Computers and Chemical Engineering 153 (2021) 107440

Contents lists available at ScienceDirect

Computers and Chemical Engineering

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

Parallel hyper-heuristics for process engineering optimization

Paola P. Oteiza[a,b], Juan I. Ardenghi[a], Nélida B. Brignole[a,c,*]

[a] Laboratorio de Investigación y Desarrollo en Computación Científica (LIDECC)-Departamento de Ciencias e Ingeniería de la Computación (DCIC), Universidad Nacional del Sur (UNS), Bahía Blanca, Argentina
[b] Departamento de Ingeniería Química (DIQ), Universidad Nacional del Sur (UNS), B8000 Bahía Blanca, Argentina
[c] Planta Piloto de Ingeniería Química (Universidad Nacional del Sur -CONICET) Camino La Carrindanga km. 7 - 8000 Bahía Blanca, Argentina

ARTICLE INFO

Article history:
Received 12 December 2020
Revised 16 April 2021
Accepted 1 July 2021
Available online 12 July 2021

Keywords:
Optimization
Evolutionary algorithms
Metaheuristics
Hyper-heuristics
Parallel programming

ABSTRACT

This paper presents the general framework of a parallel cooperative hyper-heuristic optimizer (PCHO) to solve systems of nonlinear algebraic equations with equality and inequality constraints. The algorithm comprises the classical metaheuristics called Genetic Algorithms, Simulated Annealing and Particle Swarm Optimization, whose parameters are adaptively chosen during the executions. A Master-Worker architecture was designed and implemented, where the Master processor ranks the solution candidates informed by the metaheuristics and immediately communicates the most promising candidate to update all Workers. Algorithmic performance was tested with general models, most of them corresponding to PSE process systems. The results confirmed the efficiency of the proposed approach since both online parameter retuning and parallel processing sped up the search.

1. Introduction

The exploration of the potential value of computational intelligence in process-engineering applications is a challenging topic. There are optimization processes of industrial interest associated to functions that involve many local solutions. Over the last few decades research interest in hyper-heuristic methodologies has significantly grown. These successful tools provide the potential for increasing the generality level of search methods. Therefore, there is a need in the PSE optimization field to identify opportunities for the application of hyper-heuristics in order to satisfy efficiently the demand for solving more advanced problems.

Computational Intelligence provides solutions for complex real-world problems by means of many computational approaches inspired in biology. In Evolutionary Computation, nature-inspired algorithms have recently become popular due to their simplicity and flexibility. Yang (2015) summarized the state-of-the-art for these algorithms. In turn, Huang et al. (2019) provided a survey of automatic parameter tuning methods for metaheuristic algorithms. Although the quick advances in Evolutionary Algorithms (EAs) have resulted in a richer literature (Vikhar, 2016), there has been little emphasis on their application for large industrial problems (Bonyadi et al., 2019).

Fisher (1963) introduced the concept of high-level heuristic methods, concluding that at certain stages of the search process the combination of various low-level heuristics was more effective than their individual application. Later, a new class of approximate algorithms called metaheuristic techniques emerged (Glover, 1986). Their basic idea was to combine different heuristic methods at a higher level to achieve an efficient effective exploration of the search space. A literature review about metaheuristics can be found in Pellerin et al. (2020). Metaheuristics are general strategies that guide the search process. Their goal is to explore the search space efficiently in order to find almost optimal solutions. Metaheuristics are non-exact algorithms and are generally nondeterministic (Luna Valero, 2008). They can also incorporate mechanisms to avoid unpromising regions of the search space. Two types of search strategies can be distinguished. On the one hand, there are "smart" extensions of local search methods, which try to avoid local minima and move to other promising regions of the search space. Simulated Annealing (SA) is a representative technique of these trajectory-based strategies. On the other hand, another type of metaheuristics is based on population. They incorporate a learning component that implicitly or explicitly attempts to learn the correlation between problem variables in order to identify regions of the search space with high-quality solutions. Examples of this approach are EAs, which include Genetic Algorithms (GA), and swarm strategies, such as Particle Swarm Optimization (PSO).

Ting et al. (2015) remarked that novel techniques and approaches should be based on mathematical theory and insightful

* Corresponding author.
E-mail address: dybrigno@criba.edu.ar (N.B. Brignole).

https://doi.org/10.1016/j.compchemeng.2021.107440
0098-1354/

analysis of the main mechanisms so that new hybrid algorithms would provide more effective ways to solve real-world large problems. This conceptual goal comprises the motivational essence behind the hyper-heuristics. Burke et al. (2019) suggested an updated definition of the term "hyper-heuristics", describing it as high-level methodologies that operate on a search space of heuristics rather than directly on a search space of solutions. A survey of the state of the art in this field was given by Burke et al. (2013). In turn, Asta et al. (2013) studied the generation of heuristics for an online problem, by applying an apprenticeship-learning based technique. Afterwards, Burke et al. (2019) presented an integrated classification of hyper-heuristics, especially distinguishing between generation and selection hyper-heuristics, depending on whether the algorithm either generates heuristics or both chooses and controls them, respectively. Lately, Drake et al. (2020) provided a review paper of selection hyper-heuristics.

An up-and-coming approach to solve complex optimization problems is the one of hyper-heuristics, whose agents are combinations of nature-inspired techniques. Ortiz-Bayliss et al. (2016) studied selection hyper-heuristics, analyzing how heuristics can collaboratively be employed to take full advantage of the combination of their strengths. In turn, Oteiza et al. (2018) presented a hyper-heuristic approach about pipelining, which combined GA with an ant colony optimization (ACO) and SA. When both SA and GA ran within the same algorithm, the latter contributed to diversify the search by enlarging the search space. Later, Liu et al. (2019) analyzed the case of West-East Natural Gas Pipeline II by optimizing this problem with the following metaheuristics: GA, SA and PSO. They discussed their performance based on specific scenarios. Although SA proved to be the slowest algorithm, it yielded the best optimization results. In contrast, GA exhibited the fastest behavior for the same purpose. Lhotská et al. (2006) concentrated on swarm intelligence. By using ACO and PSO, they analyzed various examples where PSO treated each optimization problem as a particle and the optimization function determined an adaptive value. In comparison with ACO, PSO differed in its information sharing strategy and it can be concluded that all of the PSO particles tended to converge faster to the best solution in most cases. Moreover, for job-shop scheduling Nguyen and Zhang (2017) proposed a PSO-based hyper-heuristics with very competitive performance because it proved to be significantly faster than a genetic-programming based hyper-heuristics. From an individual point of view, metaheuristic algorithms have extensively been recognized as effective approaches for solving complex optimization problems (Mahdavi et al., 2015). As to combinations of metaheuristics, Owa et al. (2017) concluded that GA, SA and PSO are promising candidates for team work. For the optimization of a real-life safety system, they concluded that their approach performed better than individual methods by assessing the quantity of fitness evaluations for various instances. In particular, the success of the proposed hyper-heuristics was attributed to an intelligent trade-off between exploration and exploitation of the full capacities of the three metaheuristics when working simultaneously.

With respect to each metaheuristic performance, it is usually analyzed in terms of solution quality and the time required to attain a final point. It is important to develop strategies for the choice of parameters since a universal parametrization is theoretically unviable (Bäck and Schwefel, 1993). Hence, it is advisable to include a parameter tuning stage during the development and implementation of any general algorithm. It is widely accepted that the performance of EAs can be improved by parameter tuning (Karafotias et al., 2014). Among other reasons, Stützle and Lopez-Ibañez (2019) pinpointed that the automated design of metaheuristic algorithms from automatically configurable frameworks is really important since it provides an improvement over manual ad-hoc configuration methods, thus reducing the development time and human intervention in the parameter tuning process. This task, which is sometimes very time consuming and tedious, has been carried out empirically in the literature to suit specific problems. By way of illustration, Jackson et al. (2017) investigated the feasibility of tuning SA for cross-domain search. Based on the results obtained by using several instances from the Hyflex framework (Ochoa et al., 2012), they pointed out that SA is extremely sensitive to the initial parameter settings. Hence, they confirmed that the optimal parameter settings are probably different when solving various problems. They also evaluated the main effects of changing SA parameters, concluding that significant improvements can be achieved through intelligent parameter setting strategies and/or adaptive methods.

In the last years, parallel computing has experienced a significant evolution and it can offer appealing strategies to deal with time-consuming methodologies. Moreover, Coelho and Silva (2021) remarked that parallel metaheuristic techniques provide Industry 4.0 with alternative solution methods for complex shop scheduling problems. In this sense, parallel computing is powerful to boost metaheuristics since it accelerates the computation of solutions located near optimality (Alba et al., 2013). Accelerating fundamental metaheuristics has received widespread attention in the literature. In particular, Abdelhafez et al. (2019) analyzed the energy consumption behaviour of sequential and parallel GAs, while Cheng and Gen (2019) studied granularity for parallel GAs. As to SA, Lee and Kim (2019) presented a hybrid algorithm called PSAGA (Parallel SA with a Greedy Algorithm) that exhibited better performance than other SA variants in terms of computational times and accuracy. In turn, Wang et al. (2018) also achieved efficiency in the execution of a parallel SA algorithm in comparison with a conventional approach. Regarding canonic PSO, its major disadvantage is that it is prone to get stuck in local optima. Hence, Lalwani et al. (2019) presented a comprehensive systematic survey about PSO parallelization strategies that aim at enhancing its performance. In spite of recent efforts made to run metaheuristics in parallel, new studies about comparing and benchmarking the canonical optimization methods are still lacking (Abdelhafez et al., 2020). Moreover, another promising topic is the investigation of hybridization procedures where sequential sub-algorithms (either metaheuristics or other local searchers) run in a parallelized framework.

