s as stopping criterion. Figure 8 shows the bean plots of the distribution of the best solutions obtained in the local and virtual clusters. The results of the local cluster are only slightly better than those of the virtual clus-

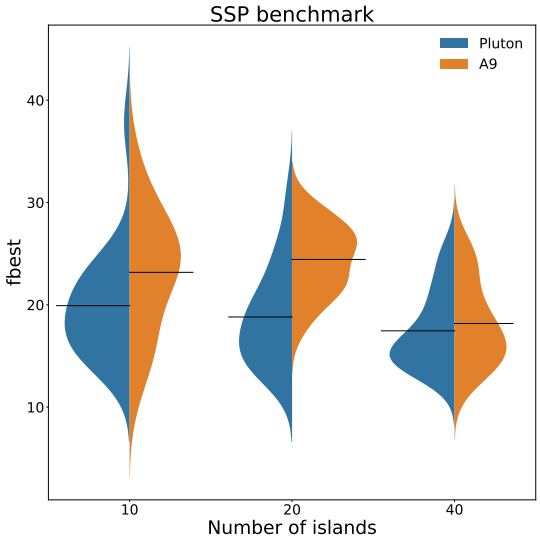


Fig. 8: Bean plots of the best solutions achieved in a vertical analysis of the SSP benchmark made by increasing the number of islands with a maximum execution time of 1500s

ter. As pointed out in section 2, the lack of high-bandwidth, low-latency networks and the virtualization overhead have a large effect on the performance of HPC applications in the cloud, especially MPI applications. However, in our experience, the use of HPC instances currently provided by leading cloud providers, together with the development of suitable parallel implementations such as the island model of the proposed multimethod, drastically reduce interprocess communications leading to competitive solutions. In fact, in our opinion, the cloud *pay-as-you-go* model can potentially be a cost-effective and timely solution for the needs of many HPC users.

5.4 Performance with the HPN-DREAM case study

The last experiments carried out in this work were devoted to evaluate the performance of the proposed multimethod approach in the solution of very difficult and complex problems. For this purpose, the HPN-DREAM benchmark described in Section 4, was used. It is an extremely challenging problem from the computational point of view, which has an unknown target value.

As the usage policy of our local cluster limits the duration of each job to 3 days, that was the maximum time used as stopping criterion for the experiments. Figure 9 shows the convergence curves for experiments using 10 and 40

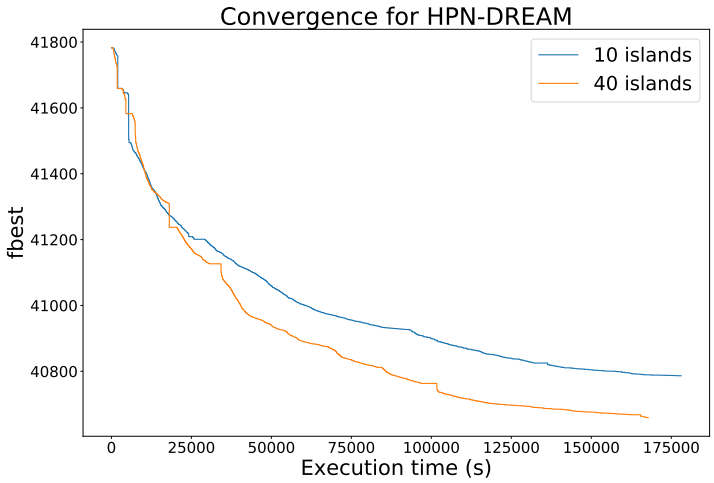


Fig. 9: Convergence curves for the HPN-DREAM benchmark, using 10 and 40 islands.

islands. As it can be seen, the performance of the proposed saCMM method to solve this challenging problem improves with the number of islands. Moreover, the performance of saCeSS, one of the cooperative single-method metaheuristics used in Section 5.1.2 to compare the performance of the proposed multimethod, was also evaluated in [36] using the HPN-DREAM benchmark. The results of saCMM are much better than those obtained in [36] for saCeSS. For instance, when using 40 islands, saCMM obtains in 2 days a solution of the same quality for which saCeSS needs more that 5 days.

