Contents lists available at ScienceDirect

Expert Systems With Applications

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

Parallel ant colony optimization for the training of cell signaling networks

Patricia González [a,*], Roberto Prado-Rodriguez [a], Attila Gábor [b], Julio Saez-Rodriguez [b], Julio R. Banga [c], Ramón Doallo [a]

[a] CITIC, Computer Architecture Group, University of A Coruña, Spain
[b] Institute for Computational Biomedicine, Faculty of Medicine, Heidelberg University, Germany
[c] Computational Biology Lab, MBG-CSIC, Spanish National Research Council, Spain

ARTICLE INFO

Keywords:
Cell signaling network
Metaheuristics
Ant colony optimization
High performance computing
MPI
OpenMP

ABSTRACT

Acquiring a functional comprehension of the deregulation of cell signaling networks in disease allows progress in the development of new therapies and drugs. Computational models are becoming increasingly popular as a systematic tool to analyze the functioning of complex biochemical networks, such as those involved in cell signaling. CellNOpt is a framework to build predictive logic-based models of signaling pathways by training a prior knowledge network to biochemical data obtained from perturbation experiments. This training can be formulated as an optimization problem that can be solved using metaheuristics. However, the genetic algorithm used so far in CellNOpt presents limitations in terms of execution time and quality of solutions when applied to large instances. Thus, in order to overcome those issues, in this paper we propose the use of a method based on ant colony optimization, adapted to the problem at hand and parallelized using a hybrid approach. The performance of this novel method is illustrated with several challenging benchmark problems in the study of new therapies for liver cancer.

1. Introduction

Mathematical optimization seeks the best solution inside a feasible search space with regard to some criterion (given by one or more objective functions). Optimization problems persistently appear in the real world, in as disparate areas as traditional scheduling and routing (Beldjilali, Benadda, & Sadouni, 2020), engineering design (Bojan-Dragos et al., 2021), or control systems (Pozna, Precup, Horvath, & Petriu, 2022; Precup, David, Petriu, Preitl, & Paul, 2011), but also in recent applications such as assessment of network traffic on the internet (Li, Chen, & Tang, 2020) or the prediction of people's behavioral intention (Tan, Ooi, Leong, & Lin, 2014). One of the fields which is currently fostering advances in optimization methods is computational systems biology (Banga, 2008). Within this area, discovering effective drugs in biomedicine and understanding how they work still lead to considerable challenges nowadays (Iorio et al., 2016). There are several model-based paradigms for identifying drugs that could be used to treat specific diseases. In this context, logic modeling is one of most useful approaches to understand signal transduction deregulation in disease and to characterize the mode of action of a drug (Traynard, Tobalina, Eduati, Calzone, & Saez-Rodriguez, 2017).

Studying how cells process signals through complex and dynamic networks, for a given context and an specific cell type, is essential to understand signaling in both physiological and disease conditions. A method that integrates static and non-context-specific knowledge about signaling networks with perturbation data was initially proposed in Saez-Rodriguez et al. (2009). By training prior knowledge networks (PKN) against experimental data, this method produces models that achieve larger predictive capacity. The model construction process is implemented by using a logic-based formalism where the relationships between species are described using logic gates that specify the state of each node given the state of its parents. The automatically generated models are subsequently trained to data. Therefore, they are useful to understand how signals are processed by cells and how they can be altered, and can be used to predict the effect of perturbations.

CellNOpt (Gjerga et al., 2020; Terfve et al., 2012) is a framework that implements this method and several extensions, allowing for building predictive logic models of signaling networks by training networks derived from prior knowledge to experimental data. It proceeds by first compressing the protein signaling network to remove species that are neither observable nor controllable, and then expands it to a scaffold model representing an overlay of all the possible logic gates supported by the network. This model is then trained against experimental data by minimizing an objective function that quantifies the difference

* Correspondence to: Facultade de Informática, University of A Coruña, Campus de Elviña s/n, 15701 A Coruña, Spain.
E-mail addresses: patricia.gonzalez@udc.es (P. González), roberto.prado@udc.es (R. Prado-Rodriguez), attila.gabor@uni-heidelberg.de (A. Gábor), julio.saez@uni-heidelberg.de (J. Saez-Rodriguez), j.r.banga@csic.es (J.R. Banga), ramon.doallo@udc.es (R. Doallo).

https://doi.org/10.1016/j.eswa.2022.118199
Received 23 February 2022; Received in revised form 11 June 2022; Accepted 15 July 2022
Available online 20 July 2022

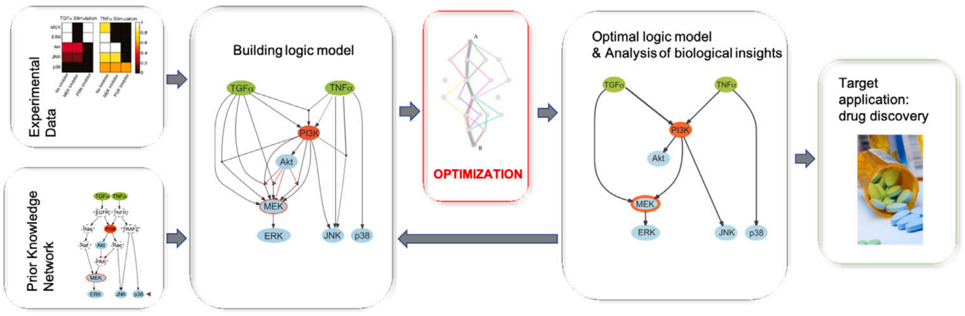
Experimental Data
Prior Knowledge Network
Building logic model
OPTIMIZATION
Optimal logic model & Analysis of biological insights
Target application: drug discovery

Fig. 1. Workflow of CellNOpt framework highlighting the optimization phase in the process.

between the data and the predictions obtained by simulation. Finally, the calibrated models are used to predict new results from which new biological information can be obtained. A typical workflow of the CellNOpt framework is shown in Fig. 1, highlighting the optimization step, the subject of our study here.

These minimization problems usually require global optimization methods (Banga, 2008). Even though deterministic global methods can be used for certain problem instances, the associated computational effort to ensure global optimality can be extremely large, making them impractical for realistic problems. As an alternative, stochastic methods (and metaheuristics, in particular) can provide near-global solutions in reasonable computation times.

Two important issues when training cell signaling networks using metaheuristics are that they may not guarantee reaching a global optima, and that they are usually not poised to report a complete family of feasible models. Both of them are key issues needed to provide precise insights about the mechanisms of signal transduction. Thus, other approaches based on Integer Linear Programming (ILP) (Mitsos et al., 2009; Sharan & Karp, 2012) and Answer Set Programming (ASP) (Guziolowski et al., 2013; Videla et al., 2012) have been applied to this class of problems, providing a proof of concept that a global optimum can be identified. However, these two methods suffer from a major issue, the superlinear increase of memory consumption with the size of the problem, which becomes a challenge for many application cases.

The two issues discussed above related to the use of metaheuristics can be addressed. First, to overcome the issue of obtaining the global optimum, for some metaheuristics, including various variants of the so-called Ant Colony Optimization (ACO), it is possible to demonstrate their convergence (Dorigo & Stutzle, 2002; Gutjahr, 2002) under certain circumstances. However, it is difficult to estimate the theoretical speed of convergence, and this is where parallel implementations can help, reducing the convergence time by speeding up computations, or even improving the convergence rate. Second, although standard metaheuristics stop when an optimal solution is found, they can be adapted to detect and report a family of high-quality feasible solutions.

However many large problems in cell signaling networks, and specifically in the case of model development leading to early drug discovery, cannot be solved with standard methods on sequential computers because, due to their complexity, they require excessive execution time or exceed the available memory. Therefore, high performance computing (HPC) infrastructures and HPC implementations of suitable solvers are particularly appealing, and even essential, to solve today's challenging problems. In this work, we propose a novel parallel implementation of an algorithm based on ACO, adapted to the problem of signaling networks in cells, and making use of a high-performance skeleton that allows us to develop a general approach that can be easily reused or extended to other metaheuristics or for other purposes. For the experimental assessment of the proposal, we have considered the training of several real case-studies of signaling networks in liver cancer using the CellNOpt framework.

The organization of this paper is as follows. Section 2 presents a brief review of related work. Section 3 describes the optimization problem. Section 4 presents an overview of the ACO metaheuristic and its adaptation to the training of signaling networks in cells. Section 5 describes the parallel implementation proposed. Its performance is evaluated in Section 6. Finally, Section 7 summarizes the main conclusions of this work.

2. Related work

The goal of systems biology is to generate new insights into complex biological systems by combining experimental data with mathematical models. Model parameter estimations of most biological processes are complex problems that may have multiple solutions (Villaverde & Banga, 2014). To efficiently solve this calibration problem, many research efforts have focused on developing metaheuristic methods that are capable of finding good solutions in reasonable computation times (Balsa-Canto, Banga, Egea, Fernandez-Villaverde, & de Hijas-Liste, 2012; Banga & Balsa-Canto, 2008; Gábor & Banga, 2015; Sun, Garibaldi, & Hodgman, 2012). Many examples using various metaheuristics can be found in the literature, such as simulating annealing (Perkins, Jaeger, Reinitz, & Glass, 2006), evolutionary strategies (Ji & Xu, 2006; Jostins & Jaeger, 2010), differential evolution (Da Ros et al., 2013; Villaverde & Banga, 2014; Zúñiga, Cruz, & García, 2014), scatter search (Egea, Balsa-Canto, García, & Banga, 2009; Egea, Martí, & Banga, 2010), particle swarm optimization (Palafox, Noman, & Iba, 2012; Tang, Chai, Wang, & Cao, 2020), among others. Also, many proposals exploit different parallelization strategies and infrastructures to solve these problems in competitive execution times (Adams et al., 2013; González et al., 2017; Lee, Hsiao, & Hwang, 2014; Penas, Banga, González and Doallo, 2015; Penas, González, Egea, Banga and Doallo, 2015; Penas et al., 2017; Teijeiro et al., 2017).

With regard to cell signaling, logic-based modeling is an approach that falls between the complexity and accuracy of differential equations on the one hand and data-driven regression approaches on the other. A review of the fundamentals of logic-based models of biochemical signaling networks can be found in Morris, Saez-Rodriguez, Sorger, and Lauffenburger (2010). This work illustrates the ability of integrating experimental and logic-based modeling to gain an in depth comprehension of the biological system. Using this formulation, the initial model can be trained to signaling data via an optimization approach to compute the values of model parameters that better fit the data (Morris, Saez-Rodriguez, Clarke, Sorger, & Lauffenburger, 2011; Saez-Rodriguez et al., 2009, 2011). A Genetic Algorithm (GA) was used in these works to prune the pathway by identifying and removing the contradicting reactions. However, the computational time needed for the optimization process points to a need for more reliable methods of training logic-based networks. In a recent study (Gjerga et al., 2020), the optimization problem was formulated as an Integer Linear Program (ILP) via CPLEX. In contrast to GA, the ILP formulation guaranteed global optimality and required a fraction of the CPU time needed by the GA. This work has focused on improving the optimization approach by means of a constructive metaheuristic, ACO, and the use of HPC techniques to overcome the problems raised with GA.

From the point of view of the parallel implementation, different strategies can be applied when dealing with metaheuristics in general, and the ACO in particular. Most parallel ACO proposals based on distributed memory implementations consists of the distributed of the ants of a single colony among the available processors (Craus & Rudeanu, 2004; Delisle, Krajecki, Gravel, & Gagné, 2001; Lv, Xia, & Qian, 2006; Talbi, Roux, Fonlupt, & Robillard, 2001; Tsutsui & Fujimoto, 2010). Synchronizations are carried out through a coordinator processor, which slows down the performance, since faster processes will be forced to wait for the slower ones. Some solutions that implement a coordinator process with asynchronous implementations can be also found in Bullnheimer, Kotsis, and Strauß (1998), Jie, CaiYun, and Zhong (2008) and Starzec, Starzec, Byrski, Turek, and Pietak (2020). This approaches demonstrate their superiority over synchronous ones. Other solutions follow a *divide-and-conquer* approach (Doerner, Hartl, Benkner, & Luckà, 2006; Mocholi, Jaen, & Canos, 2005), based on the decomposition of the problem and the search for partial solutions in different processes while a coordinator builds the complete solution afterwards. Shared memory implementations are also proposed in Delisle, Gravel, Krajecki, Gagné, and Price (2005), Delisle et al. (2001) and Hadian, Shahrivari, and Minaei-Bidgoli (2012), where small tasks are distributed among processes while a coordinator is in charge of carrying out the update of the global information and control the progress of the algorithm.