Decision-making processes improved by means of optimization strategies are an increasingly common element of modern industry throughout the world. Since there is a pervading trend towards the introduction of more realism in industrial modelling, large constrained optimization problems often arise (Ramos Figueroa et al., 2020). A critical aspect to solve many optimization problems is the availability of good feasible points. They are necessary because they help to speed up convergence by providing upper bounds for the objective function. Due to the inherent difficulties to solve most of the engineering problems, a feasible point can be regarded as a satisfactory solution, even though its objective value might not be the optimal one. Furthermore, in hard-to-solve mixed-binary nonlinear optimization problems (MINLPs) it is useful to be able to pinpoint feasible solutions of good quality very fast, instead of awaiting for proven global optimality (Schewe and Schmidt, 2019). Wherever it is even difficult to find a feasible point, being effective becomes a valuable achievement. This is especially interesting when an efficient algorithm is designed, i.e. when the best use of computational resources is made. For bilevel problems, Kleinert and Schmidt (2021) presented an algorithm whose goal was to compute bilevel feasible points of good quality as fast as posible. They suggested that an attractive topic for future research would be the integration of their strategy into state-of-the-art exact solution methods for bilevel problems.

In view of the above-described state of the art, the aims of the study presented here comprise both effectiveness and efficiency. First of all, the objective was to develop a framework of optimization strategies that proves to be effective for various PSE optimization problems. Moreover, an efficient algorithm was implemented by using a parallel approach together with an automatic updating of the crucial metaheuristic parameters in order to tackle computationally demanding models.

The Parallel Cooperative Hyper-heuristics Optimizer (PCHO) depicted here aims at obtaining high-quality solutions with an emphasis on both problem search-space diversity and reasonable execution times. Section 2 outlines the hyper-heuristic approach. Next, Section 3 refers to parameter retuning, which has been automated to find adequate settings for efficient processing. This learning technique has the ability to explore and exploit the problem search space more effectively to solve any problem whose model can be defined as a system of nonlinear algebraic equations with algebraic constraints. The algorithmic performance and results are analyzed in Section 4. Finally, some conclusions are presented in Section 5.

2. The cooperative hyper-heuristic approach

For continuous variable optimization, let us consider the solution of nonlinear problems with both equality and inequality constraints. The general formulation is given by Eq. (1). Let us assume that scalar, real functions are in n variables, i.e. $f(x) : \mathbb{R}^n \to \mathbb{R}$, $g(x) : \mathbb{R}^n \to \mathbb{R}^r$ and $h(x) : \mathbb{R}^n \to \mathbb{R}^m$, where $x \in \mathbb{R}^n$ corresponds to the optimization variables, $f(x)$ is the objective function, $g(x)$ is a system of r inequality constraints and $h(x)$ is a system of m equality constraints.

$$\begin{aligned} &\min_{s.t.} f(x) \\ &g(x) \le 0 \\ &h(x) = 0 \end{aligned} \tag{1}$$

A parallel cooperative multi-search method (PCHO) to solve the optimization problems defined by Eq. (1) was designed and implemented with a Master-Worker architecture. Each Worker (agent) performs its task independently and only communicates with the Master, who organises the cooperativism. Each Worker is in charge of its own search, while the Master establishes communication with them. The Workers are autonomous agents because they are completely self-contained. They are canonic procedures for the following metaheuristics: GA, SA and PSO. As stated above, their judicious combination looks promising when a general-purpose optimizer is desired. Moreover, given that the experienced user can often envisage the approximate location of the optimum point from an engineering viewpoint, an optional input that allows entering suggested solutions is provided.

Fig. 1 outlines the proposed hyper-heuristics (PCHO), illustrating the low-level strategy that comprises the Workers Domain. It contains the n underlying metaheuristics (Workers), who evolve separately by using n threads. A coarse-grained parallel model is followed. The complete computation carried out by each Worker runs in parallel with respect to the other Workers. Deep down, each metaheuristics runs sequentially. The model representation and the initial settings for the Workers are incorporated in the Initialization Module. A different processor is exclusively assigned to the Master, who performs the high-level strategy that leads the whole search. As soon as a Worker (either GA, SA or PSO) finishes its own single-point search, it informs the Master about the best solution it could find (Candidate Solution). The high-level methodology (Master) immediately accepts or rejects this proposed result based on its quality and locates it in an ordered list, which ranks the Candidate Solutions according to their fitness values. The rank update scheme is additive and the worst candidate is always excluded. In this way, the search behaviour improves overtime.

The Master selects the winning solution, informing everybody about the Winner. The metaheuristics that has succeeded in finding the Winner keeps its settings, while the other agents are retuned. All Workers are also updated with information coming from the newly established ranking by reassigning either the starting point for SA or enriching the initial population for GA or PSO. The agent who found the Winner is updated with the Candidate Solution that has been ranked in the third place because it is a promising initialization that deserves exploitation. The remaining agents update their search with the Winner. A different policy has to be adopted when it is impossible to detect a Winner (i. e. in the event of a tied decision). There is a Tie whenever at least two agents have simultaneously obtained the same fitness value, the condition being *relative error* $\le 10^{-1}$. When a Tie has occurred, the winner site is kept vacant, while all agents without exceptions retune their parameters and resume their own search in order to refine it.

After updating, the high-level process continues until 30 outer iterations are completed. It should be kept in mind that each Worker sequentially carries out 15 inner runs, internally administrating his own termination criteria. Finally, the Master reports the best solution found.

Aiming at achieving a balance between limited computational times and SA's potential to converge to high-quality solutions, a high-level control was introduced in PCHO in order to overcome delays. When SA is still iterating after the GA and PSO search procedures have already finished, it is necessary to check whether SA is making enough progress in its search. For this purpose, a minimum decrement is imposed (*relative decrement rate in objective function* ≥ 0.01) as an indicator of a successful search. Whenever this condition cannot be satisfied, the Master process orders SA to finish its search.

Conceptually, this cooperative hyper-heuristic strategy can be viewed as a set of sequential metaheuristic tasks running in parallel, i.e. working simultaneously in an asynchronous way in order to find the optimal solution. Each metaheuristics was implemented as a sequential algorithm. Hence, each branch of the topology is fully assigned to a given processor/ thread, thus amounting to a coarse-grained architecture. As to the communication approach, a Master-slave topology was adopted. Fig. 1 graphically depicts this topology, where the slaves were represented inside the Workers Domain. The communication allows prospective candidate solutions to be shared in order to achieve faster/ better results. Even though the computational cost of running a single metaheuristic procedure can be significantly different, this parallel execution strategy reduces the search time and improves the solution quality by means of a learning strategy that employs the different distributed search information from all the computing units.

3. Online parameter retuning

Parameter settings have significant influence on the heuristic optimization methods. Since parameter tuning is both problem-dependent and computationally difficult, ad hoc tuning strategies were incorporated into the Updating Module in order to improve online the performance of the metaheuristics during their search procedures. Hence, PCHO was conceived as a general-purpose algorithm. Therefore, a self-adaptive mechanism was implemented with a dynamic selection of the agents' settings. The learning scheme is based on past experience in order to control the parameters by varying their values to achieve the best overall performance.

Since the metaheuristics' performance greatly depends on the values of their parameters and operators, online selection is carried out to adaptively choose them during the search. From a practical