6 Conclusions

In this paper, we propose and evaluate a hybrid implementation (using MPI + OpenMP) of the self-adaptive cooperative multimethod (saCMM) extended with some additional mechanisms to handle MINLP problems. The extensive evaluation carried out includes experiments to assess both the coarse-grained parallelization (MPI) and the shared memory parallelization (OpenMP), as well as the impact on the performance of the balance between diversification and intensification. The proposal shows a good performance in comparison with sequential single methods and self-adaptive parallel single methods in cell signaling case studies from the computational systems biology domain. These results demonstrate that the proposed multimethod can be successfully used to reverse engineer dynamic models of biological pathways.

Additionally, the proposal has been evaluated using different infrastructures: a supercomputer, a local cluster, and two different virtual clusters on

the Microsoft Azure cloud. We believe that these results can be specially useful for those interested in the potential of the cloud computing model for their HPC applications.

The code of the multimethod proposed, that includes the extension to handle MINLP problems, is publicly available at: `https://bitbucket.org/pabloarguale/sacmm-library-minlp`

Acknowledgements This research received financial support from the Spanish Government through the projects DPI2017-82896-C2-2-R and TIN2016-75845-P (AEI/FEDER, UE), and from the Galician Government under the Consolidation Program of Competitive Research Units (Network Ref. R2016/045 and Project Ref. ED431C 2017/04), all of them co-funded by FEDER funds of the EU. We also acknowledge Microsoft Research for being awarded with a sponsored Azure account, and CESGA for the access to their facilities.

References

1. Alba, E.: Parallel Metaheuristics: A New Class of Algorithms. Wiley-Interscience, NJ, USA (2005)
2. Alba, E., Luque, G., Nesmachnow, S.: Parallel metaheuristics: Recent advances and new trends. International Transactions in Operational Research **20**(1), 1–48 (2013)
3. Aldridge, B.B., Burke, J.M., Lauffenburger, D.A., Sorger, P.K.: Physicochemical modelling of cell signalling pathways. Nature cell biology **8**(11), 1195–1203 (2006)
4. Alexopoulos, L.G., Saez-Rodriguez, J., Cosgrove, B.D., Lauffenburger, D.A., Sorger, P.K.: Networks inferred from biochemical data reveal profound differences in toll-like receptor and inflammatory signaling between normal and transformed hepatocytes. Molecular & Cell Proteomics **9**(9), 1849–1865 (2010)
5. Almeida, F., Giménez, D., López-Espín, J.J.: A parameterized shared-memory scheme for parameterized metaheuristics. The Journal of Supercomputing **58**(3), 292–301 (2011)
6. Balsa-Canto, E., Banga, J.R.: AMIGO, a toolbox for advanced model identification in systems biology using global optimization. Bioinformatics **27**(16), 2311–2313 (2011)
7. Banga, J.R.: Optimization in computational systems biology. BMC Systems Biology **2**(1), 47 (2008)
8. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: A survey of the state of the art. Journal of the Operational Research Society **64**(12), 1695–1724 (2013)
9. CESGA: Centro de supercomputacin de galicia. URL `http://www.cesga.es`
10. Chachuat, B., Singer, A., Barton, P.: Global methods for dynamic optimization and mixed-integer dynamic optimization. Industrial & Engineering Chemistry Research **45**(25), 8373–8392 (2006)
11. Chen, X., Ong, Y.S., Lim, M.H., Tan, K.C.: A multi-facet survey on memetic computation. IEEE Transactions on Evolutionary Computation **15**(5), 591–607 (2011)
12. Crainic, T.G.: Parallel meta-heuristic and cooperative search. Technical report, CIRRELT-2017-58, Universite du Quebec a Montreal (2017)
13. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. The 6th USENIX Symposium on Operating Systems Design and Implementation (2004)
14. Egea, J.A., Balsa-Canto, E., García, M.S.G., Banga, J.R.: Dynamic optimization of nonlinear processes with an enhanced scatter search method. Industrial & Engineering Chemistry Research **48**(9), 4388–4401 (2009)
15. Evangelinos, C., Hill, C.: Cloud computing for parallel scientific HPC applications: Feasibility of running coupled atmosphere-ocean climate models on amazon's EC2. In: 1st Workshop on Cloud Computing and its Applications (CCA'08), pp. 1–6 (2008)
16. Exler, O., Lehmann, T., Schittkowski, K.: A comparative study of sqp-type algorithms for nonlinear and nonconvex mixed-integer optimization. Mathematical Programming Computation **4**(4), 383–412 (2012)