In this paper we adopt for the parallelization a multicolony model. Other authors have previously explored this model for the parallelization of the ACO algorithm, such as Chu, Roddick, and Pan (2004), Jie et al. (2008), Michel and Middendorf (1998, 1999), Piriyakumar and Levi (2002) and Twomey, Stützle, Dorigo, Manfrin, and Birattari (2010). One of the limitations of those works is the use of synchronous communications that limits the scalability of the proposal. Alternatives that use asynchronous communications and/or adjust the frequency of information exchanging can also be found in Chen, Sun, and Wang (2012), Ellabib, Calamai, and Basir (2007) and Ling Chen, Hai-Ying Sun, and Shu Wang (2008). All of these studies confirm that a tradeoff between exploration within each colony and cooperation through information exchanging is desirable to achieve accurate results and performance.

The authors of the present study have extensive experience in parallelizing different metaheuristics and using different problem-oriented strategies and frameworks in computational systems biology. Differential evolution has been parallelized using an asynchronous strategy in Penas, Banga, González, and Doallo (2014), which has been extended with heuristic improvements in Penas, Banga et al. (2015). An island-based version has been proposed in Teijeiro, Pardo, González, Banga and Doallo (2016) using Spark to be executed in distributed and cloud environments. An exhaustive evaluation of the Spark implementation versus a Mapreduce solution was presented in Teijeiro et al. (2016). Scatter Search has also been parallelized using a decentralized asynchronous strategy with a ring topology for message passing in Penas, González et al. (2015), and with a centralized strategy with a coordinator process that included self-adaptation in Penas, González, Egea, Doallo and Banga (2017). In Pardo, Argüeso-Alejandro, González, Banga, and Doallo (2020), a cloud-based implementation in Spark was proposed for the Scatter Search method. Furthermore, in González et al. (2019) a hybrid parallel multimethod prototype was proposed that exploit multiple computational resources to perform simultaneous searches using different metaheuristics.

Based on our previous experience, the implementation used in this work is committed to a hybrid model that combines attributes of several of the previous models, using both distributed memory and a multicolony model, as well as shared memory and a fine-grained model within each colony. Its most remarkable features are decentralization (a coordinator process is not needed to organize and control the algorithm) and the use of an asynchronous communication protocol between processes (maximizing the use of resources).

3. Problem statement

Fig. 1 shows a workflow of the CellNOpt framework. The methodology used in this framework is based on identifying a logic-based model describing the interactions of a set of species, starting from a prior knowledge network (PKN) and a set of experimental data. To build the logic-based model, the PKN is compressed to contain only the measured, perturbed, and inhibited nodes based on the experimental data, and then expanded to add other possible logical descriptions of gates connecting more than one input node to a single output node.

Training the cell signaling network model against experimental data is an optimization problem in a search space defined by the hypercube:

$$\sum = \{0,1\}^r \tag{1}$$

where candidate solutions are encoded in vectors $\mathbf{P} \in \sum$ and r is the number of gates in the scaffold model. Each gate is assigned an index i in vector $\mathbf{P}$, $i = 1,\dots,r$ such that $P_i = 1$ when the gate is included in the model and $P_i = 0$ when it is not.

The objective function for optimization is based on the mean squared error (MSE) deviation between the data and model (Θ_f), and a second term that penalized increasing model size (Θ_S). Thus, for a set of data containing n_E data points collected for m readouts at n time points under S experimental conditions, the training procedure consists in minimize:

$$\Theta(\mathbf{P}) = \Theta_f(\mathbf{P}) + \alpha \cdot \Theta_S(\mathbf{P}) \tag{2}$$

where

$$\Theta_f(\mathbf{P}) = \frac{1}{n_E}\sum_{k=1}^{S}\sum_{l=1}^{m}\sum_{t=1}^{n}(B^M_{k,l,t}(\mathbf{P}) - B^E_{k,l,t})^2 \tag{3}$$

and

$$\Theta_S(\mathbf{P}) = \frac{1}{v^S_e}\sum_{e=1}^{r} v_e P_e \tag{4}$$

such that $B^M_{k,l,t}(\mathbf{P}) \in \{0,1\}$ is the value predicted by computation of the model's logical steady state and $B^E_{k,l,t} \in [0,1]$ is the data value for readout l at time t under the kth experimental condition.

To compute the size penalty, each gate in a given solution $\mathbf{P}$ is weighted by the number of starting nodes v_e. By imposing a size penalty, unnecessary and redundant gates are not included in the final model. The size penalty is normalized to the total number of inputs across all gates $v^S_e = \sum_{e=1}^{r} v_e$ and weighted with the tunable parameter α.

Further details on how the steady state of the Boolean model is calculated can be found in Klamt, Saez-Rodriguez, Lindquist, Simeoni, and Gilles (2006) and in Saez-Rodriguez et al. (2009).

4. Ant colony optimization

Ant Colony Optimization (ACO) is a metaheuristic in which a *colony of ants* cooperate in finding good solutions to difficult optimization problems. An artificial ant in ACO is a stochastic procedure that gradually builds a solution by adding appropriate elements to the solution under construction. Thus, the ACO metaheuristic can be applied to combinatorial optimization problems for which a constructive heuristic can be defined, such as the training of signaling networks.

Algorithm 1 shows the basic pseudo-code of most ACO algorithms, with three main procedures: *ConstructAntsSolutions*, *UpdatePheromones*, and *DaemonActions*. *ConstructAntsSolutions* manages a colony of artificial ants that incrementally build solutions to the optimization problem by means of stochastic local decisions based on pheromone trails and heuristic information. Then, the *UpdatePheromones* procedure modifies the pheromone trails based both on the evaluation of the new solutions and on a pheromone evaporation mechanism. Finally, a *DaemonActions* procedure performs problem specific or centralized actions, which cannot be performed by single ants. Usually, this includes the application of local or global optimizations to further approximate the solution.

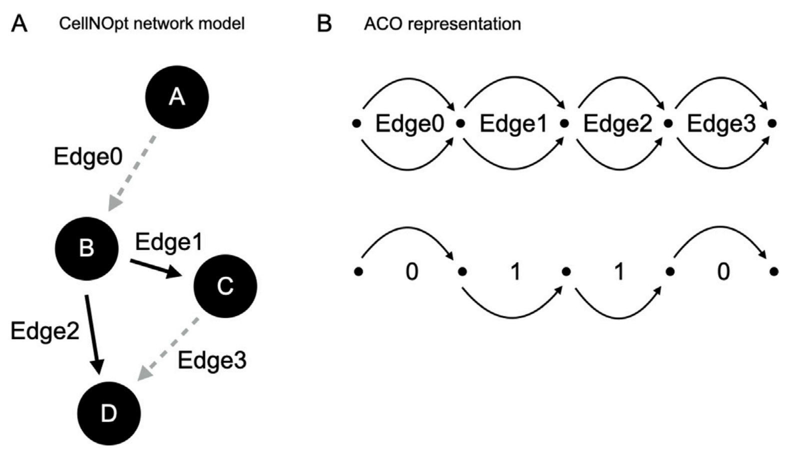

Fig. 2. Digraph representing the application of the ACO metaheuristic to the training of the compressed network in CellNOpt.

Algorithm 1: Pseudo-code of the Ant Colony Optimization metaheuristic

```
1 Set parameters
2 Initialize pheromone trails
3 while termination condition not met do
4     ConstructAntsSolutions
5     UpdatePheromones
6     DaemonActions %opt
```

A large number of variants of this basic ACO algorithm can be found in the literature. Among all, the $\mathcal{MAX}-\mathcal{MIN}$ Ant System (MMAS) (Stützle & Hoos, 2000) has been used in this problem. MMAS is chosen for being one of the most popular ACO variants, especially for its convenient efficiency, but also because it is one of the metaheuristics that guarantees convergence to a global optimum (Dorigo & Stutzle, 2002).

In the rest of this section the adaptation of the previous three basic procedures of the ACO metaheuristic, for its use in solving the training of cell signaling networks, are described in depth.

4.1. Ants solution

ACO is an algorithm widely used to solve graph-related problems. The artificial ants move from one node of the graph to another building solutions based on a probability function that depends on the amount of pheromones deposited in each arch and, eventually, on some heuristics that the user can add.

In the training of cell signaling network, each ant builds a solution that consists of a binary vector representing the inclusion (1) or exclusion (0) of an edge of the previous compressed network obtained by the CellNOpt framework. The digraph shown in Fig. 2 is used to represent the application of the ACO to this problem.

Each node of the graph has only two paths, P_1 and P_0, representing the inclusion or not of the corresponding edge. Compared to the traditional ACO application, that uses a complete graph, this approach saves considerable memory and computational time.

To decide the new path each artificial ant applies the following probabilistic transition rule, that depends on pheromone values:

$$p_{ij} = \frac{\tau_{ij}}{\tau_{i0} + \tau_{i1}} \tag{5}$$

where, τ_{ij} represents the desirability of using the path j to cross edge i given by the pheromone trails, that is, the desirability of include that gate (τ_{i1}) or not (τ_{i0}).

A difference between this implementation and the traditional ACO is that in this proposal a particular heuristic for this problem is not included in the transition rule. Thus, traditional ACO parameters α, that sets the amount of pheromone deposited on the edges, and β, that sets the relative significance of pheromone versus heuristic value, are not used in this implementation.

4.2. Pheromones update

After the construction of a new solution by each ant, the pheromone trails are updated, increasing their values when ants deposit pheromone on promising paths to guide other ants in constructing new solutions, or decreasing their values due to pheromone evaporation to avoid unlimited accumulation of pheromone trails and also to allow bad choices to be forgotten.

As shown in the transition rule (Eq. (5)), the possibility for an ant to cross a path increases with the pheromone trail. It is in the implementation of this procedure where many of the variants of the ACO algorithm differ. As commented above, the $MAX-MIN$ Ant System (MMAS) variant of the ACO algorithm (Stützle & Hoos, 2000) has been used in this work.

Algorithm 2 shows the pseudocode for the *UpdatePheromones()* procedure. Lines 1–7 determine whether the best solution of the current iteration improves the best global solution so far. In this case, the limits of the trails are updated, being this a particular feature of the MMAS variant of the ACO (Stützle & Hoos, 2000), which prevents the pheromone trails from augmenting excessively, eluding disproportionately favoring some paths over others, and thus, avoiding premature convergence to local optima. Note that in these lines the progress of the algorithm is also controlled, counting the number of iterations that have stalled. This is so because the MMAS variant will later trigger a mechanism to restart the pheromone matrix if it detects that the search process has stalled (lines 18–21 of Algorithm 2). These two strategies are described again in more detail in the next section.

The evaporation process prevents the algorithm from premature convergence to suboptimal regions and from getting stuck in a local optimum, by decreasing τ by a constant rate ρ (the pheromone evaporation rate):

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} \tag{6}$$

Then, ants deposit pheromone on the paths they have crossed in their construction. The MMAS strongly exploits the best paths found since only the *iteration-best* ant, that is, the ant that constructed the best solution in the current iteration, or the *best-so-far* ant, that is, the ant that constructed the best solution so far, deposits pheromones in each iteration:

$$\Delta(\tau_{ij})^{best} = \begin{cases} 1/f(S^{best}), & \text{if path j for edge i belongs to } S^{best} \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

Algorithm 2: UpdatePheromones_and_DaemonActions

```
   // Check whether a new best-solution is found
1  if solution-iteration-best-ant improves global-best-solution-so-far then
2      global-best-solution-so-far = solution-iteration-best-ant
3      trail_max = $1/(\rho\times$ global-best-solution-so-far)
4      trail_min = trail_max$/(2\times n)$
5      stall_iters = 0
6  else
7      stall_iters++
   // Update pheromones with iteration-best ant or best-so-far ant
   // Applying Equation (6) and Equation (7)
8  if iteration % update_gb then
9      update_pheromone(iteration_best_ant)
10 else
11     update_pheromone(best_so_far_ant)
   // Gradually increase the frequency best-so-far ant updates the pheromone trails
12 if stall_iters < restart_iters$\times$0.25 then
13     update_gb = 10
14 else if stall_iters < restart_iters$\times$0.50 then
15     update_gb = 5
16 else
17     update_gb = 2
   // Restart pheromone trails when algorithm is stagnated
18 if stall_iters > restart_iters then
19     init_pheromone_trails()
20     stall_iters = 0
21     update_gb = 10
```

where $f(S^{best})$ is the function score of the solution S^{best} found by the *iteration-best* ant or the *best-so-far* ant.