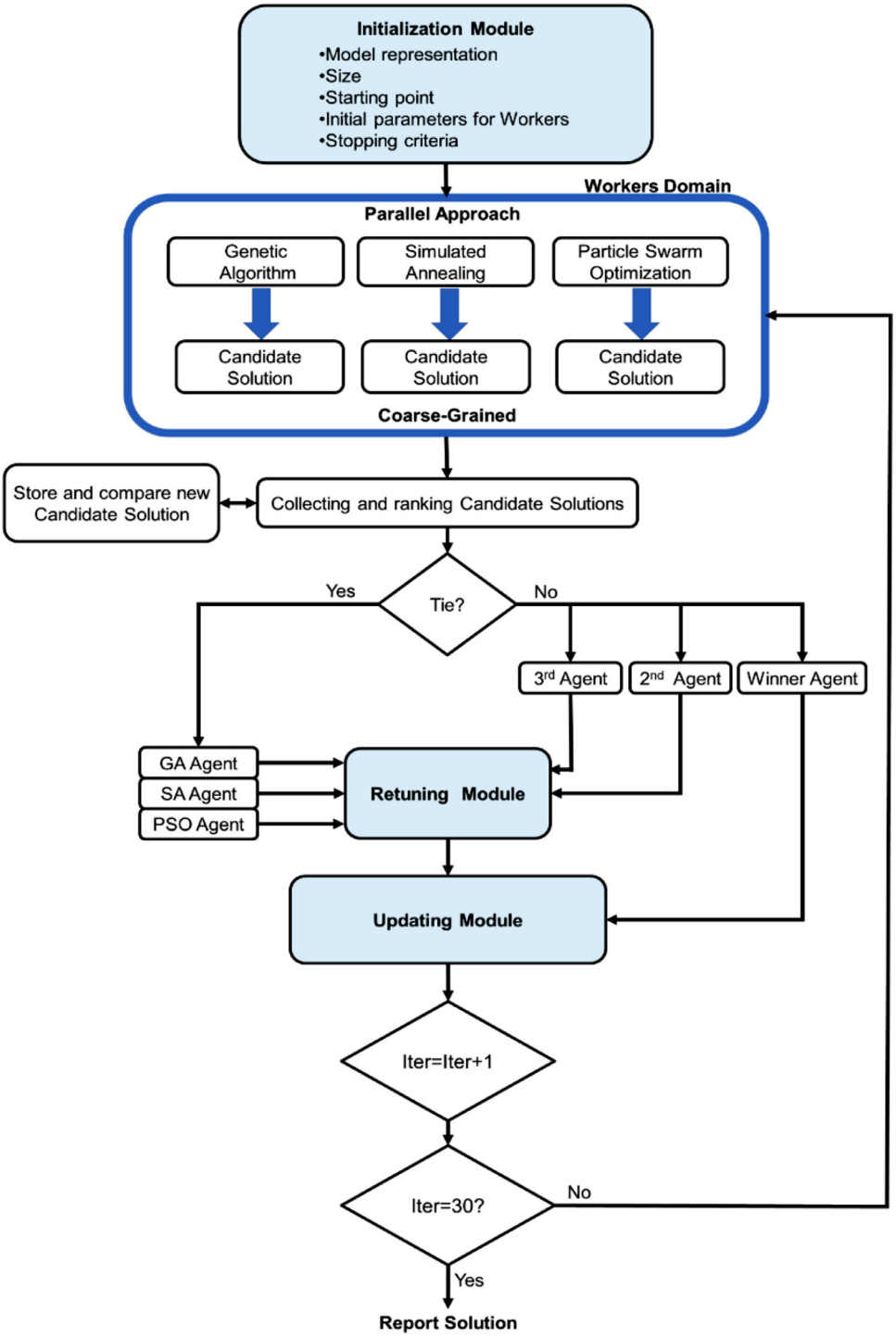


Fig. 1. The cooperative hyper-heuristic (PCHO) framework.

perspective, this strategy contributes to favour diversity. Whenever an agent has failed to provide a winning solution, which is clearly an unsuccessful past experience, some of his settings are retuned based on his latest performance for the same problem. Figs. 2–;4 show the tuning adopted for each metaheuristics in PCHO. The boxes with dark circles refer to parameter tuning, which varies depending on the metaheuristics. Each dark circle represents a potentiometer, which symbolises an instrument with a rotating contact that controls a given parameter. For SA, the tunable parameters are the temperature, the temperature function and the annealing function. Respecting PSO, it is possible to tune the swarm size, the inertial range, the self-adjustment and the social-adjustment weights. As to GA, both the crossover rate and the crossover mechanism are automatically adapted during GA evolution.

Depending on its input, each metaheuristics adopts a different behavior in order to include the best discovery so far. In general terms, the shared information contains the most promising candidate; hence, it provides an idea of the location that is worthwhile exploiting. In other words, the population is enriched with the recently informed candidate solutions in order to give adaptative advantages to the environment. In this way, convergence is favoured by diversity. In turn, for PSO the movement of each particle is

Initialization coming from the first call:
$CrFctSetNumber \leftarrow 1$
$CrRate \leftarrow 0.8$
$PS_1 = [CrFctSetNumber \;\; CrRate]$
Algorithm GA_retuning (PS_1)
(Crossover Function set numbers: $CrFctSetNumber$ = {1= Two-point; 2=Heuristic; 3=Scattered})
Assignment:
if $CrRate > 0.6$
    $CrRate \leftarrow CrRate - 0.05$
else
    if $CrFctSetNumber < 3$
        $CrFctSetNumber \leftarrow CrFctSetNumber + 1$
    else
        $CrFctSetNumber \leftarrow 1$
        $CrRate \leftarrow 0.8$
    end
end
Return $PS_1 = [CrFctSetNumber \;\; CrRate]$

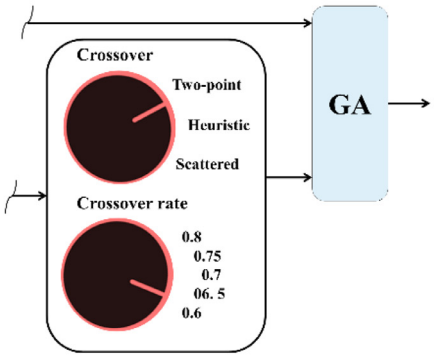
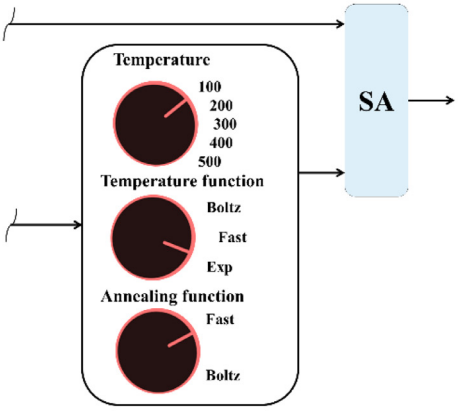


Fig. 2. Pseudoalgorithm for the retuning of GA parameters.

Initialization coming from the first call:
$Temperature \leftarrow 100$
$TempFctSetNumber \leftarrow 1$
$AnnFctSetNumber \leftarrow 1$
$PS_1 = [Temperature \;\; TempFctSetNumber \;\; AnnFctSetNumber]$
Algorithm SA_retuning (PS_1)
(Temperature Function set numbers: $TempFctSetNumber$ = {1= Boltz; 2= Fast; 3= Exp})
(Annealing Function set numbers: $AnnFctSetNumber$ = {1= Fast; 2= Boltz})
Assignment:
if $TempFctSetNumber < 3$
    $TempFctSetNumber \leftarrow TempFctSetNumber + 1$
else
    if $AnnFctSetNumber < 2$
        $AnnFctSetNumber \leftarrow AnnFctSetNumber + 1$
else
    if $Temperature < 500$
        $Temperature \leftarrow Temperature + 100$
    else
        $Temperature \leftarrow 100$
        $TempFctSetNumber \leftarrow 1$
        $AnnFctSetNumber \leftarrow 1$
    end
end
Return $PS_1 = [Temperature \;\; TempFctSetNumber \;\; AnnFctSetNumber]$



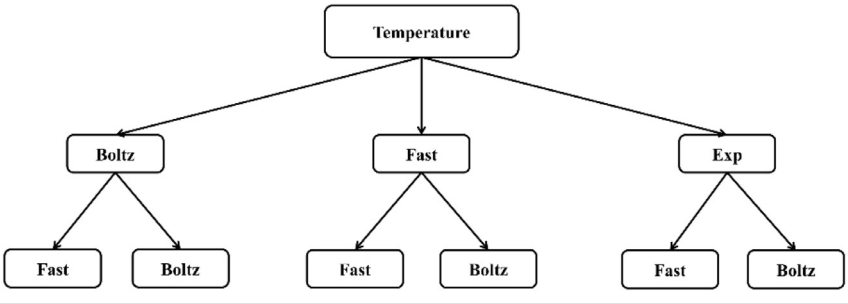


Fig. 3. Pseudoalgorithm for the retuning of SA parameters.

1: **Initialization coming from the first call:**
2: $SwarmSize \leftarrow 20$
3: $IneRangeSetNumber \leftarrow 1$
4: $SelfWeight \leftarrow 1.49$
5: $SocialWeight \leftarrow 1.49$
6: $PS_1 = [SwarmSize \;\; IneRangeSetNumber \;\; SelfWeight \;\; SocialWeight]$
7: **Algorithm** PSO_retuning (PS_1)
8: (Inertia Range set numbers: $IneRangeSetNumber = \{1 = [0.8\ 1.2];\ 2 = [0.9\ 1.3];\ 3 = [1\ 1.4];\}$)
9: **Assignment:**
10: **if** $IneRangeSetNumber < 3$
11: $IneRangeSetNumber \leftarrow IneRangeSetNumber + 1$
12: **else**
13: **if** $SelfWeight < 1.99$
14: $SelfWeight \leftarrow SelfWeight + 0.1$
15: **else**
16: **if** $SocialWeight < 1.99$
17: $SocialWeight \leftarrow SocialWeight + 0.1$
18: **else**
19: **if** $SwarmSize > 120$
20: $SwarmSize \leftarrow SwarmSize + 20$
21: **else**
22: $SwarmSize \leftarrow 20$
23: $IneRangeSetNumber \leftarrow 1$
24: $SelfWeight \leftarrow 1.49$
25: $SocialWeight \leftarrow 1.49$
26: **end**
27: **end**
28: **Return** $PS_1 = [SwarmSize \;\; IneRangeSetNumber \;\; SelfWeight \;\; SocialWeight]$

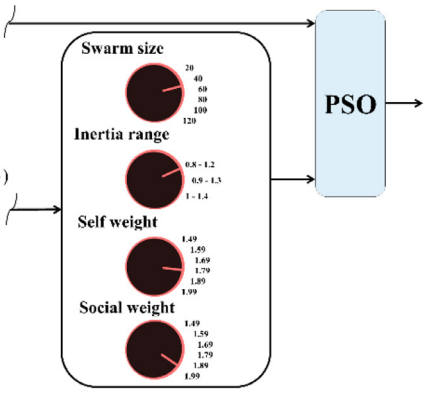


Fig. 4. Pseudoalgorithm for the retuning of PSO parameters.

guided by its own best-discovered location and the entire swarm's best-known position. Parameter retuning helps to avoid premature or slow convergence rate. For instance, changing the swarm size is effective in maintaining high diversity of the population.

It should be taken into account that the update policy depends on the metaheuristic category. PSO and GA are based on populations, whereas SA starts from an initial solution and iteratively tries to replace it by another first-class solution within the neighbourhood. In particular, SA calculates energy values (i.e., the fitness) in every iteration so as to keep the best individual. Given a newly-informed better candidate, the existing solution is replaced so that the random walk continues from a newly-created point. In this way, the restart might potentially help to escape local optima. This policy enables a wider exploration and avoids the SA algorithm to remain stuck. If SA remained stuck running towards a local optimum, restarting would be beneficial for two main reasons. First of all, computational speed might be accelerated. Secondly, the search would be guided to find the global optimum.