17. Exler, O., Schittkowski, K.: A trust region sqp algorithm for mixed-integer nonlinear programming. Optimization Letters **1**(3), 269–280 (2007)
18. Expósito, R.R., Taboada, G.L., Ramos, S., Touriño, J., Doallo, R.: Performance analysis of HPC applications in the cloud. Future Generation Computer Systems **29**(1), 218–229 (2013)
19. González, P., Pardo, X.C., Penas, D.R., Teijeiro, D., Banga, J.R., Doallo, R.: Using the cloud for parameter estimation problems: comparing spark vs mpi with a case-study. The 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (2017)
20. González, P., Penas, D.R., Pardo, X.C., Banga, J.R., Doallo, R.: Multimethod optimization for reverse engineering of complex biological networks. In: Proceedings of the 6th International Workshop on Parallelism in Bioinformatics, PBio 2018, pp. 11–18 (2018)
21. González, P., Penas, D.R., Pardo, X.C., Banga, J.R., Doallo, R.: Multimethod optimization in the cloud: A case study in systems biology modelling. Concurrency and Computation: Practice and Experience **30**(12) (2018)
22. Grobler, J., Engelbrecht, A.P., Kendall, G., Yadavalli, V.: Alternative hyper-heuristic strategies for multi-method global optimization. The 2010 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8. IEEE (2010)
23. Hansen, N., Auger, A., Finck, S., Ros, R.: Real-parameter black-box optimization benchmarking 2009: Experimental setup. Tech. Rep. RR-6828, INRIA (2009)
24. Henriques, D., Rocha, M., Saez-Rodriguez, J., Banga, J.R.: Reverse engineering of logic-based differential equation models using a mixed-integer dynamic optimization approach. Bioinformatics **31**(18), 2999–3007 (2015)
25. Hill, S.M., Heiser, L.M., Cokelaer, T., Unger, M., Nesser, N.K., Carlin, D.E., Zhang, Y., Sokolov, A., Paull, E.O., Wong, C.K., et al.: Inferring causal molecular networks: empirical assessment through a community-based effort. Nature methods **13**(4), 310–318 (2016)
26. Jin, C., Vecchiola, C., Buyya, R.: MRPGA: an extension of MapReduce for parallelizing genetic algorithms. The 2008 IEEE Fourth International Conference on eScience, pp. 214–221. IEEE (2008)
27. Lee, W.P., Hsiao, Y.T., Hwang, W.C.: Designing a parallel evolutionary algorithm for inferring gene networks on the cloud computing environment. BMC systems biology **8**(1), 5 (2014)
28. Luke, S.: Essentials of metaheuristics, vol. 113. Lulu Raleigh (2009)
29. MacNamara, A., Terfve, C., Henriques, D., Bernabé, B.P., Saez-Rodriguez, J.: State–time spectrum of signal transduction logic models. Physical Biology **9**(4), 045003 (2012)
30. McNabb, A.W., Monson, C.K., Seppi, K.D.: Parallel PSO using MapReduce. The 2007 IEEE Congress on Evolutionary Computation (CEC), pp. 7–14. IEEE (2007)
31. Napper, J., Bientinesi, P.: Can cloud computing reach the top500? In: Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop, pp. 17–20. ACM (2009)
32. Olorunda, O., Engelbrecht, A.P.: An analysis of heterogeneous cooperative algorithms. The 2009 IEEE Congress on Evolutionary Computation (CEC), pp. 1562–1569. IEEE (2009)
33. Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T., Epema, D.: An early performance analysis of cloud computing services for scientific computing. Delft University of Technology, Tech. Rep (2008)
34. Penas, D., Banga, J., González, P., Doallo, R.: Enhanced parallel differential evolution algorithm for problems in computational systems biology. Applied Soft Computing **33**, 86–99 (2015)
35. Penas, D., González, P., Egea, J.A., Doallo, R., Banga, J.: Parameter estimation in large-scale systems biology models: a parallel and self-adaptive cooperative strategy. BMC Bioinformatics **18**(1), 52 (2017)
36. Penas, D.R., Henriques, D., González, P., Doallo, R., Saez-Rodriguez, J., Banga, J.R.: A parallel metaheuristic for large mixed-integer dynamic optimization problems, with applications in computational biology. Plos One **12**(8) (2017)
37. Peng, F., Tang, K., Chen, G., Yao, X.: Population-based algorithm portfolios for numerical optimization. IEEE Transactions on Evolutionary Computation **14**(5), 782–800 (2010)