In lines 8–11 of Algorithm 2, it can be seen how the pheromone update is sometimes based on the solution of the best ant in the current iteration and other times on the solution of the best ant so far. How often this happens depends on the progression of the algorithm and is controlled in the daemon procedure, explained in the next section.

4.3. Daemon actions

The daemon actions procedure is used to implement centralized actions which cannot be performed by single ants because they require access to non-local information. A daemon action is, for instance, the gathering of global information to decide whether or not deposit additional pheromones to guide the search process from a non-local perspective. Local search procedures are also popular daemon actions.

In most general implementations of the MMAS algorithm, the use of *iteration-best* solution and the *best-so-far* solution in Eq. (7) alternates. The choice of the relative frequency with which the two pheromone update rules are applied has an impact in the search: when pheromone updates are always performed by the *best-so-far* ant, the search focuses very quickly around the *best-so-far* solution, whereas when the *iteration-best* ant updates pheromones the search is less directed. Experimental results indicate that for small problems it may be best to use only *iteration-best* pheromone updates, while for large ones the best performance is obtained by giving an increasingly stronger emphasis to the *best-so-far* solution. This can be achieved by gradually increasing the frequency with which the *best-so-far* ant updates the pheromone trails. Lines 12–17 of Algorithm 2 implement the adjustment between the update of the pheromone trials using the *iteration-best* ant or the *best-so-far* ant.

The strategy to update the pheromone trails using only the *iteration-best* ant or the *best-so-far* ant, speedups the algorithm, but may also lead to a premature convergence. To offset this effect, MMAS limits the possible range of pheromone trails to an interval $[\tau_{min}, \tau_{max}]$. This improves the exploration, giving each path a minimum probability of being chosen. It can be observed that, because of pheromone evaporation, the maximum value for the pheromone trail is $\tau_{max} = 1/(\rho \cdot f(S*))$, where $f(S*)$ is the score of the optimal solution. Based on this observation, and using the *best-so-far* solution obtained during the execution of the algorithm, the maximum trial value can be estimated as $\tau_{max} = 1/(\rho \cdot f(S^{best}))$. The minimum value of the pheromone trial τ_{min} is estimated as a constant factor lower than τ_{max}, in this work $\tau_{min} = \tau_{max}/2n$ being n the number of edges (lines 3–4 in Algorithm 2).

In the MMAS variant, the pheromone trails are initialized to the maximum value. This leads to a further diversification in the algorithm since during the first iterations the relative difference between the different paths will not be very marked, as it is when the trails are initialized to zero. Furthermore, when it is detected that the algorithm stagnates at local minima, that is, when a certain number of subsequent iterations fail to improve the best solution obtained so far, the pheromone matrix is restarted, again with the pheromone trails initialized to their maximum value (lines 18–21 of Algorithm 2). This favors the search for alternative paths that allow the algorithm getting out from the local minima.

5. Parallel implementation

The inherently parallel nature of the ACO, where each ant can build its path independently from the rest, allow to parallelize the algorithm both in the data and in the population domains. Many different parallel strategies can easily be adapted to ACO. Most of them can be classified into fine-grained and coarse-grained strategies. In fine-grained approaches, few ants are assigned to individual processors that exchange information frequently. In coarse-grained strategies, larger sub-populations or even entire populations are assigned to individual processors and information exchanged is lower. A recent review of the literature on parallel approaches for the ACO algorithms can be found in González, Osorio, Pardo, Banga, and Doallo (2022).

A fined-grained parallelization attempts to find parallelism in the sequential algorithm preserving its behavior by assigning few and small tasks to unique processes that would carry out a frequent exchange of information among them. In the case of the ACO metaheuristic, finding the parallelism in the sequential algorithm is straightforward, since most of the time-consuming operations are placed in loops that can be performed in parallel. However, the frequent exchange of information negatively affects the scalability of these approaches.

Therefore, a more efficient solution is a coarse-grained approach, which involves looking for a parallel variant of the sequential algorithm. In many metaheuristics the most popular coarse-grained solution consists of implementing an island-based model. In these models, the initial population is distributed among different islands where the original algorithm is executed in isolation and, from time to time, islands exchange information that allow them to benefit their searches from the knowledge of the rest. This solution drastically reduces communications between distributed processes, which improves the scalability of the proposals. In the case of the ACO metaheuristics, the islands are called *colonies* and each colony is assigned to a different processor and executes the sequential ACO algorithm remotely, until a communication step is reached, where information about the best solutions found so far and/or the pheromone matrix is exchange among processors. This kind of parallel ACO implementation is usually called *multicolony* model.

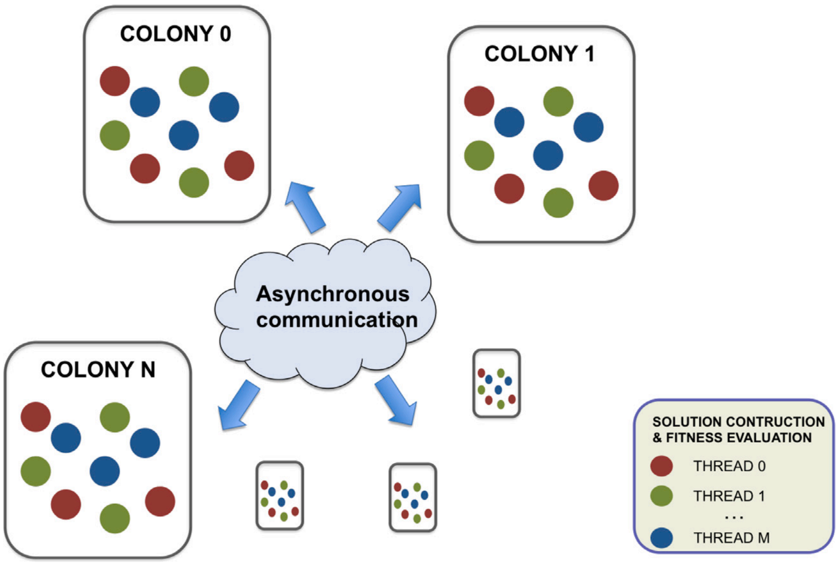
COLONY 0
COLONY 1
COLONY N
Asynchronous communication
SOLUTION CONTRUCTION & FITNESS EVALUATION
THREAD 0
THREAD 1
...
THREAD M

Fig. 3. Scheme of the MPI+OpenMP multicolony implementation.

In this work we have followed the MPI+OpenMP hybrid parallel strategy for the ACO metaheuristic described in González et al. (2022). Fig. 3 illustrates this approach, that improves the efficiency of the algorithm through a balance between diversification and intensification. It combines a coarse-grained multicolony parallelization, focused on stimulating diversification in the search and cooperation between different colonies, with a fine-grained strategy, aimed at speeding up calculations by carrying out the construction of solutions in parallel.

Algorithm 3 shows a basic pseudocode for the parallelization adapted for the problem at hand. This pseudocode reflects both levels of parallelization. Each colony in the group executes a copy of this code in isolation, albeit communicating with each other as explained below. On the other hand, within each colony, those tasks that can be performed in parallel are distributed among different threads, following a fork-join programming model.

The majority of the operations of the ACO algorithm can be carried out in parallel. Specifically, both the construction of the solutions and their evaluation (lines 6–12 in Algorithm 3) are directly parallelizable using the OpenMP shared memory library. When the running thread of each colony reaches the parallel loop (line 6), it creates a group of threads so that the iterations of the loop are distributed among them. At the end of the parallel loop (line 12) the threads are synchronized and join into a single thread again, that continues the colony execution. The only procedure that cannot be straight parallelized within each single colony is the updating of the pheromone matrix. However, since in the MMAS variant the update is carried out by a sole ant, its computational cost is irrelevant.

On top of this fine-grained parallelization within each colony, a cooperative scheme is built among a group of different colonies. When designing this cooperative scheme some key issues have to be addressed, such as what information is exchanged, which colonies are involved in the communication process, when and how the information exchange takes place, and how the information received from other colonies is used. In our approach, the exchange of information between colonies is driven by the quality of the solutions, rather than by an elapsed time or a predefined effort, to achieve more effective cooperation. Thus, every time a new promising solution is achieved in one of the colonies (line 13 in Algorithm 3), it is spread to the rest of colonies (line 14). When a new solution arrives in the colony and improves the best-solution-so-far, the latter is replaced by the former (lines 18–19). An asynchronous communication protocol is used between colonies

Algorithm 3: MPI+OpenMP ACO parallelization

```
   // Initialize MPI environment
1  MPI_Init
2  Initialize colony parameters
3  Initialize colony pheromone trails
   // Prepare a reception buffer for asynchronous
      communications
4  MPI_IRecv(promising-solution,request)
5  while termination condition not met do
       // Share work among ants in the colony
6      $$ omp parallel loop
7      for $k = 1$ to number_of_ants do
8          step = 1
9          while step < number_of_edges do
               // Applying Equation (5)
10             ApplyACODecisionRuleToSelectGate
11             step ++
12         EvaluateSolutionFitness
       // When new local best solution is found, send it
          asynchronously
13     if local-best-solution-so-far < global-best-solution-so-far then
14         MPI_ISend(local-best-solution-so-far)
       // Check the reception of foreign promising
          solutions
15     repeat
           // MPI check asynchronous reception
16         MPI_Test(request, recvflag)
17         if recvflag then
18             if promising-solution < global-best-solution-so-far then
19                 global-best-solution-so-far ← promising solution
               // Prepare a new reception buffer for next
                  receptions
20             MPI_IRecv(promising-solution,request)
21     until (!recvflag)
       // Using Algorithm 2
22     UpdatePheromones_and_DaemonActions
```

(lines 4, 14, 16 and 20), thus avoiding inactive processes while waiting for the information exchange. Therefore, the proposal implements a completely decentralized approach, in contrast to classic centralized approaches, which address the efficient use of available resources while providing a fault tolerant solution, since the coordinator represents a *single point of failure* that is avoided here.

6. Experimental results

In this section the performance of the proposed parallel ACO is evaluated using three benchmarks related to the target application, early drug discovery. Specifically, the benchmarks used in this work are related to the study of cell signaling in liver cancer. Benchmarks used to carry out the experiments reported in this section are summarized in Table 1. *LiverDREAM* (Saez-Rodriguez et al., 2009) was part of the DREAM4 — In Silico Network Challenge (Chun et al., 2011). *ExtLiverBMC2012* (Terfve et al., 2012) uses phosphorylation data obtained from a human hepatocellular cell-line (the HepG2) after perturbation with inhibitory drugs, and *ExtLiverPCB* (Morris et al., 2011) is a larger version of ExtLiverBMC2012.

All the experiments were performed at the Galicia Supercomputing Center (CESGA) using the *FinisTerrae-II* supercomputer. Each FinisTerrae-II node is composed of two Intel Haswell E5-2680v3 CPUs running at 2.50 GHz, with 12 cores per processor (24 cores per node), and 128 GB of RAM. The nodes are connected using an InfiniBand FDR 56 Gbps interconnect using a Fat-tree topology.