All the metaheuristics involved in PCHO start with a standard set of parameters and operators that are commonly used as default options in programming platforms. For all pseudoalgorithms (see Figs. 2, 3 and 4), this set is stored in PS_1, which is updated by the retuning procedure. When the retuning is triggered for a given Worker, the procedure updates PS_1, indicating a different parametric update. The retuning event takes place for all agents, excepting the Winner who has just provided the winning solution. Whenever an agent has become the Winner, it is not retuned at that stage. Instead, its last settings are kept because the successful combination seems to be adequate for the problem that is being solved.

Fig. 2 presents the pseudoalgorithm that summarizes GA retuning. The following list of options was implemented for the crossover operators: $CrFctSetNumber = 1$ means Two-point crossover, $CrFctSetNumber = 2$ indicates Heuristic crossover and $CrFctSetNumber = 3$ corresponds to Scattered crossover. For its first retuning GA starts with Two-point crossover and the crossover rate (*CrRate*) is initialized with the highest value (i.e. 0.8). Whenever GA loses, its parameters are retuned. For each updating, the crossover rate is changed in a decreasing way (i.e. always reducing it in 0.05) while keeping the crossover type as it had been set before. This is done until the crossover rate has reached the minimum value (i.e. 0.6). From this point onwards, the operator is changed by moving forward on the list of options. The crossover operator is kept constant, while going through all values of *CrRate* in a decreasing way as explained before.

Fig. 3 shows the pseudoalgorithm for SA retuning. The procedure returns a set of paths originated by touring the directed tree presented at the bottom of Fig. 3. The following list of options was implemented to set the temperature function: $TempFctSetNumber = 1$ means Boltz, $TempFctSetNumber = 2$ indicates Fast and $TempFctSetNumber = 3$ means Exp. The Annealing function choice is either Boltz or Fast. For its first retuning, SA starts with *Temperature*=100. Given that temperature, a temperature function is selected and then, deep down the Annealing function is chosen in order to complete the choice. As illustrated in Fig. 3, a tree-shaped tour is carried out to build all the combinations of settings. Each complete branch is informed through PS_1 as the next set to be employed. When the tour is over, the temperature is increased in 100 before another tour begins.

Finally, the pseudoalgorithm for PSO is presented in Fig. 4. Like in the pseudoalgorithms explained above, the tree-touring policy was also applied to build the parameter combinations. For its first retuning, PSO is initialized with the swarm size in 20, the inertia range lying between 0.8 and 1.2 and both the self and social weights in their minimum values, i.e. 1.49. As soon as PSO fails to be the Winner, the inertia range is shifted by assigning $IneRangeSetNumber = 2$, and the other parameters are kept constant. After going through all possibilities, the Swarm size is increased in 20 and the complete tour is made again.

The retuning approach explained above is basically a trial-and-error policy, keeping everything static, except for a given parameter/ operator. Though there are some theoretical improvements in parameter tuning strategies, the detailed techniques proposed in the literature may waste too much time while performing their search for optimal parameters. For example, Consoli et al. (2016) investigated population-based EAs and proposed to apply credit assignment together with an ensemble of different online fitness landscape analysis, but this approach demanded high computational costs. From the practical perspective, the basic strategy adopted in PCHO was chosen because it is simple and it does not require excessive runtime.

4. Results and discussion

Four threads were employed to test PCHO: one was dedicated to the Master and the other 3 were assigned to the Workers Domain. Based on their intrinsic features and their potential for complementary behaviour, GA, PSO and SA have been adopted to provide the strengths and weaknesses of the hyper-heuristics. In this work the algorithm was always run on a PC with an Intel Core i7-4700 processor and 12 GB of RAM.

4.1. Representative examples

Nowadays, the availability of metaheuristics and parallel programming opens an opportunity to address and solve challenging PSE problems previously thought intractable. This paper focuses on the development of a method to find solutions efficiently for NLP problems of continuous-variable optimization (Eq. 1) frequently encountered on the PSE field. Being a preliminary study of the strategy, its scope does not comprise integer variables or large-scale problem testing.

Biegler (2010) summarized some applications that are commonly formulated in PSE by using an NLP model. It is employed to handle rigorous design and synthesis stages, mainly related to heat exchangers, mass exchangers, separations, reactors and flowsheeting. NLP also facilitates process building, real-time optimization and nonlinear model predictive control.

Over the past 50 years optimization has been a major technology that helps the chemical industry to remain competitive. In consequence, it has deeply been studied to be applied in research, process design and development stages, online operations and process control. In this context, rigorous specific collections of test problems were carefully compiled. Therefore, Aggarwal and Floudas (1990), Floudas and Pardalos (1990), Adjiman et al. (1998) and Hock and Schittkowski (1981) were the sources where the test problems for our evaluation were extracted from.

The problems that were selected to test PCHO are summarized on Table 1, where #Prob is the numbering assigned to each problem in this paper, #OptV is the number of optimization variables, #EqC and #IneqC are the number of equality and inequality constraints, respectively. Ref is the corresponding bibliographic reference, where the problem is reported with the number given in brackets. The last column punctuates the main features that distinguish each problem, making them eligible for the tests. Problems 1-36 are academic problems whose theoretical solution is known, whereas the remaining ones correspond to real-world practical problems whose designated "best solution" is the best empirical solution obtained so far.

As NLP problems based on real-life situations are often unavailable or too large, well-known problems coming from the state-of -the-art compilations in the PSE field were chosen for testing purposes. Although these collections were compiled a long time ago, they are still valid from a mathematical viewpoint because the test problems represent most of the numerical difficulties that may arise in practice. For example, badly scaled variables, non-convex model functions, ill-conditioned optimization problems, non-regular solutions at points where the constraint qualification is not satisfied, different local solutions or infinitely many solutions.

4.2. Numerical experiments

First of all, some experiments were carried out in order to compare the initial strategy, where only the winning agent modifies his initial point without retuning his method, with the improved learning strategy implemented in PCHO, where a tie is considered. When the initial strategy was applied to 51 problems, 19 test cases reached the best-known solution. For the rest of them, the algorithm could reach a feasible non-optimal solution. Only two of those cases approached the optimum (i.e., $d \leq 1$). Then, an improvement proved to be necessary. Table 2 shows a performance comparison between both policies. The improved strategy contributed to augment the number of solved problems to 29, while for other 7 problems the solutions were feasible points located close to the optimum ($d \leq 1$). In this analysis the effectiveness of the designed algorithm was measured in terms of how it responds to user demand. On the one hand, when the target was to reach optimality, the algorithm with its improved strategy showed 56.86 % effectiveness. In the rest of the problems (43.14%), the point that was found was not the optimal one, but it was feasible.

On the other hand, when the user decides to interrupt the execution as soon as a feasible point appears for the first time, there is an 80% reduction in the iteration number, being significantly reflected in the execution times. For 48 test problems, Fig. 5 shows the impact of an early stop when a feasible individual has been reached. The percentages given on the right-hand side of the graph were calculated with respect to the 30 iterations executed by the hyper-heuristic algorithm. In all cases its search always found the feasible point in fewer than 40% of the iterations. Then, significant CPU savings are achieved since there is at least a 60% reduction in the iteration number. It can also be observed that for more than half of the test problems, the first time the feasible point was reached in about 5% of the iterations, which reveals a fast successful performance. Moreover, it is important to remark that providing just a feasible point is also desirable because it may be helpful as an initialization to any other solver whose strength is the exploitation of particular features of certain problems, though they need a satisfactory starting point.

To compare algorithmic effectiveness, two performance indicators were adopted. The first one is a distance d that shows the closeness of the final result to the best solution. Whenever the algorithm has found either the best solution or a feasible one that is lying close to the optimum value, the degree of successfulness is measured by means of d. Closeness is assessed with the Euclidean norm of the difference between the best-known solution x_{best}, which has been reported in the literature, and the attained result x_{att} yielded by the algorithm under study. (Eq. (2)) This indicator can be applied only if the optimum value is well-known, thus requiring this feature in the test problems.

$$d = \|x_{best} - x_{att}\|_2 \tag{2}$$

The other performance indicator analyzed in this paper is the fitness gap that evaluates the solution quality in terms of the relative error of the attained fitness value f_{att}., with respect to the best known fitness value f_{best}. (Eq. 3)

$$fitness\ gap = \frac{f_{att} - f_{best}}{|f_{best}|} \tag{3}$$

Table 1
References and main features of some problems evaluated in the algorithmic tests.