38. Prill, R.J., Saez-Rodriguez, J., Alexopoulos, L.G., Sorger, P.K., Stolovitzky, G.: Crowdsourcing network inference: The dream predictive signaling network challenge. Science Signaling **4**(189), mr7 (2011)
39. Qin, A.K., Suganthan, P.N.: Self-adaptive differential evolution algorithm for numerical optimization. pp. 1785–1791. IEEE (2005)
40. Radenski, A.: Distributed simulated annealing with MapReduce. Applications of Evolutionary Computation, pp. 466–476. Springer (2012)
41. Saez-Rodriguez, J., Alexopoulos, L.G., Epperlein, J., Samaga, R., Lauffenburger, D.A., Klamt, S., Sorger, P.K.: Discrete logic modelling as a means to link protein signalling networks with functional analysis of mammalian signal transduction. Molecular Systems Biology **5**, 331 (2009)
42. Saez-Rodriguez, J., Costello, J.C., Friend, S.H., Kellen, M.R., Mangravite, L., Meyer, P., Norman, T., Stolovitzky, G.: Crowdsourcing biomedical research: leveraging communities as innovation engines. Nature Reviews Genetics **17**(8), 470–486 (2016)
43. Salto, C., Minetti, G., Alba, E., Luque, G.: Developing genetic algorithms using different mapreduce frameworks: Mpi vs. hadoop. In: Conference of the Spanish Association for Artificial Intelligence, pp. 262–272. Springer (2018)
44. Storn, R., Price, K.: Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization **11**(4), 341–359 (1997)
45. Teijeiro, D., Pardo, X.C., González, P., Banga, J.R., Doallo, R.: Implementing parallel differential evolution on Spark. Applications of Evolutionary Computation. Lecture Notes in Computer Science, Vol. 9598, pp. 75–90. Springer (2016)
46. Teijeiro, D., Pardo, X.C., González, P., Banga, J.R., Doallo, R.: Towards cloud-based parallel metaheuristics: A case study in computational biology with differential evolution and spark. International Journal of High Performance Computing Applications (2016). DOI 10.1177/1094342016679011
47. Terfve, C., Cokelaer, T., Henriques, D., MacNamara, A., Goncalves, E., Morris, M.K., van Iersel, M., Lauffenburger, D.A., Saez-Rodriguez, J.: CellNOptR: a flexible toolkit to train protein signaling networks to data using multiple logic formalisms. BMC Systems Biology **6**(1), 133 (2012)
48. Verma, A., Llora, X., Goldberg, D.E., Campbell, R.H.: Scaling genetic algorithms using MapReduce. The Ninth International Conference on Intelligent Systems Design and Applications, ISDA'09, pp. 13–18. IEEE (2009)
49. Villaverde, A.F., Banga, J.R.: Reverse engineering and identification in systems biology: strategies, perspectives and challenges. Journal of the Royal Society Interface **11**(91), 20130505 (2014)
50. Vrugt, J.A., Robinson, B.A., Hyman, J.M.: Self-adaptive multimethod search for global optimization in real-parameter spaces. IEEE Transactions on Evolutionary Computation **13**(2), 243–259 (2009)
51. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics bulletin **1**(6), 80–83 (1945)
52. Wittmann, D.M., Krumsiek, J., Saez-Rodriguez, J., Lauffenburger, D.A., Klamt, S., Theis, F.J.: Transforming boolean models to continuous models: methodology and application to t-cell receptor signaling. BMC Systems Biology **3**(1), 98 (2009)
53. Yang, P., Hwa Yang, Y., B Zhou, B., Y Zomaya, A.: A review of ensemble methods in bioinformatics. Current Bioinformatics **5**(4), 296–308 (2010)
54. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. The 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI (2012)
55. Zhai, Y., Liu, M., Zhai, J., Ma, X., Chen, W.: Cloud versus in-house cluster: evaluating amazon cluster compute instances for running mpi applications. In: SC'11: State of the Practice Reports, p. 11. ACM (2011)