To comprehensively evaluate the performance of different approaches, tests are performed from different perspectives. Most of the tests were performed with a quality stopping criterion. An objective value is pre-established and the effort needed to reach the target solution is evaluated. These experiments have, as a handicap, the long execution times required in some cases to achieve the optimum, especially if we attempt to compare with sequential executions. For this reason, other tests were carried out using a predefined effort. The algorithm finishes when a maximum execution time has been spent.

In this work, execution time has been used as the main performance metric. This is a very popular metric when evaluating the performance of parallel algorithms, since the main objective of parallelization is to improve response time. When the computer infrastructure is the same in all experiments, this is the preferred metric to assess and compare different parallel alternatives. In addition, for the design of the experiments in this work, the guide of Hansen, Auger, Finck, and Ros (2009) has been followed.

Due to the stochastic behavior of the algorithm, 100 runs are performed independently for each experiment, and the distribution of the data is taken into account in the evaluation and discussion of the results.

6.1. Tuning of the ACO implementation

As usual in most metaheuristics, the performance of the ACO algorithm is straight dependent on the choice of its parameters. As described in Section 4, in the implementation proposed in this work, only the evaporation rate (ρ), the number of ants, and the number of stagnant iterations that triggers the restart of the pheromone matrix, are applicable and, thus, need to be tuned. In this subsection we discuss the choice made for each of them based on the performance and efficiency of the method.

First, the number of stagnant iterations that trigger the restart helps to speed up convergence, especially for problems that tend to stagnate at local optima, as this is the case. Fig. 4 shows the cumulative probability of reaching the global optimum for the LiverDREAM case-study with different value of this parameter. Similar experiments have been carried out with the rest of the benchmarks obtaining similar results. Thus, the number of stagnant iterations that trigger the restart is set to 50 for these problems, so that convergence is accelerated but not too aggressive to remove the information in the pheromone trails while it can still be useful.

Table 1
Benchmarks: #edges: number of edges, #nodes: number of nodes, #readouts: number of measurements, #cond.: number of experimental conditions, TP: time-points, opt.: optimum of the objective function cost.

Benchmark	#edges	#nodes	#readouts	#cond.	TP	opt.
LiverDREAM	62	17	7	25	0, 30	0.01977874
ExtLiverPCB	99	31	15	64	0, 30	0.024962085
ExtLiverBMC2012	108	31	15	64	0, 30	0.024959938

Previous studies in the literature (Nallaperuma, Wagner, & Neumann, 2015) have demonstrated that the number of ants in the MMAS variant of the ACO does not present a significant impact in the final result in terms of the number of iterations for convergence. However, the more ants there are in the system, the more costly the loop to build the solution, in terms of the number of evaluations performed and the sequential execution time. For this reason, applications using the MMAS variant use few ants regardless of the size of the problem. However, notice that in the parallel versions it may be convenient to increase the number of ants, especially if several threads are going to be used to distribute the tasks of the construction loop. For the experiments reported here we have used 24 ants. This number is chosen because the thread configuration used in the experiments is multiple of 2, and, thus, the distribution of ants between tasks/processors is balanced.

Finally, the pheromone evaporation rate (ρ) is key to rapidly decrease the pheromone trails for *bad* paths (when large rates are used) or to keep the exploration ability (when small rates are taken). Therefore, tuning this ACO parameter is critical to maximize the algorithm efficacy and efficiency.

Fig. 5 shows the cumulative probability of reaching the global optimum for the LiverDREAM case-study with different value of parameter ρ. Moreover, in this work we include in the algorithm a self-adaptation strategy for the evaporation parameter. Initially ρ starts with large values (which can be selected by the user, or by default $\rho = 0.5$). This high rate will remove from the pheromone matrix the trails of initial solutions that are quickly improved in the first iterations of the algorithm. Once the algorithm advances and begins to stagnate, ρ parameter begins to decrease 0.1 for every 20% of the stagnant iterations before a new restart. Fig. 5 shows also the cumulative probability for an experiment using the self-adaptive parameter.

In summary, for the experiments reported in the following sections, only two parameters of the ACO are user-defined: the number of ants and the number of stagnant iterations that trigger a restart of the pheromone matrix. The former has been set to 24 ants for each colony, while the latter is set to 50 stagnant iterations.

6.2. Assessment of the sequential ACO versus GA

As it has already been mentioned, until now the CellNOpt framework used a GA algorithm as an optimization method for network training (Terfve et al., 2012). This method has limitations, especially regarding the execution time and the quality of the solutions when the size of the problem grows. In this work we have compared the performance of the proposed ACO with the GA. However, the GA provided in CellNOpt package was programmed in R and executed sequentially. In order to fairly compare the results, we have implemented the GA algorithm in C, using the same structure as for the ACO algorithm and also including a parallel implementation based on the island model.

Fig. 6 shows the cumulative probability of reaching the global optimum for both the ACO algorithm and the GA for the LiverDREAM case-study in sequential executions. Up to 100 runs were performed for each of the experiments, limiting the execution time to 1 h in all of them. In the lower part of the picture it can be seen that, in some

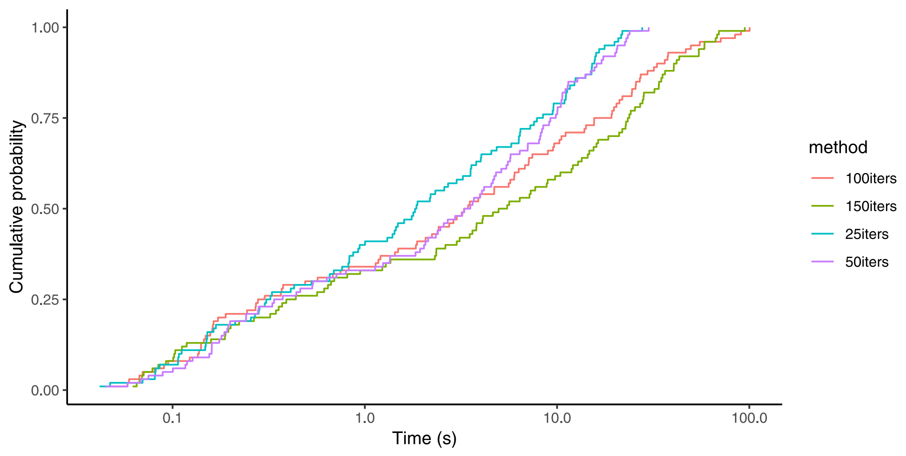

Fig. 4. Cumulative probability of reaching the global optimum in LiverDREAM case-study using 6 colonies and 4 threads each, for different values of the number of stagnant iterations to trigger the restart. Results for 100 experiments each.

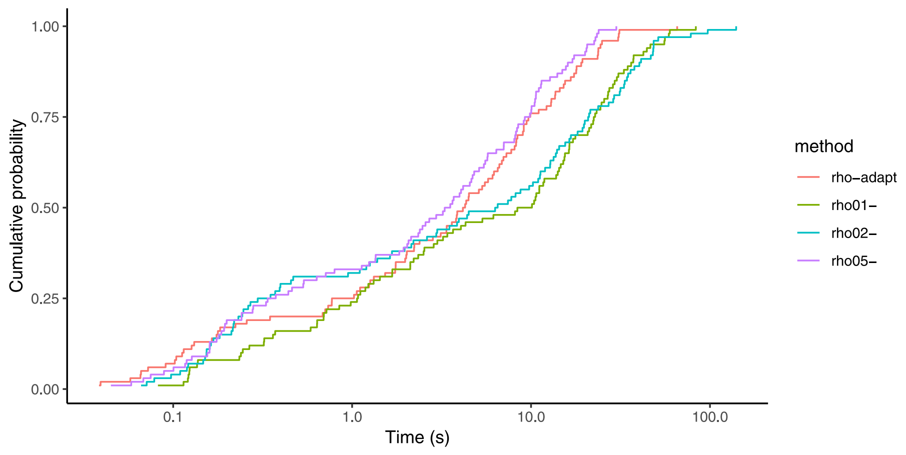

Fig. 5. Cumulative probability of reaching the global optimum in LiverDREAM case-study using 6 colonies and 4 threads each, for different values of ρ. Results for 100 experiments each.

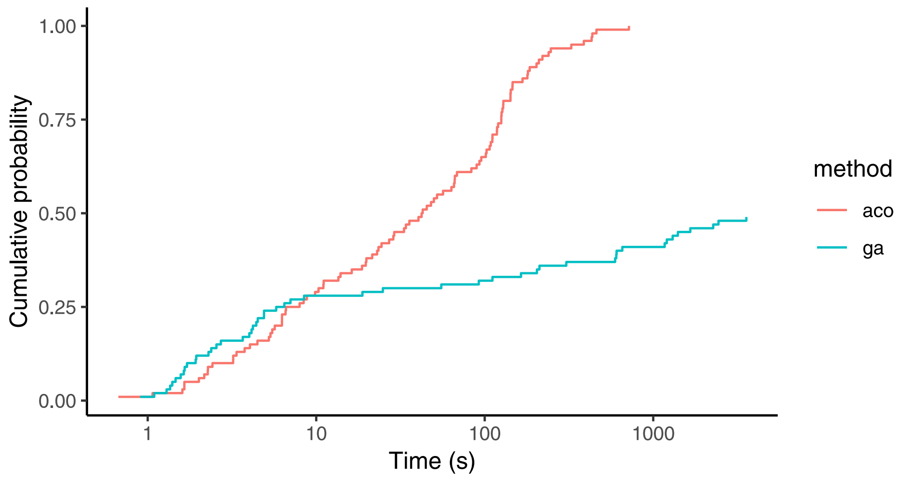

Fig. 6. Cumulative probability of reaching the global optimum in LiverDREAM case-study for both ACO and GA algorithm. Results for 100 experiments each. Maximum time allowed: 1 h.

experiments, the GA achieves the optimum in shorter execution times than the ACO. However, all in all, the ACO is more robust than the GA because the dispersion of the results in the set of 100 experiments is smaller, and all the executions converge in less than 1000 s, when only 49% of GA executions converge in 3600 s.

To get insight in these results, Fig. 7 shows the best values achieved during the first 800 iterations, for both the ACO and the GA, in sequential executions. It can be clearly seen how the GA algorithm is faster in the first iterations but it converges quickly to suboptimal solutions while ACO is able to get out of them. Each point in the

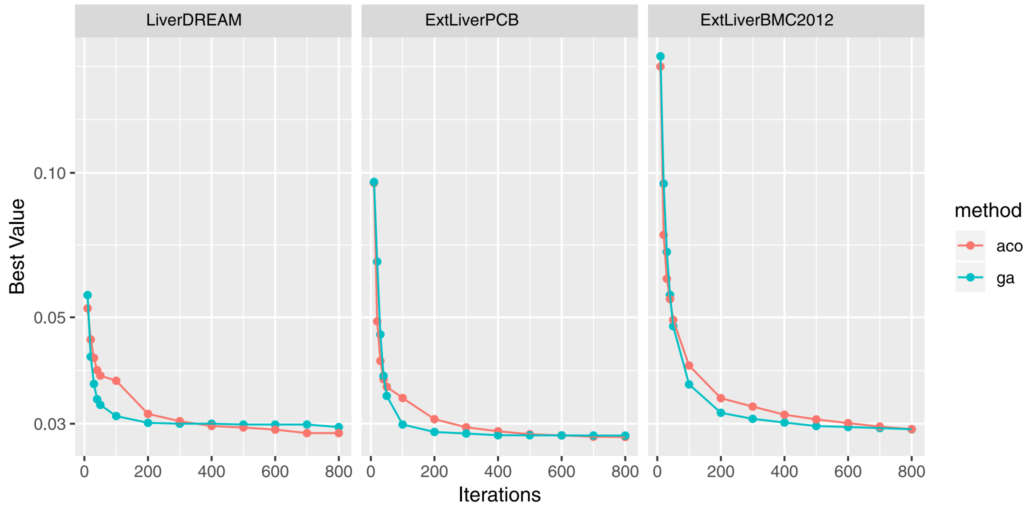

Fig. 7. Best value over iteration for the first 800 iterations, comparing ACO with GA algorithms. Each point represents the average of the best value obtained in 100 runs.

figure represents the average of the final solutions obtained in 100 runs. Although in the first iterations GA has a faster convergence, in the long term ACO outperforms GA, as shown in Fig. 6.