#Prob	#OptV	#EqC	#IneqC	Ref (Problem #)	Main features
1-2-3	2	-	3-4-4	H&S (15-16-17)	Non-convex, generalized polynomial, theoretic
4	2	-	4	H&S (18)	Non-convex, quadratic, theoretic.
5	2	-	6	H&S (19)	Convex, generalized polynomial, theoretic
6	2	-	5	H&S (20)	Non-convex, generalized polynomial, theoretic
7	2	-	5	H&S (21)	Convex, quadratic, theoretic
8	2	-	2	H&S (22)	Convex, quadratic, theoretic
9	2	-	9	H&S (23)	Non-convex, quadratic, theoretic.
10	2	-	5	H&S (24)	Non-convex, generalized polynomial, theoretic
11-12	3	1	-	H&S (26-27)	Non-convex, generalized polynomial, theoretic
13	3	1	-	H&S (28)	Convex, quadratic, theoretic
14	3	-	1	H&S (29)	Non-convex, generalized polynomial, theoretic
15	3	-	7	H&S (30)	Non-convex, quadratic, theoretic.
16	3	-	7	H&S (31)	Convex, quadratic, theoretic
17	3	1	4	H&S (32)	Non-convex, quadratic, theoretic.
18	3	-	6	H&S (33)	Non-convex, generalized polynomial, theoretic
19	3	-	8	H&S (34)	Convex, generalized polynomial, theoretic
20	3	-	7	H&S (36)	Convex, generalized polynomial, theoretic
21-22-23-24	3-4-4-4	0-2-3-1	8-0-0-8	H&S (37-39-40-41)	Non-convex, generalized polynomial, theoretic
25	4	2	-	H&S (42)	Non-convex, quadratic, theoretic.
26	4	-	10	H&S (44)	Quadratic, linearly constrained, theoretic
27	5	-	10	H&S (45)	Nonlinear, bound-constrained, theoretic
28	5	3	-	H&S (47)	Non-convex, generalized polynomial, theoretic
29	5	2	-	H&S (48)	Convex, quadratic, theoretic
30-31	5	2-3	-	H&S (49-50)	Generalized polynomial, linearly constr, theoretic
32	5	3	-	H&S (51)	Non-convex, quadratic, theoretic
33	5	3	-	H&S (52)	Non-convex, generalized polynomial, theoretic
34	5	3	10	H&S (53)	Convex, quadratic, theoretic
35-36	6-7	6-4	8-0	H&S (55-56)	Non-convex, generalized polynomial, theoretic
37-39	3	1	6	H&S (60-62)	Non-convex, generalized polynomial, practical
38-40	3	2	0-3	H&S (61)	Non-convex, quadratic, practical
41	3	-	5	H&S (55)	Non-convex, generalized polynomial, practical
42	3	-	1	H&S (35)	Path optimization
43	4	-	10	H&S (72)	Optimal sample size.
44	4	3	8	H&S (68)	Cost optimal inspection plan.
45	5	3	10	Adj&Flou	Small reactor network design.
46	8	-	22	H&S (106)	Heat exchanger design
47	9	4	20	Adj&Flou	Haverly's pooling problem.
48	10	3	10	H&S (112)	Chemical equilibrium.
49	38	32	-	Flou&Par	Blending, pooling, separation problem.
50	48	38	-	Flou&Par	Propane, isobutane, n-butane nonsharp separation.
51	111	62	-	Agg&Flou	Separation network synthesis.

*Ad&Flou (Adjiman et al., 1998), Flou&Par (Floudas and Pardalos, 1990), Agg&Flou (Aggarwal and Floudas, 1990), H&S (Hock and Schittkowski, 1981).

Table 2
Performance analysis: percentages of successful executions.

Strategy	Solved	# of Feasible solutions (distance ≤ 1)		# of Feasible solutions (distance ≥ 1)	
		#	%	#	%
Initial	19	2	12.5	27	35.42
Improved	29	7	13.72	15	29.41

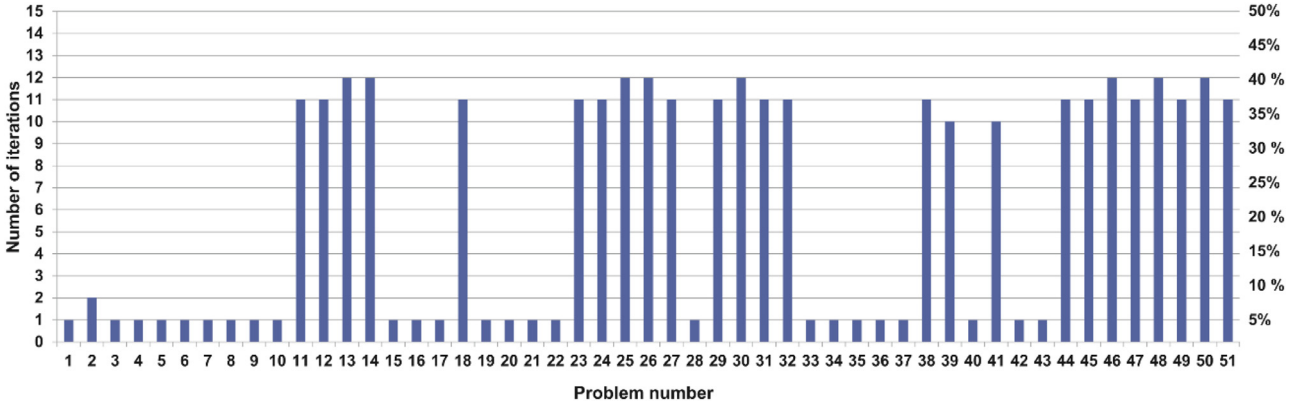


Fig. 5. Performance analysis: iterations required to find a feasible point for the first time.

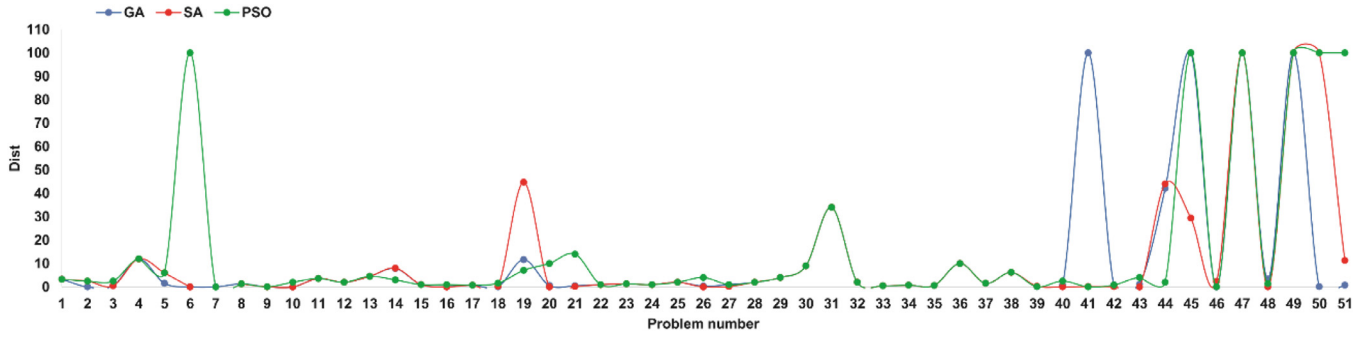


Fig. 6. Distances achieved by all test problems.

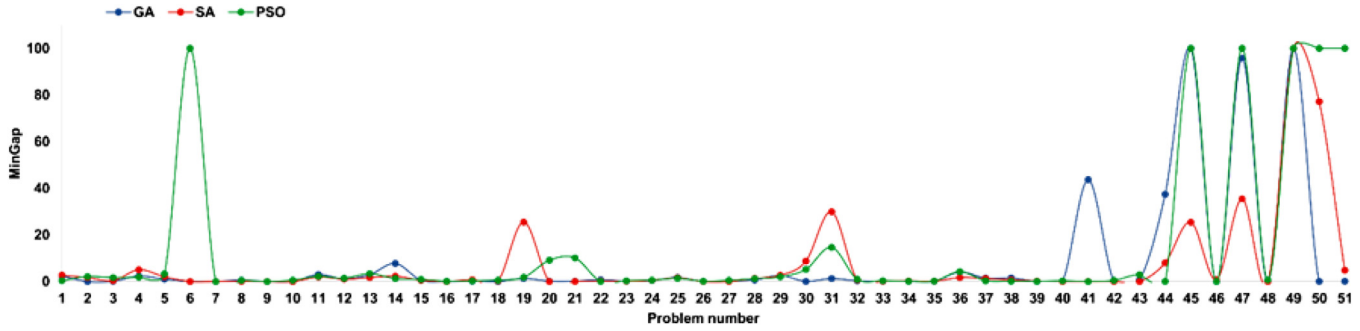


Fig. 7. Fitness minimum gaps for all test problems.

Table 3
Some statistics concerning performance indicators.

	GA	SA	PSO	PCHO
#Prob with d<1	25	23	27	32
#Prob with minGap<1	32	29	39	39

For each test instances both indicators were evaluated. The distances and minimum fitness gap achieved by the standalone metaheuristics and PCHO are shown in Figs. 6 and 7, respectively. Distances tend to zero and the corresponding gaps are small in most cases, thus revealing a satisfactory behaviour.

Table 3 summarizes the performance comparison between PCHO and the standalone algorithms. The number of solved problems that reached the theoretically optimal solution ($d < 1$) is smaller than the amount that has reached a minimum gap lower than 1. Nevertheless, exhibiting a low minimum gap reveals that the problem has yielded a feasible solution. For certain particular problems, the availability of feasible solutions is of utmost importance because there are dedicated solvers whose efficiency strongly depends on a good initial point. Therefore, a feasible point becomes remarkably valuable in order to initialize other solvers. These results, together with the performance exhibited by each independent (standalone) solver, show the improvement produced by an algorithm where the agents act collaboratively, sharing the information obtained.

Fig. 8 shows the disaggregated execution times, whereas in Fig. 9 average runtimes were grouped according to problem size. It can be noticed that the execution time is higher in PCHO, despite having been reduced by the action of parallelism. The balance between solution quality and execution time yields a

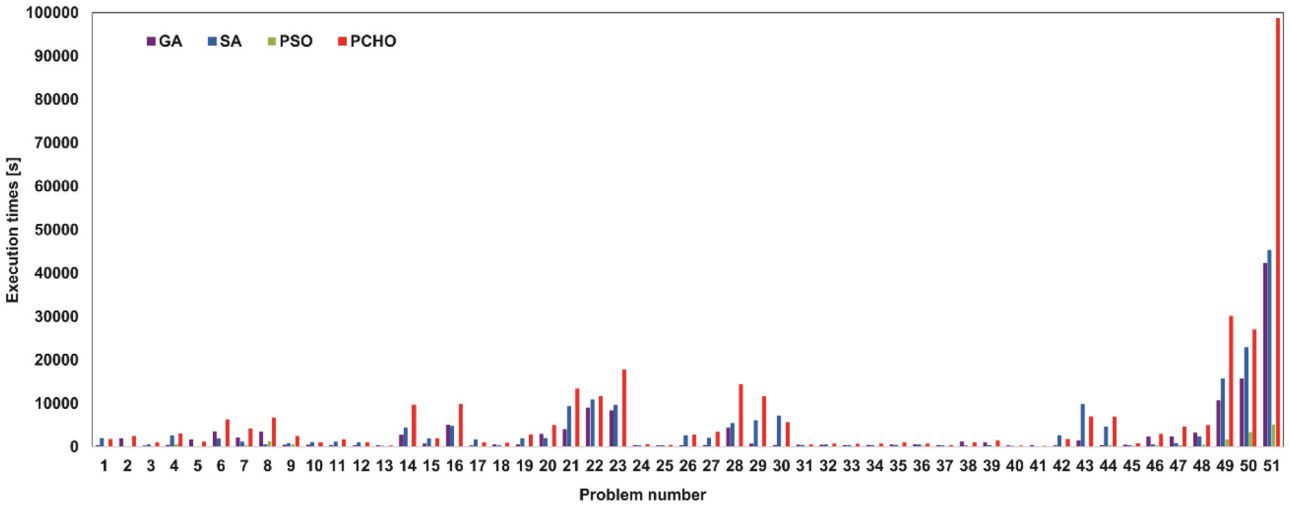


Fig. 8. Comparison of execution times between PCHO and the standalone sequential metaheuristics.

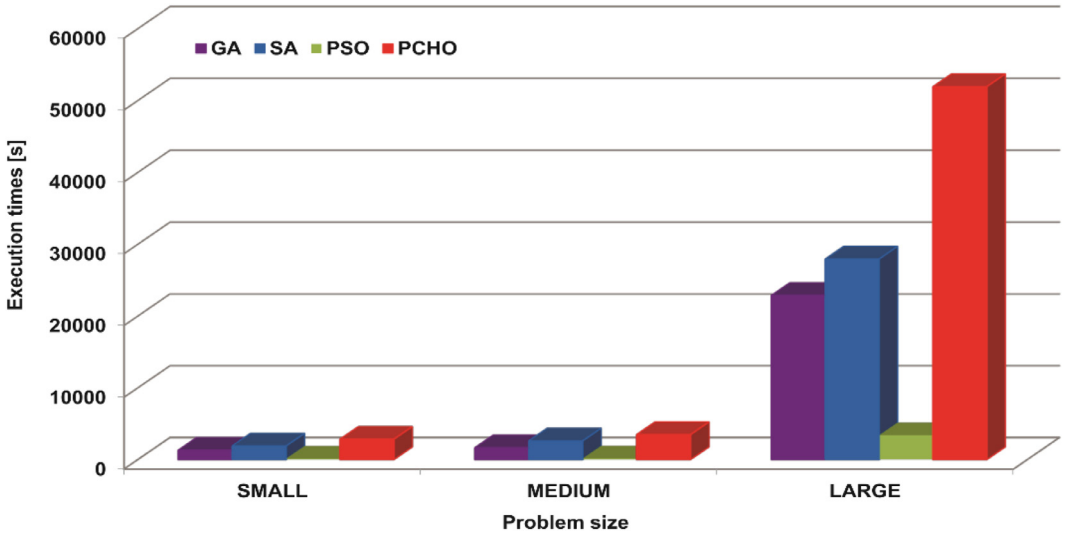


Fig. 9. Execution times for all test problems.

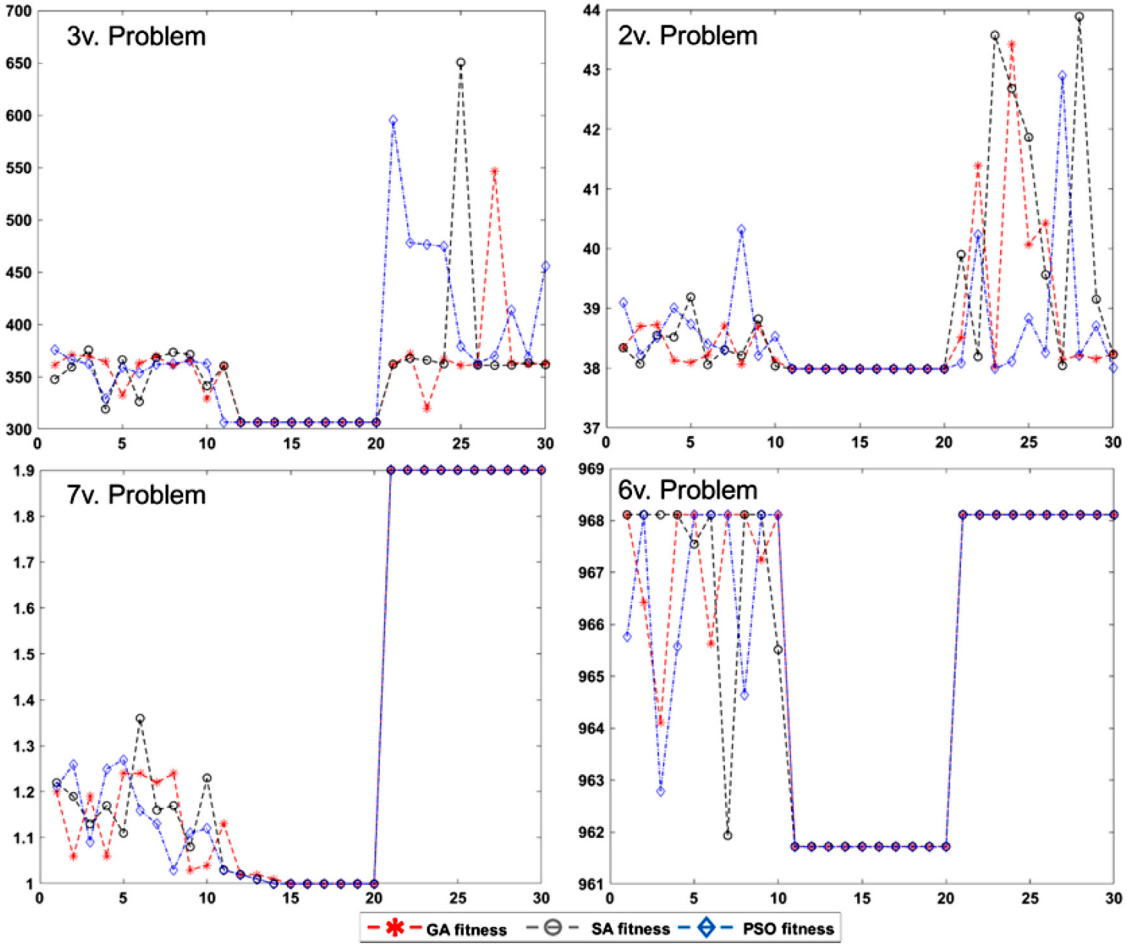


Fig. 10. Disaggregated algorithmic performance: best fitness vs. number of executions.

performance with slightly higher times, but on account of more satisfactorily solved problems. When compared to the sequential version of the standalone metaheuristics, the improvement in the quality of the solutions of the PCHO approach justifies the use of PCHO. It should be taken into account that the PCHO approach allows a larger number of neighborhoods to be simultaneously explored, instead of extensively exploring a single one in a sequential way, as in the standalone versions. In consequence, the PCHO algorithm makes it possible to explore the space of solutions more efficiently.

Fig. 10 shows the optimization results for 2v, 3v, 6v and 7v Problems, which have 2, 3, 6 and 7 optimization variables, respectively. In all cases the goal was to minimize fitness applying PCHO with the improved learning strategy. The best-known solution