6.3. Performance evaluation of the parallel implementation

The parallel implementation proposed in this work allows, not only to accelerate the resolution of the problem through the construction of the solutions and their evaluation in parallel, by means of the fine-grained parallelization that distributes the calculations among the different threads, but also to help the algorithm to get out of local optima thanks to the cooperation of multiple colonies that are conducting searches simultaneously.

Fig. 8 shows, for illustrative purposes, the results obtained for the LiverDREAM case study in a pre-defined effort test (maximum execution time 10 s) with different combinations of number of colonies and number of threads per colony. Again each experiment is carried out 100 times, and the best values achieved in each of them are dotted (that is, there are 100 dots in each plot). As can be seen, with a single colony and a single thread (sequential execution), most of the runs do not reach the global optimum (0.019) and get stuck in a local optimum (0.031). However, as the number of threads and, especially, the number of colonies increases the number of runs that reach the global optimum increases as well.

More detailed results of these experiments, as well as their comparison with the GA algorithm (and its parallel version) can be seen in Fig. 9, for the three case-studies. The average of the best value obtained in the 100 runs of each experiment, as well as the percentage of success, that is, the percentage of executions that managed to reach the global optimum, are reported.

It can be seen how the parallel versions improve the results achieved as the number of processors increases. But it is also interesting to see how the distribution of computational resources between threads, that share the tasks of each colony, and independent colonies, that cooperate with each other, impacts in the behavior of the parallel algorithm. For example, if 8 processors are dedicated to the execution, it can be seen the differences between using a single colony with 8 parallel threads, or using 8 colonies with a single thread each. In these problems that tend to stagnate, cooperation between colonies is essential to get out of stagnation, so it is usually more appropriate to enhance the colonies against the threads. It is also important to assess the results obtained by the ACO versus the GA in the parallel experiments. The inherently parallel nature of the ACO means that parallelization, even following the same strategy, obtains better results than in GA, where more operations have to be carried out sequentially.

Table 2
p-values results of Mann Whitney test to compare the performance of parallel ACO with the sequential ACO and with the parallel GA.

	Colonies	seq.ACO vs. par.ACO	ACO vs. GA
LiverDREAM	1	–	**0.2450**
	2	1.11e−7	6.70e−5
	4	1.84e−9	1.17e−3
	6	5.54e−9	6.02e−3
	8	2.20e−16	1.18e−11
	12	1.12e−11	1.18e−11
	24	2.20e−16	3.51e−8
ExtLiverPCB	1	–	**0.2714**
	2	5.01e−5	**0.5173**
	4	2.36e−11	7.51e−5
	6	6.49e−12	1.59e−3
	8	3.43e−14	2.27e−4
	12	7.89e−13	2.28e−2
	24	2.20e−16	9.24e−7
ExtLiverBMC2012	1	–	1.26e−2
	2	5.37e−3	6.10e−3
	4	5.51e−3	6.93e−4
	6	5.51e−3	6.93e−4
	8	7.89e−5	7.13e−4
	12	9.85e−7	3.42e−6
	24	2.29e−10	1.82e−9

6.4. Statistical analysis

In order to prove the significance of the results a statistically analyze was performed using the Mann Whitney U test (MacFarland & Yates, 2016). The Mann Whitney U test is a non-parametric statistical test to compare the results obtained by a pair of algorithms. The test is based on two hypotheses: the null and the alternative hypothesis. The null hypothesis assumes that there is no difference between the ranks of the results obtained by the two algorithms, and the alternative hypothesis considers that there is a difference between them. We have applied the Mann Whitney test to compare first the results obtained in the parallel version of the ACO algorithm with the results obtained in its sequential version, and second, the results obtained in the parallel version of the ACO algorithm with the results obtained in the parallel version of the GA algorithm. To carry out these tests, we used the results obtained in the experiments reported in the previous section.

Table 2 shows the *p-value* obtained by each pair of algorithms compared. We have used a test based on a 5% significant level. Thus, when *p-value* is larger than 0.05 then the null hypothesis is true, whereas when *p-value* is under 0.05 the alternative hypothesis is true.

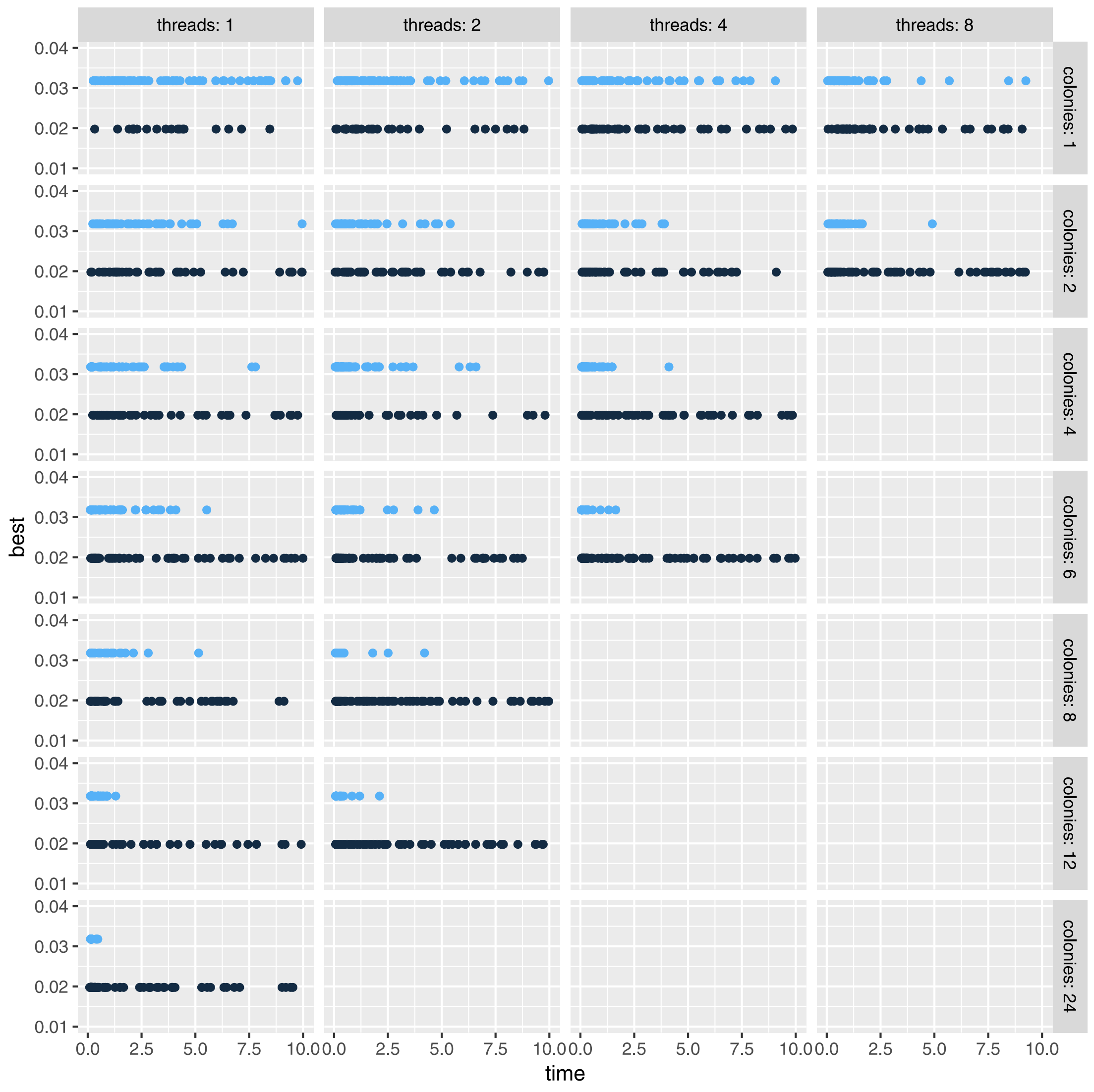

Fig. 8. Best value achieved for different combination of colonies and threads in the parallel implementation in predefined effort experiments. Case-study: LiverDREAM. Number of runs per experiment: 100. Maximum time: 10 s. Blue dots: suboptimal solution, black dots: optimal solution.

As it can be seen, for the comparison between the sequential ACO and the parallel multicolony approach, in all the cases the alternative hypothesis is true, that is, the Wann Whitney U test concludes that the parallel ACO outperforms the sequential ACO. Note also that the *p-value* decreases, in general, when the number of colonies increases. For the comparison between the ACO and the GA algorithm, note that the comparison is performed with the same number of colonies en the ACO algorithm and the GA algorithm. That is, the results reported in the table for 6 colonies compare the parallel ACO with 6 colonies against the parallel GA with 6 colonies. Most of the *p-value* for these tests are under 0.05, that is the parallel ACO outperforms its counterpart parallel GA. However, for the sequential version of the ACO and the GA in LiverDREAM and ExtLiverPCB benchmarks, the *p-value* achieved was larger than 0.05. This means that the null hypothesis is true and we cannot prove that there is significance difference between the results obtained by ACO and GA. This result is consistent with the discussion in Section 6.2 and Fig. 6, where it has been evidenced that, in the first iterations of the algorithm, the convergence results of the GA are similar and even superior to the ACO, but in the long term GA stagnates easily. Note that these Mann Whitney tests have been applied to the results of the predefined-effort experiments reported in Fig. 9.

6.5. Speedup and efficiency analysis

A very popular metric for assess parallel versus sequential implementations is speedup. In the HPC field, speedup is usually defined as the relationship between the time required to execute the sequential algorithm versus the time required to execute the parallel implementation. However, when studying stochastic problems, such as metaheuristics, it is necessary to be very careful with this definition. When performing a series of runs in each experiment, it is necessary to use the averages of the times to calculate the average speedup. Moreover, the stopping criterion should be the quality of the solution reached (that is, all the experiments should stop at the same target value), since otherwise we may obtain a large speedup while the parallel executions reach worse values from the quality point of view. In this case, we have performed experiments with quality stopping criterium for the LiverDREAM case study, and we show in Fig. 10 the average time to achieve the global optimum, the minimum time of the 100 runs, the speedup, and the efficiency. The speedup is computed as:

$$sp = T^{avg}_{sequential}/T^{avg}_{parallel}.$$

The efficiency is calculated as:

$$e = sp/cores.$$

		Average best value					% hits				
		Colonies	Threads/OMP				Colonies	Threads/OMP			
		MPI	1	2	4	8	MPI	1	2	4	8
LiverDREAM	aco	1	0,029524	0,028441	0,026396	0,026276	1	19%	28%	45%	46%
		2	0,026877	0,025794	0,025554	0,023990	2	38%	48%	52%	64%
		4	0,025313	0,026997	0,024471		4	52%	40%	61%	
		6	0,025554	0,024832	0,022305		6	51%	58%	78%	
		8	0,025674	0,022907			8	51%	73%		
		12	0,024832	0,021704			12	58%	84%		
		24	0,022185				24	80%			
	ga	1	0,029404	0,029524	0,028923	0,029043	1	20%	19%	24%	23%
		2	0,029043	0,028923	0,028321	0,028321	2	23%	24%	29%	29%
		4	0,027840	0,027960	0,027960		4	33%	32%	32%	
		6	0,027719	0,027960	0,026516		6	34%	31%	44%	
		8	0,027960	0,027479			8	32%	36%		
		12	0,027599	0,028080			12	35%	31%		
		24	0,026757				24	42%			

		Average best value					% hits				
		Colonies	Threads/OMP				Colonies	Threads/OMP			
		MPI	1	2	4	8	MPI	1	2	4	8
ExtLiverPCB	aco	1	0,028207	0,027877	0,027682	0,027395	1	3%	17%	31%	33%
		2	0,027905	0,027997	0,027674	0,027512	2	19%	22%	33%	37%
		4	0,027633	0,027516	0,027633		4	34%	35%	32%	
		6	0,027593	0,026986	0,026824		6	33%	48%	54%	
		8	0,027350	0,026864			8	39%	52%		
		12	0,027633	0,026702			12	33%	57%		
		24	0,026824				24	53%			
	ga	1	0,028218	0,028269	0,027925	0,028038	1	3%	14%	27%	24%
		2	0,028103	0,028010	0,028605	0,027957	2	18%	24%	10%	25%
		4	0,028370	0,028078	0,028200		4	13%	23%	20%	
		6	0,028200	0,027916	0,027876		6	20%	27%	28%	
		8	0,027997	0,027795			8	25%	30%		
		12	0,027795	0,027674			12	30%	33%		
		24	0,028038				24	24%			