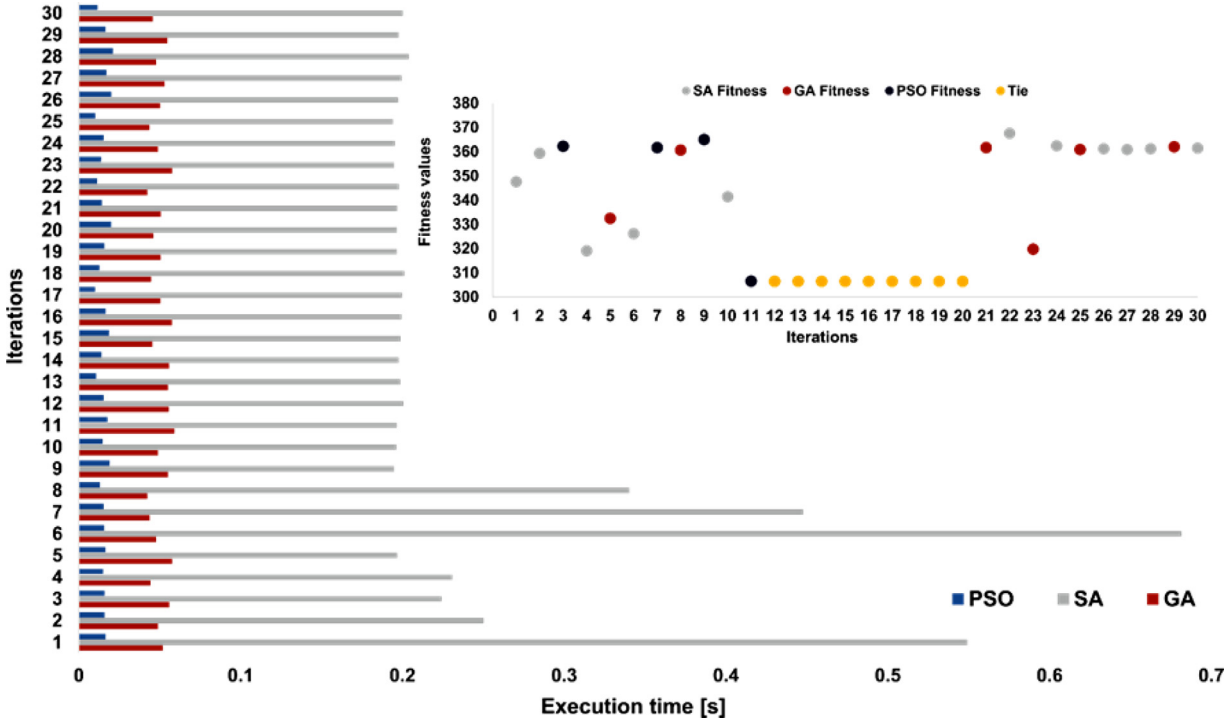


Fig. 11. Disaggregated execution times and the corresponding fitness values for Betts' Problem.

reported in the literature was satisfactorily reached. The best fitness values for all the agents are displayed for each execution. In these plots a braid has ensued, thus representing competitiveness. All the agents become competitive while the algorithm is running. How the learning strategy works can be noticed whenever the fitness value falls down suddenly overtaking the other agents. For example, in iteration 4 for problem 6V, GA exhibited his worst performance, whereas in the next iteration GA became the winner after a judicious retuning of his parameters. The braid behavior shows that all agents compete and learn, jumping out towards distinct fitness values to cope with different situations in the search space. Nevertheless, according to the results for all these problems, the best fitness value was reached between executions 10 and 20.

By way of illustration, let us compare individual execution times spent by SA with those consumed by the other agents (i.e. GA and PSO). Fig. 11 summarizes the performance of SA, GA, and PSO during a complete execution of PCHO for the theoretical problem defined in Eq. (4) (Betts, 1977). For each of the 30 iterations executed by the hyper-heuristic algorithm, the bars show the times taken by each agent to find a proper solution candidate. It can be noticed that GA and PSO, which are population-based metaheuristics, have outperformed SA. Although SA, which is a single-point method, may demand fewer function evaluations, a high computational time is not so unusual. Comparative reviews where SA is contrasted with other metaheuristics show that SA efficiency for problems with a very large search domain are sometimes accompanied by higher investments in computational time (Owa, 2017, Drake et al. 2020). When facing either a non-convex problem in its objective function or a non-convex feasibility region, the optimizer performance may suffer (Rafique, 2012) and be negatively affected. Hence, further testing became necessary to check whether SA stagnates or gets stuck in a local solution, being unable to correct its trajectory.

For Betts' Problem its starting point is (-2,1) and its solution point is (0.5, 2). It should be noted that this example is a non-convex problem with a non-convex feasibility región, whose

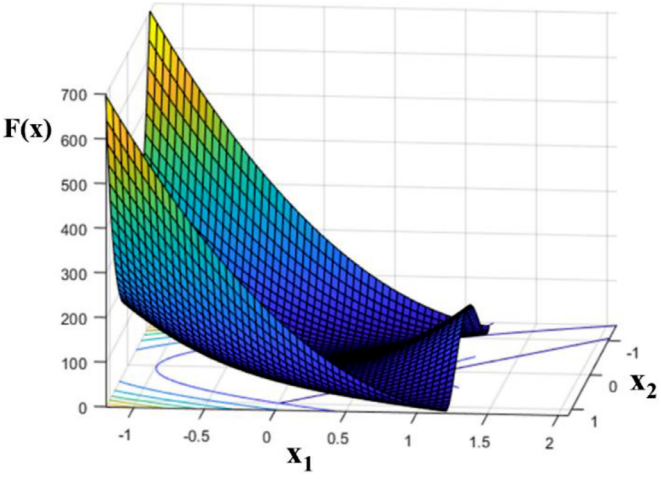


Fig. 12. Objective function for Betts' Problem.

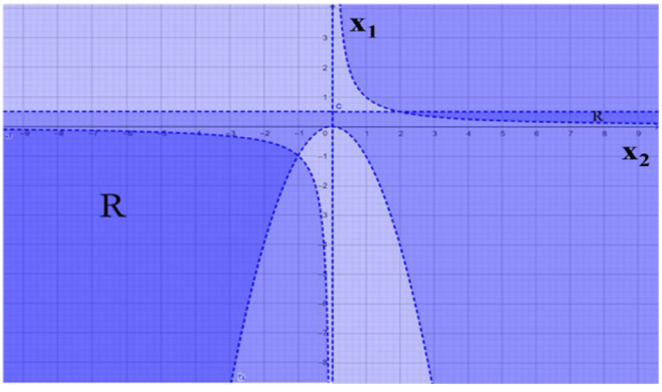


Fig. 13. Feasibility region for Betts' Problem. This zone is depicted in dark blue.

Table 4
Average execution times expressed in seconds.

Size	Sequential CPU mean time (s)	Parallel CPU mean time (s)
Small (2 - 3 variables)	248.9	204.1
Medium (4 - 10 variables)	696.1	560.6
38 variables	3010.3	2443.6
48 variables	3489.5	2785.6
111 variables	9874.8	7820.7

geometry is illustrated in Figs. 12 and 13.

$$\begin{aligned} &\min_{s.t.} 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2 \\ &x_1 x_2 - 1 \geq 0 \\ &x_2^2 + x_1 \geq 0 \\ &x_1 \leq 0.5 \end{aligned} \tag{4}$$

Betts' problem shows the potential of the balancing control between computational time control and solution quality. Although SA consumed much longer time than the others in the sixth iteration, it contributed with the best solution at that stage (see Fig. 8). All in all, SA, whose computational time elevated the general time average for each iteration, provided the best solution in 20 iterations (i.e., 66.7% of successful hits) being in 11 times the unique winner. In this case, premature detention of SA would have been detrimental to a solution that would have enriched the entire PCHO procedure. In this case, stopping SA during its search would have been a wrong policy since it would have acted in detriment of solution quality. Hence, it is unadvisable to impose a time limit to any agent. Though there was higher computational time demanded by SA, in many iterations SA yielded the most promising individuals with the highest fitness values.

Table 4 shows the average of the CPU times required by 51 test problems. For this analysis, twenty-seven small-size problems (i.e. comprising between 2 and 3 optimization variables), eighteen medium-size problems (i.e. ranging from 4 to 10 optimization variables) and three large cases, which respectively involved 38, 48 and 111 degrees of freedom, were evaluated. In general terms, the coarse-grain parallel implementation led to a 20% average time reduction, yielding a speed-up of 0.3125. As problem size increases, the parallel method runs faster than the sequential one. Therefore, the empirical results show that parallel programming is of practical interest, especially for large problems. If more efficiency were desired, it is possible to increase the attainable speed-up by refining task distribution among workers.

5. Conclusions

The general framework of a parallel hyper-heuristic optimizer was developed and implemented. It solves predefined problems by means of various metaheuristics working collectively and cooperatively. It was tested by using GA, SA and PSO, which contributed to the search with their intrinsic features. The results provided by each agent are administrated by means of an innovative learning mechanism, where improvements of the optimizer intelligence are achieved by including self-adaptive features in every metaheuristics. For future work it would be challenging to explore the incorporation of novel parameter tuning techniques into PCHO, investigating how they may affect PCHO's efficiency and data storage.

A set of tests that are representative of PSE problems was built in order to evaluate algorithmic performance. The hyper-heuristics proved to be accurate and efficient due to the incorporation of self-adaptability. Future studies should focus on evaluating more test functions, thus introducing more variety in the mathematical models in order to investigate the scope of this proposal.

This algorithm is flexible because it is automatically configurable to capture problem-specific features concerning runtimes. By tailoring metaheuristic performances to particular instances, it effectively boosts the applicability of the proposed hyper-heuristics.

In chemical engineering applications, this topic is nowadays open for active research. There are plenty of opportunities in Industry 4.0, mainly regarding process control and plant-wide design that might profit from the employment of hyper-heuristic methods. The hyper-heuristic strategy is innovative as a modelling tool. An adequate ad-hoc implementation of this approach might enrich the optimization platforms of traditional PSE packages in order to contribute to next-generation advanced process modelling environments. Focusing on an efficient optimizer to tackle with time-demanding problems, it is worthwhile exploring how to hybridize traditional equation-oriented methods with the hyper-heuristic approach presented here, so that the new solver helps to find improved solutions and/or initializations in competitive CPU times. Moreover, since the prediction of properties can also be posed as problems of error minimization, the hyper-heuristic approach boosted by automatic configuration of metaheuristic agents might be useful as an efficient computational tool to provide accurate predictions in this field.

Declaration of Competing Interest

None.

CRediT authorship contribution statement

Paola P. Oteiza: Investigation, Visualization, Resources. **Juan I. Ardenghi:** Investigation, Software, Conceptualization. **Nélida B. Brignole:** Investigation, Supervision, Writing – original draft, Writing – review & editing.