		Average best value					% hits				
		Colonies	Threads/OMP				Colonies	Threads/OMP			
		MPI	1	2	4	8	MPI	1	2	4	8
ExtLiverBMC2012	aco	1	0,028020	0,027348	0,027265	0,026741	1	23%	41%	43%	56%
		2	0,027591	0,027429	0,027024	0,026538	2	35%	39%	49%	60%
		4	0,027307	0,026538	0,026822		4	42%	61%	54%	
		6	0,026903	0,027186	0,026215		6	52%	45%	69%	
		8	0,027064	0,026538			8	48%	61%		
		12	0,026741	0,026579			12	56%	60%		
		24	0,026255				24	68%			
	ga	1	0,027916	0,028166	0,027672	0,028198	1	21%	19%	33%	20%
		2	0,027793	0,028036	0,027955	0,027955	2	30%	24%	26%	26%
		4	0,027874	0,027995	0,027914		4	28%	25%	27%	
		6	0,027550	0,027834	0,027469		6	36%	29%	38%	
		8	0,027833	0,027793			8	29%	30%		
		12	0,027834	0,027753			12	29%	31%		
		24	0,027591				24	35%			

Fig. 9. Average best value achieved and %hits (global optimum achieved) in predefined effort experiments, for different configurations of the number of colonies (MPI processes) and the number of OpenMP threads in each colony. Number of runs per experiment: 100. Maximum time: LiverDREAM = 10 s, ExtLiverPCB = 100 s, ExtLiverBMC2012 = 1000 s. Color coded from worse in red, to best in green.

Several conclusions can be drawn from these results. First, as already mentioned, the more cores used in the parallel implementation, the better the results obtained. In addition, it is again highlighted that the allocation of resources between colonies and threads affects performance. Specifically, it can be seen how the average time improves as the number of colonies increases, although the minimum time improves with the number of threads (for the same number of total cores). For example, a better average time is obtained with 24 colonies and 1 thread per colony than with 6 colonies and 4 threads per colony, but with the latter combination a lower minimum time is obtained. This is due to the effect of each of the parallel strategies employed (fine grain vs. coarse grain). Parallelization into different cooperating colonies helps drive executions that stagnate out of their local optima, thanks to diversification. The fine-grained parallelization, on its turn, favors an intensification in the search within each colony because it speeds up the calculations.

Average time

Colonies	Threads/OMP			
MPI	1	2	4	8
1	86,802128	77,096032	34,306727	20,529403
2	47,388304	39,609473	15,080601	10,549374
4	23,204108	19,323643	10,670650	
6	14,193998	9,869347	5,928100	
8	12,957251	8,962198		
12	9,025619	4,895997		
24	4,991933			

minimum time

Colonies	Threads/OMP			
MPI	1	2	4	8
1	0,669456	0,162129	0,150189	0,222448
2	0,528825	0,156996	0,066042	0,066123
4	0,110078	0,058069	0,04518	
6	0,154828	0,071127	0,044841	
8	0,124357	0,079087		
12	0,126762	0,07804		
24	0,115271			

Speedup

Colonies	Threads/OMP			
MPI	1	2	4	8
1	-	1,125896	2,530178	4,228186
2	1,831720	2,191449	5,755880	8,228178
4	3,740809	4,492017	8,134662	
6	6,115411	8,795123	14,642487	
8	6,699116	9,685361		
12	9,617305	17,729205		
24	17,388482			

Efficiency

Colonies	Threads/OMP			
MPI	1	2	4	8
1	-	56%	63%	53%
2	92%	55%	72%	51%
4	94%	56%	51%	
6	102%	73%	61%	
8	84%	61%		
12	80%	74%		
24	72%			

Fig. 10. Results of average time, minimum time, speedup and efficiency for experiments with a quality stopping criteria for LiverDREAM benchmark. Number of runs per experiment: 100. Color coded from worse in red, to best in green.

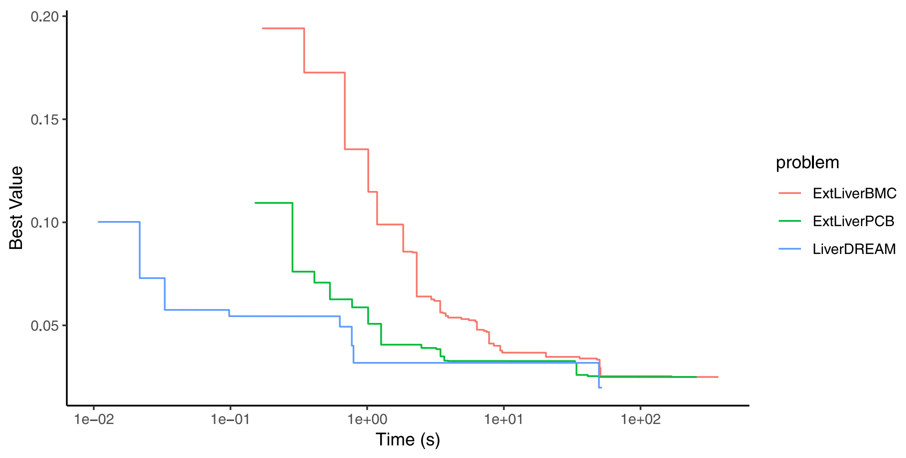

Fig. 11. Typical convergence curves using ACO to solve the three case-studies.

6.6. Convergence analysis

Fig. 11 shows, for illustrative purposes, some typical convergence curves for the three problems at hand using the ACO algorithm proposed. One of the first issues that can be observed in these convergence curves is the tendency of the ACO algorithm to get stuck in suboptimal solutions.

In solving these problems, rapid convergence to local suboptimals and stagnation is not only a problem attributable to the ACO algorithm, since other metaheuristics, specifically the GA commented above, also present this same handicap. It would therefore be very interesting to have a mathematical demonstration of the convergence of the different metaheuristics, even for specific problems. However, as pointed in Hussain, Mohd Salleh, Cheng, and Shi (2019), although metaheuristics have successfully solved a wide range of difficult problems, this field is still immature in terms of convergence, complexity, and runtime analysis.

Fortunately, there are several works (Gutjahr, 2002; Stützle & Hoos, 2000) that have already demonstrate that, for the MMAS variant of the ACO algorithm, the objective function value of the best solution found so far converges with probability one to the optimal value. These papers consider only the sequential version of the algorithm. But they do not estimate the theoretical speed of convergence. The parallelization of the algorithm can certainly help at this point, as has been demonstrate throughout this work.

Therefore, it is interesting to study how the parallelization of the algorithm affects the convergence of the sequential ACO method. However, the theoretical development in parallel approaches of most metaheuristics is at an early stage. Therefore, mathematical analysis regarding the rate of convergence is very difficult, if not impossible. In fact, there is a lack of mathematical analysis and mostly ad-hoc approaches have been adopted in the literature to measure the performance of parallel metaheuristic algorithms.

The results reported in previous sections already implicitly reflect the improvement in the convergence of the proposed parallel algorithm. In this work, a hybrid parallelization is proposed combining fine-grained parallelization with coarse-grained parallelization. The former, as explained previously, hits intensification by means of the distribution of ants among processors. With this parallelization, the original sequential algorithm does not change its properties, and the convergence of the sequential method, in number of iterations and evaluations carried out, is maintained when the number of threads increases. In

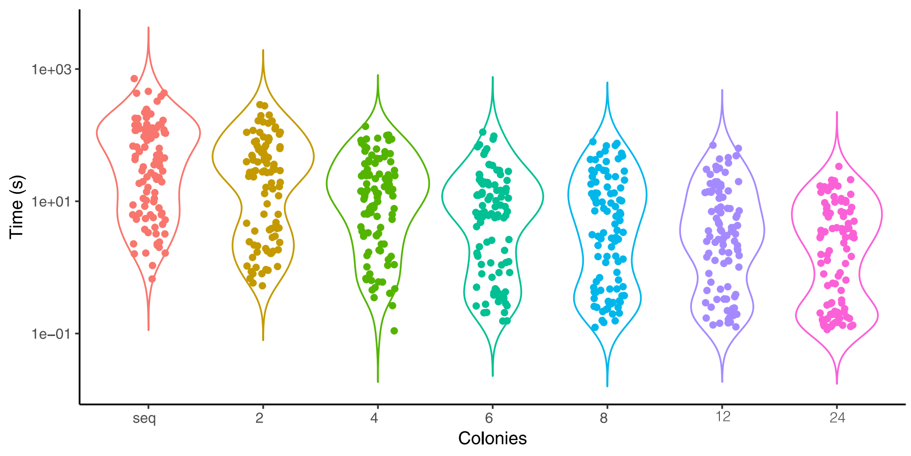

Fig. 12. Beanplots with the distribution of execution times for experiments with a quality stopping criteria.

Table 3
Number of restarts triggered in the experiments of Fig. 10 for a different number of colonies using one thread per colonie.

Colonies	Number of restarts			%executions
	min	Average	max	without restarts
seq	2	295	2404	0
2	1	158	954	0
4	0	78	447	2
6	0	44	351	7
8	0	37	233	15
12	0	26	206	17
24	0	14	99	26

these cases we see an improvement in the elapsed time motivated by the execution of the original tasks in parallel. However, with coarse-grained parallelization, as explained in detail in Section 5, the focus is on diversification. The original algorithm changes, being the parallel algorithm the result of the cooperation between different instances of the sequential algorithm. In this case, the convergence of the algorithm improves, since the systemic properties of the parallel version and a simple instance of the sequential algorithm are not the same. Fig. 12 shows the dispersion of the results (execution time) in the tests that use quality as stopping criterium, for the LiverDREAM benchmark, using different numbers of colonies but only one thread per colony. It can be clearly seen that the more colonies cooperating, the less time to reach the optimum. Note that the y-axis scale is logarithmic. The dispersion of the results is significantly lower (the average execution time decreases more than an order of magnitude between the sequential and the parallel execution with 24 colonies). These results demonstrate the improvement in convergence and in the robustness of the parallel algorithm.

To further demonstrate the improvement in convergence when the number of colonies increases, Table 3 shows the number of restarts of the pheromone matrix that have been triggered in each of the experiments in Fig. 10. Let us remember that, as explained in Section 5, this ACO algorithm uses a restart strategy for the pheromone matrix every time it detects that the ants are stuck in a local minimum (lines 18–21 in Algorithm 2). Specifically, Table 3 shows the average number of restarts that were triggered in the 100 runs of each experiment, the minimum and maximum number of restarts, and the percentage of runs that reached the optimum without needing any restart. It can be seen that as the number of colonies increases, the number of average restarts drops significantly. Also the maximum number of restarts needed by the slowest execution decreases with the number of colonies. And, finally, the percentage of executions for each experiment that did not need any restarts increases from 0% in the sequential case to 26% with 24 colonies.

6.7. *Scalability analysis*

To study the scalability of the parallel algorithm, we have also carried out experiments using a large number of cores. Table 4 shows results for different number of colonies and threads, using from 2 nodes (48 cores) to 8 nodes (192 cores) of the FinisTerrae-II. These experiments demonstrate the scalability of the algorithm, since using up to 192 cores the algorithm continues to scale.

It can be seen that, as the number of cores continues to increase, the percentage of success continues to grow. It is also important to note that, in general, the number of colonies has a larger impact than the number of threads, since, as already indicated, the number of colonies significantly impacts the ability of the algorithm to get out of local optima. Regarding the execution time, note that the time reported in the table is the average time of the 100 runs, however, depending on the percentage of success of these runs, many would have finished in the maximum time allowed without reaching the optimum (case of the ExtLiverBMC2012 benchmark), so this average time does not allow us to calculate a speedup.

6.8. *Comparison with other training methods*

Formulated as a non-linear optimization problem, the training of network models in CellNOpt can be solved with stochastic search methods as shown previously. However, other alternatives are possible. In a recent work (Gjerga et al., 2020), a declarative problem-solving technique using Answer Set Programming (ASP) via Cell ASP Optimizer (CASPO) method and a Integer Linear Programming (ILP) formulation via the IBM ILOG CPLEX Optimization Studio has been also tested.

Table 5 shows a comparison between the ASP and ILP approaches and the parallel ACO proposed in this paper. In the case of the ASP and ILP columns the time results are the ones reported in Gjerga et al. (2020). In the case of the columns for ACO, we report results of the parallel algorithm using a single supercomputer node (24 cores) and 8 nodes (192 cores). The time reported is the minimum time obtained in all the experiments. To analyze the results we must take into account that the experiments have not been carried out in the same infrastructure, so, to be fair, the conclusions drawn must take this into account. However, though a deeper analysis cannot be supported on this results, the times obtained using the parallel ACO metaheuristic are competitive with alternative methods.

7. Conclusions

Here we describe the adaptation of the ACO algorithm for the training of cell signaling networks which can help in the study of

Table 4
Results of best, average and worst solutions in predefined effort experiments, for the three case-studies. #c × #t: number of colonies × number of threads. Number of runs per experiment: 100. Maximum time: 10 s for LiverDREAM case-study and 100 s for ExtLiverPCB and ExtLiverBMC2012.

	Cores	conf.	fitness			Avg.	%hits
		#c × #t	Best	Average	Worst	Time (s)	
LiverDREAM	48	24 × 2	0.019778749	0.020981858	0.031809836	2.88	90
	48	48 × 1	0.019778749	0.020259993	0.031809836	1.98	96
	96	24 × 4	0.019778749	0.019778749	0.019778749	1.89	100
	96	48 × 2	0.019778749	0.019778749	0.019778749	1.06	100
	96	96 × 1	0.019778749	0.019778749	0.019778749	0.94	100
	192	48 × 4	0.019778749	0.019778749	0.019778749	0.64	100
	192	96 × 2	0.019778749	0.019778749	0.019778749	0.55	100
	192	192 × 1	0.019778749	0.019778749	0.019778749	0.39	100
ExtLiverPCB	48	24 × 2	0.024962085	0.026014277	0.029008977	44.49	74
	48	48 × 1	0.024962085	0.02617616	0.029008977	45.62	70
	96	24 × 4	0.024962085	0.02556918	0.029008977	36.99	84
	96	48 × 2	0.024962085	0.025366782	0.029008977	29.75	89
	96	96 × 1	0.024962085	0.02552865	0.029008977	29.89	86
	192	48 × 4	0.024962085	0.025204921	0.029008977	24.78	93
	192	96 × 2	0.024962085	0.02512396	0.029008977	20.71	98
	192	192 × 1	0.024962085	0.025002554	0.029008977	15.21	99
ExtLiverBMC2012	48	24 × 2	0.024959938	0.027428696	0.029007760	69.63	39
	48	48 × 1	0.024959938	0.027023977	0.029007760	60.17	49
	96	24 × 4	0.024959938	0.027145384	0.029007060	64.45	46
	96	48 × 2	0.024959938	0.026862134	0.029007760	56.10	52
	96	96 × 1	0.024959938	0.026497851	0.029007060	51.09	62
	192	48 × 4	0.024959938	0.026174074	0.029007060	42.74	70
	192	96 × 2	0.024959938	0.026497844	0.029007060	51.86	62
	192	192 × 1	0.024959938	0.026335959	0.029007060	45.96	66

Table 5
Results of execution time in seconds. Times for ASP and ILP: reported in Gjerga et al. (2020). Times for ACO: minimum time needed for parallel ACO executed in 1 or 8 nodes to reach the optimum.

Benchmark	ASP	ILP	ACO (1 node)	ACO (8 nodes)
LiverDREAM	0.060	0.136	0.044	0.032
ExtLiverPCB	239.89	4.733	1.229	0.716
ExtLiverBMC2012	273.66	33.933	3.443	0.825

new cancer drugs and therapies. These networks are modeled using the CellNOpt framework. In addition to adapting the algorithm to the problem, we also describe its parallelization using a hybrid strategy combining fine-grained parallelization using OpenMP with coarse-grained parallelization (multicolony) using MPI.

In order to ensure a fair comparison with the results previously obtained with a GA algorithm, this GA was coded using C, and parallelized using the same structure and strategy as the new ACO-based proposal, further demonstrating that the proposed parallel strategy can be easily extended to other metaheuristics.

The experimental results show the good performance of our ACO implementation, outperforming the GA, and the convenience of the parallel version, especially for difficult problems that present many local minima where many methods can get trapped. In particular, the fine-grained parallelization accelerates the calculations within each colony, while the multicolony cooperation favors escaping from the local minima and therefore facilitates the convergence to the global solution.

Our approach can be further refined by pursuing several promising research directions, but we are especially interested in exploring the extension of the parallel implementation to include approaches based on heterogeneous colonies, including self-adaptive strategies, so it can efficiently handle very complex large-scale problems.

Our contribution will help the development of better models in order to obtain a functional understanding of the deregulation of signaling networks in disease. Further, these models can be the basis for the development of new therapies.

The source code is made public at https://doi.org/10.5281/zenodo.6630397

CRediT authorship contribution statement

Patricia González: Conceptualization, Methodology, Software, Validation, Writing – original draft, Supervision. **Roberto Prado-Rodriguez:** Software, Validation. **Attila Gábor:** Conceptualization, Validation, Writing – review & editing. **Julio Saez-Rodriguez:** Conceptualization, Writing – review & editing, Funding acquisition. **Julio R. Banga:** Conceptualization, Methodology, Writing – review & editing, Funding acquisition. **Ramón Doallo:** Conceptualization, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We have shared the data/code through Zenodo, linked in the conclusions section.

Acknowledgments

This work was supported by the Ministry of Science and Innovation of Spain (PID2019-104184RB-I00/AEI/10.13039/501100011033), and by Xunta de Galicia, Spain and FEDER funds of the EU (Centro de Investigación de Galicia accreditation 2019–2022, ref. ED431G 2019/01; Consolidation Program of Competitive Reference Groups, ref. ED431C 2021/30). JRB acknowledges funding from the Ministry of Science and Innovation of Spain MCIN/AEI/10.13039/501100011033 through grant PID2020-117271RB-C22 (BIODYNAMICS). AG acknowledges funding from the European Union Innovative Medicines Initiative TransQST (agreement No. 116030). Authors also acknowledge the Galician Supercomputing Center (CESGA) for the access to its facilities. Funding for open access charge: Universidade da Coruña/CISUG.

References

Adams, R., Clark, A., Yamaguchi, A., Hanlon, N., Tsorman, N., Ali, S., et al. (2013). SBSI: an extensible distributed software infrastructure for parameter estimation in systems biology. *Bioinformatics*, *29*(5), 664–665.

Balsa-Canto, E., Banga, J. R., Egea, J. A., Fernandez-Villaverde, A., & de Hijas-Liste, G. M. (2012). Global optimization in systems biology: Stochastic methods and their applications. In I. I. Goryanin, & A. B. Goryachev (Eds.), *Advances in systems biology* (pp. 409–424).

Banga, J. R. (2008). Optimization in computational systems biology. *BMC Systems Biology*, *2*(1), 47.

Banga, J. R., & Balsa-Canto, E. (2008). Parameter estimation and optimal experimental design. *Essays in Biochemistry*, *45*, 195–210.

Beldjilali, B., Benadda, B., & Sadouni, Z. (2020). Vehicles circuits optimization by combining GPS/GSM information with metaheuristic algorithms. *Romanian Journal of Information Science and Technology*, *23*(T), T5–T17.

Bojan-Dragos, C.-A., Precup, R.-E., Preitl, S., Roman, R.-C., Hedrea, E.-L., & Szedlak-Stinean, A.-I. (2021). GWO-based optimal tuning of type-1 and type-2 fuzzy controllers for electromagnetic actuated clutch systems. *IFAC-PapersOnLine*, *54*(4), 189–194.

Bullnheimer, B., Kotsis, G., & Strauß, C. (1998). Parallelization strategies for the ant system. In *High performance algorithms and software in nonlinear optimization* (pp. 87–100). Springer.

Chen, L., Sun, H.-Y., & Wang, S. (2012). A parallel ant colony algorithm on massively parallel processors and its convergence analysis for the travelling salesman problem. *Information Sciences*, *199*, 31–42. http://dx.doi.org/10.1016/j.ins.2012.02.055.

Chu, S.-C., Roddick, J. F., & Pan, J.-S. (2004). Ant colony system with communication strategies. *Information Sciences*, *167*(1), 63–76. http://dx.doi.org/10.1016/j.ins.2003.10.013.

Chun, H., Kang, J., Zhang, X., Deng, M., Ma, H., & Zhao, H. (2011). Reverse engineering of gene regulation networks with an application to the DREAM4 in silico network challenge. In *Handbook of statistical bioinformatics* (pp. 461–477). Springer.

Craus, M., & Rudeanu, L. (2004). Parallel framework for ant-like algorithms. In *Third international symposium on parallel and distributed computing/third international workshop on algorithms, models and tools for parallel computing on heterogeneous networks* (pp. 36–41). http://dx.doi.org/10.1109/ISPDC.2004.37.

Da Ros, S., Colusso, G., Weschenfelder, T., De Marsillac Terra, L., De Castilhos, F., Corazza, M., et al. (2013). A comparison among stochastic optimization algorithms for parameter estimation of biochemical kinetic models. *Applied Soft Computing*, *13*(5), 2205–2214.

Delisle, P., Gravel, M., Krajecki, M., Gagné, C., & Price, W. L. (2005). Comparing parallelization of an ACO: message passing vs. shared memory. In *International workshop on hybrid metaheuristics* (pp. 1–11). Springer.

Delisle, P., Krajecki, M., Gravel, M., & Gagné, C. (2001). Parallel implementation of an ant colony optimization metaheuristic with OpenMP. In *Proceedings of the 3rd European workshop on OpenMP (EWOMP'01), Barcelona, Spain* (pp. 1–7).

Doerner, K., Hartl, R., Benkner, S., & Luckà, M. (2006). Parallel cooperative savings based ant colony optimization – multiple search and decomposition approaches. *Parallel Processing Letters*, *16*(3), 351–370.

Dorigo, M., & Stutzle, T. (2002). A short convergence proof for a class of ACO algorithms. *IEEE Transactions on Evolutionary Computation*, *6*(10.1109).

Egea, J. A., Balsa-Canto, E., García, M. S. G., & Banga, J. R. (2009). Dynamic optimization of nonlinear processes with an enhanced scatter search method. *Industrial and Engineering Chemistry Research*, *48*(9), 4388–4401.

Egea, J. A., Martí, R., & Banga, J. R. (2010). An evolutionary method for complex-process optimization. *Computers & Operations Research*, *37*(2), 315–324.

Ellabib, I., Calamai, P., & Basir, O. (2007). Exchange strategies for multiple ant colony system. *Information Sciences*, *177*(5), 1248–1264. http://dx.doi.org/10.1016/j.ins.2006.09.016.

Gábor, A., & Banga, J. R. (2015). Robust and efficient parameter estimation in dynamic models of biological systems. *BMC Systems Biology*, *9*(1), 74.

Gjerga, E., Trairatphisan, P., Gabor, A., Koch, H., Chevalier, C., Ceccarelli, F., et al. (2020). Converting networks to predictive logic models from perturbation signalling data with CellNOpt. *Bioinformatics*, *36*(16), 4523–4524.

González, P., Argüeso-Alejandro, P., Penas, D. R., Pardo, X. C., Saez-Rodriguez, J., Banga, J. R., et al. (2019). Hybrid parallel multimethod hyperheuristic for mixed-integer dynamic optimization problems in computational systems biology. *The Journal of Supercomputing*, *75*(7), 3471–3498.

González, P., Osorio, R. R., Pardo, X. C., Banga, J. R., & Doallo, R. (2022). An efficient ant colony optimization framework for HPC environments. *Applied Soft Computing*, *114*, Article 108058.

González, P., Pardo, X. C., Penas, D. R., Teijeiro, D., Banga, J. R., & Doallo, R. (2017). Using the cloud for parameter estimation problems: comparing spark vs MPI with a case-study. In *Proceedings of the 17th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGRID'2017)*.

Gutjahr, W. J. (2002). ACO algorithms with guaranteed convergence to the optimal solution. *Information Processing Letters*, *82*(3), 145–153.

Guziolowski, C., Videla, S., Eduati, F., Thiele, S., Cokelaer, T., Siegel, A., et al. (2013). Exhaustively characterizing feasible logic models of a signaling network using answer set programming. *Bioinformatics*, *29*(18), 2320–2326.

Hadian, A., Shahrivari, S., & Minaei-Bidgoli, B. (2012). A fine-grained parallel ant colony system for shared-memory architectures. *International Journal of Computer Applications*, *53*(8).

Hansen, N., Auger, A., Finck, S., & Ros, R. (2009). *Real-parameter black-box optimization benchmarking 2010: experimental setup*: *Technical reports rapports de recherche RR-6828*, Institut National de Recherche en Informatique et en Automatique (INRIA).

Hussain, K., Mohd Salleh, M. N., Cheng, S., & Shi, Y. (2019). Metaheuristic research: a comprehensive survey. *Artificial Intelligence Review*, *52*(4), 2191–2233.

Iorio, F., Knijnenburg, T. A., Vis, D. J., Bignell, G. R., Menden, M. P., Schubert, M., et al. (2016). A landscape of pharmacogenomic interactions in cancer. *Cell*, *166*(3), 740–754.

Ji, X., & Xu, Y. (2006). libSRES: a c library for stochastic ranking evolution strategy for parameter estimation. *Bioinformatics*, *22*(1), 124–126.

Jie, X., CaiYun, L., & Zhong, C. (2008). A new parallel ant colony optimization algorithm based on message passing interface. In *2008 IEEE Pacific-Asia workshop on computational intelligence and industrial application, Vol. 2* (pp. 178–182). http://dx.doi.org/10.1109/PACIIA.2008.248.

Jostins, L., & Jaeger, J. (2010). Reverse engineering a gene network using an asynchronous parallel evolution strategy. *BMC Systems Biology*, *4*(1).

Klamt, S., Saez-Rodriguez, J., Lindquist, J. A., Simeoni, L., & Gilles, E. D. (2006). A methodology for the structural and functional analysis of signaling and regulatory networks. *BMC Bioinformatics*, *7*(1), 1–26.

Lee, W.-P., Hsiao, Y.-T., & Hwang, W.-C. (2014). Designing a parallel evolutionary algorithm for inferring gene networks on the cloud computing environment. *BMC Systems Biology*, *8*(1), 5.

Li, X., Chen, L., & Tang, Y. (2020). HARD: Bit-split string matching using a heuristic algorithm to reduce memory demand. *Romanian Journal of Information Science and Technology*, *23*(T), T94–T105.

Ling Chen, Hai-Ying Sun, & Shu Wang (2008). Parallel implementation of ant colony optimization on MPP. In *2008 international conference on machine learning and cybernetics, Vol. 2* (pp. 981–986). http://dx.doi.org/10.1109/ICMLC.2008.4620547.

Lv, Q., Xia, X., & Qian, P. (2006). A parallel aco approach based on one pheromone matrix. In *International workshop on ant colony optimization and swarm intelligence* (pp. 332–339). Springer.

MacFarland, T. W., & Yates, J. M. (2016). Introduction to nonparametric statistics for the biological sciences using R.

Michel, R., & Middendorf, M. (1998). An island model based ant system with lookahead for the shortest supersequence problem. In *International conference on parallel problem solving from nature* (pp. 692–701). Springer.

Michel, R., & Middendorf, M. (1999). An ACO algorithm for the shortest common supersequence problem. In *New ideas in optimization* (pp. 51–62). McGraw-Hill.

Mitsos, A., Melas, I. N., Siminelakis, P., Chairakaki, A. D., Saez-Rodriguez, J., & Alexopoulos, L. G. (2009). Identifying drug effects via pathway alterations using an integer linear programming optimization formulation on phosphoproteomic data. *PLoS Computational Biology*, *5*.

Mocholi, J. A., Jaen, J., & Canos, J. H. (2005). A grid ant colony algorithm for the orienteering problem. In *2005 IEEE congress on evolutionary computation, Vol. 1* (pp. 942–949). http://dx.doi.org/10.1109/CEC.2005.1554784, Vol. 1.

Morris, M. K., Saez-Rodriguez, J., Clarke, D. C., Sorger, P. K., & Lauffenburger, D. A. (2011). Training signaling pathway maps to biochemical data with constrained fuzzy logic: quantitative analysis of liver cell responses to inflammatory stimuli. *PLoS Computational Biology*, *7*(3), 1–20.

Morris, M. K., Saez-Rodriguez, J., Sorger, P. K., & Lauffenburger, D. A. (2010). Logic-based models for the analysis of cell signaling networks. *Biochemistry*, *49*(15), 3216–3224.

Nallaperuma, S., Wagner, M., & Neumann, F. (2015). Analyzing the effects of instance features and algorithm parameters for max–min ant system and the traveling salesperson problem. *Frontiers in Robotics and AI*, *2*, 18.

Palafox, L., Noman, N., & Iba, H. (2012). Reverse engineering of gene regulatory networks using dissipative particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, *17*(4), 577–587.

Pardo, X. C., Argüeso-Alejandro, P., González, P., Banga, J. R., & Doallo, R. (2020). Spark implementation of the enhanced scatter search metaheuristic: Methodology and assessment. *Swarm and Evolutionary Computation*, *59*, Article 100748.

Penas, D. R., Banga, J. R., González, P., & Doallo, R. (2014). 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)* (pp. 173–181).

Penas, D. R., Banga, J. R., González, P., & Doallo, R. (2015). Enhanced parallel differential evolution algorithm for problems in computational systems biology. *Applied Soft Computing*, *33*, 86–99.

Penas, D. R., González, P., Egea, J. A., Banga, J. R., & Doallo, R. (2015). Parallel metaheuristics in computational biology: An asynchronous cooperative enhanced scatter search method. *Procedia Computer Science*, *51*, 630–639.

Penas, D. R., González, P., Egea, J. A., Doallo, R., & Banga, J. R. (2017). Parameter estimation in large-scale systems biology models: a parallel and self-adaptive cooperative strategy. *BMC Bioinformatics*, *18*(1), 52.

Penas, D. R., Henriques, D., González, P., Doallo, R., Saez-Rodriguez, J., & Banga, J. R. (2017). A parallel metaheuristic for large mixed-integer nonlinear dynamic optimization problems, with applications in computational biology. *PLoS One*.

Perkins, T. J., Jaeger, J., Reinitz, J., & Glass, L. (2006). Reverse engineering the gap gene network of drosophila melanogaster. *PLoS Computational Biology*, *2*(5), Article e51.

Piriyakumar, D. A. L., & Levi, P. (2002). A new approach to exploiting parallelism in ant colony optimization. In *Proceedings of 2002 international symposium on micromechatronics and human science* (pp. 237–243). http://dx.doi.org/10.1109/MHS.2002.1058041.

Pozna, C., Precup, R.-E., Horvath, E., & Petriu, E. M. (2022). Hybrid particle filter-particle swarm optimization algorithm and application to fuzzy controlled servo systems. *IEEE Transactions on Fuzzy Systems*.

Precup, R.-E., David, R.-C., Petriu, E. M., Preitl, S., & Paul, A. S. (2011). Gravitational search algorithm-based tuning of fuzzy control systems with a reduced parametric sensitivity. In *Soft computing in industrial applications* (pp. 141–150).

Saez-Rodriguez, J., Alexopoulos, L. G., Epperlein, J., Samaga, R., Lauffenburger, D. A., Klamt, S., et al. (2009). Discrete logic modelling as a means to link protein signalling networks with functional analysis of mammalian signal transduction. *Molecular Systems Biology*, *5*, 331.

Saez-Rodriguez, J., Alexopoulos, L. G., Zhang, M., Morris, M. K., Lauffenburger, D. A., & Sorger, P. K. (2011). Comparing signaling networks between normal and transformed hepatocytes using discrete logical models. *Cancer Research*, *71*(16), 5400–5411.

Sharan, R., & Karp, R. M. (2012). Reconstructing boolean models of signaling. *Research in Computational Molecular Biology*, *5*, 261–271.

Starzec, M., Starzec, G., Byrski, A., Turek, W., & Pietak, K. (2020). Desynchronization in distributed ant colony optimization in hpc environment. *Future Generation Computer Systems*, *109*, 125–133.

Stützle, T., & Hoos, H. H. (2000). MAX–MIN ant system. *Future Generation Computer Systems*, *16*(8), 889–914.

Sun, J., Garibaldi, J. M., & Hodgman, C. (2012). Parameter estimation using meta-heuristics in systems biology: a comprehensive review. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, *9*(1), 185–202.

Talbi, E.-G., Roux, O., Fonlupt, C., & Robillard, D. (2001). Parallel ant colonies for the quadratic assigment problem. *Future Generation Computer Systems*, *17*(4), 441–449.

Tan, G. W.-H., Ooi, K.-B., Leong, L.-Y., & Lin, B. (2014). Predicting the drivers of behavioral intention to use mobile learning: A hybrid SEM-neural networks approach. *Computers in Human Behavior*, *36*, 198–213.

Tang, Z., Chai, X., Wang, Y., & Cao, S. (2020). Gene regulatory network construction based on a particle swarm optimization of a long short-term memory network. *Current Bioinformatics*, *15*(7), 713–724.

Teijeiro, D., Pardo, X. C., González, P., Banga, J. R., & Doallo, R. (2016). Implementing parallel differential evolution on spark. In *European conference on the applications of evolutionary computation* (pp. 75–90).

Teijeiro, D., Pardo, X. C., Penas, D. R., González, P., Banga, J. R., & Doallo, R. (2016). Evaluation of parallel differential evolution implementations on MapReduce and spark. In *European conference on parallel processing* (pp. 397–408).

Teijeiro, D., Pardo, X. C., Penas, D. R., González, P., Banga, J. R., & Doallo, R. (2017). A cloud-based enhanced differential evolution algorithm for parameter estimation problems in computational systems biology. *Cluster Computing*, under Minor Revision.

Terfve, C., Cokelaer, T., Henriques, D., MacNamara, A., Goncalves, E., Morris, M. K., et al. (2012). CellNOptR: a flexible toolkit to train protein signaling networks to data using multiple logic formalisms. *BMC Systems Biology*, *6*(1), 133.

Traynard, P., Tobalina, L., Eduati, F., Calzone, L., & Saez-Rodriguez, J. (2017). Logic modeling in quantitative systems pharmacology. *CPT: Pharmacometrics & Systems Pharmacology*, *6*(8), 499–511.

Tsutsui, S., & Fujimoto, N. (2010). Parallel ant colony optimization algorithm on a multi-core processor. In *International conference on swarm intelligence* (pp. 488–495). Springer.

Twomey, C., Stützle, T., Dorigo, M., Manfrin, M., & Birattari, M. (2010). An analysis of communication policies for homogeneous multi-colony aco algorithms. *Information Sciences*, *180*(12), 2390–2404. http://dx.doi.org/10.1016/j.ins.2010.02.017.

Videla, S., Guziolowski, C., Eduati, F., Thiele, S., Grabe, N., Saez-Rodriguez, J., et al. (2012). Revisiting the training of logic models of protein signaling networks with ASP. In *Computational methods in systems biology* (pp. 342–361).

Villaverde, A. F., & Banga, J. R. (2014). Reverse engineering and identification in systems biology: strategies, perspectives and challenges. *Journal of the Royal Society Interface*, *11*(91), Article 20130505.

Zúñiga, E. C. T., Cruz, I. L. L., & García, A. R. (2014). Parameter estimation for crop growth model using evolutionary and bio-inspired algorithms. *Applied Soft Computing*, *23*, 474–482.