References

Abdelhafez, A., Luque, G., Alba, E., 2020. Parallel execution combinatorics with metaheuristics: comparative study. Swarm Evol. Comput. 55, 100692. doi:10.1016/j.swevo.2020.100692.

Abdelhafez, A., Luque, G., Alba, E., 2019. Analyzing the energy consumption of sequential and parallel metaheuristics. In: 2019 International Conference on High Performance Computing & Simulation (HPCS). IEEE, pp. 121–128. doi:10.1109/HPCS48598.2019.9188170.

Adjiman, C.S., Androulakis, I.P., Floudas, C.A., 1998. A global optimization method, αBB, for general twice-differentiable constrained NLPs-II. Implementation and computational results. Comput. Chem. Eng. 22 (9), 1159–1179. doi:10.1016/S0098-1354(98)00218-X.

Aggarwal, A., Floudas, C.A., 1990. Synthesis of general distillation sequences—nonsharp separations. Comput. Chem. Eng. 14 (6), 631–653. doi:10.1016/0098-1354(90)87033-L.

Alba, E., Luque, G., Nesmachnow, S., 2013. Parallel metaheuristics: recent advances and new trends. Int. Trans. Oper. Res. 20 (1), 1–48. doi:10.1111/j.1475-3995.2012.00862.x.

Asta, S., Özcan, E., Parkes, A.J., Etaner-Uyar, A.Ş., 2013. Generalizing hyper-heuristics via apprenticeship learning. In: European Conference on Evolutionary Computation in Combinatorial Optimization, 7832, Berlin, Heidelberg. Springer, pp. 169–178. doi:10.1007/978-3-642-37198-1_15.

Bäck, T., Schwefel, H.P., 1993. An overview of evolutionary algorithms for parameter optimization. Evol. Comput. 1 (1), 1–23. doi:10.1162/evco.1993.1.1.1.

Betts, J.T., 1977. An accelerated multiplier method for nonlinear programming. J. Optim. Theory Appl. 21 (2), 137–174. doi:10.1007/BF00932517.

Biegler, L.T., 2010. Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes. SIAM, Philadelphia, USA doi:10.1137/1.9780898719383.

Bonyadi, M.R., Michalewicz, Z., Wagner, M., Neumann, F., 2019. Evolutionary computation for multicomponent problems: opportunities and future directions. In: Datta, S., Davim, J. (Eds.), Optimization in Industry. Management and Industrial Engineering. Springer, Cham, Switzerland.

Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.R., 2019. A classification of hyper-heuristic approaches: revisited. In: Handbook of Metaheuristics. Springer, Cham, Switzerland, pp. 453–477. doi:10.1007/978-3-319-91086-4_14.

Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R., 2013. Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Soc. 64 (12), 1695–1724. doi:10.1057/jors.2013.71.

Cheng, J.R., Gen, M., 2019. Accelerating genetic algorithms with GPU computing: a selective overview. Comput. Ind. Eng. 128, 514–525. doi:10.1016/j.cie.2018.12.067.

Coelho, P., Silva, C., 2021. Parallel metaheuristics for shop scheduling: enabling industry 4.0. Procedia Comput. Sci. 180, 778–786. doi:10.1016/j.procs.2021.01.328.

Consoli, P.A., Mei, Y., Minku, L.L., Yao, X., 2016. Dynamic selection of evolutionary operators based on online learning and fitness landscape analysis. Soft Comput. 20 (10), 3889–3914. doi:10.1007/s00500-016-2126-x.

Drake, J.H., Kheiri, A., Özcan, E., Burke, E.K., 2020. Recent advances in selection hyper-heuristics. Eur. J. Oper. Res. 285 (2), 405–428. doi:10.1016/j.ejor.2019.07.073.

Fisher, H., 1963. Probabilistic learning combinations of local job-shop scheduling rules. Ind. Scheduling 225–251.

Floudas, C.A., Pardalos, P.M., 1990. A Collection of Test Problems for Constrained Global Optimization Algorithms. Springer Science & Business Media, p. 455.

Glover., F, 1986. Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res. 13 (5), 533–549. doi:10.1016/0305-0548(86)90048-1.

Hock, W., Schittkowski, K., 1981. Test Examples for Nonlinear Programming Codes. Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, Berlin.

Huang, C., Li, Y., Yao, X., 2019. A survey of automatic parameter tuning methods for metaheuristics. IEEE Trans. Evol. Comput. 24 (2), 201–216. doi:10.1109/TEVC.2019.2921598.

Jackson, W.G., Özcan, E., John, R.I., 2017. Tuning a simulated annealing metaheuristic for cross-domain search.. In: 2017 IEEE Congress on Evolutionary Computation (CEC), pp. 1055–1062. doi:10.1109/CEC.2017.7969424.

Karafotias, G., Hoogendoorn, M., Eiben., A.E, 2014. Parameter control in evolutionary algorithms: trends and challenges. IEEE Trans. Evol. Comput. 19 (2), 167–187. doi:10.1109/TEVC.2014.2308294.

Kleinert, T., Schmidt, M., 2021. Computing feasible points of bilevel problems with a penalty alternating direction method. INFORMS J. Comput. 33 (1), 198–215. doi:10.1287/ijoc.2019.0945.

Lalwani, S., Sharma, H., Satapathy, S.C., Deep, K., Bansal, J.C., 2019. A survey on parallel particle swarm optimization algorithms. Arabian J. Sci. Eng. 44 (4), 2899–2923. doi:10.1007/s13369-018-03713-6.

Lee, S., Kim, S.B., 2019. Parallel simulated annealing with a greedy algorithm for bayesian network structure learning. IEEE Trans. Knowl. Data Eng. 32 (6), 1157–1166. doi:10.1109/TKDE.2019.2899096.

Lhotská, L., Macaš, M., Burša, M., 2006. PSO and ACO in optimization problems. In: International Conference on Intelligent Data Engineering and Automated Learning. Springer, pp. 1390–1398. doi:10.1007/11875581_165.

Liu, E., Lv, L., Yi, Y., Xie, P., 2019. Research on the steady operation optimization model of natural gas pipeline. IEEE Access 7, 83251–83265. doi:10.1109/ACCESS.2019.2924515.

Luna Valero, F., 2008. Metaheurísticas avanzadas para problemas reales en redes de telecomunicaciones. Malaga University.

Mahdavi, S., Shiri, M.E., Rahnamayan, S., 2015. Metaheuristics in large-scale global continues optimization: a survey. Inf. Sci. 295, 407–428. doi:10.1016/j.ins.2014.10.042.

Nguyen, S., Zhang, M., 2017. A PSO-based hyper-heuristic for evolving dispatching rules in job shop scheduling. In: 2017 IEEE congress on evolutionary computation (CEC), pp. 882–889. doi:10.1109/CEC.2017.7969402.

Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J.A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A.J., Petrovic, S., Burke, E.K., 2012. Hyflex: a benchmark framework for cross-domain heuristic search. In: European Conference on Evolutionary Computation in Combinatorial Optimization, Berlin, Heidelberg. Springer, pp. 136–147. doi:10.1007/978-3-642-29124-1_12.

Ortiz-Bayliss, J.C., Terashima-Marín, H., Conant-Pablos, S.E., 2016. Combine and conquer: an evolutionary hyper-heuristic approach for solving constraint satisfaction problems. Artif. Intell. Rev. 46 (3), 327–349. doi:10.1007/s10462-016-9466-x.

Oteiza, P.P., Rodriguez, D.A., Brignole, N.B., 2018. A parallel hyper-heuristic algorithm for the design of pipeline networks. Ind. Eng. Chem. Res. 57 (42), 14307–14314. doi:10.1021/acs.iecr.8b02818.

Owa, K., Jackson, L., Jackson, T., 2017. An intelligent novel tripartite-(PSO-GA-SA) optimisation strategy. Int. J. Metaheuristics 6 (3), 210–233. doi:10.1504/IJMHEUR.2017.085125.

Pellerin, R., Perrier, N., Berthaut, F., 2020. A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. Eur. J. Oper. Res. 280 (2), 395–416. doi:10.1016/j.ejor.2019.01.063.

Rafique, A.F., 2012. Multiobjective hyper heuristic scheme for system design and optimization. Am. Inst. Phys. (AIP) Conf. Proc. 1493 (1), 764–769. doi:10.1063/1.4765574.

Ramos-Figueroa, O., Quiroz-Castellanos, M., Mezura-Montes, E., Schütze, O., 2020. Metaheuristics to solve grouping problems: a review and a case study. Swarm Evol. Comput. 53, 100643. doi:10.1016/j.swevo.2019.100643.

Schewe, L., Schmidt, M., 2019. Computing feasible points for binary MINLPs with MPECs. Math. Programm. Comput. 11 (1), 95–118. doi:10.1007/s12532-018-0141-x.

Stützle, T., López-Ibáñez, M., 2019. Automated design of metaheuristic algorithms. In: Handbook of Metaheuristics. Springer, Cham, Switzerland, pp. 541–579. doi:10.1007/978-3-319-91086-4_17.

Ting, T.O., Yang, X.S., Cheng, S., Huang, K., 2015. Hybrid metaheuristic algorithms: past, present, and future. Recent Adv. Swarm Intell. Evol. Comput. 71–83. doi:10.1007/978-3-319-13826-8_4.

Vikhar, P.A., 2016. Evolutionary algorithms: a critical review and its future prospects. In: 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), pp. 261–265. doi:10.1109/ICGTSPICC.2016.7955308.

Wang, Z., Zhao, Y., Liu, Y., Lv, C., 2018. A speculative parallel simulated annealing algorithm based on Apache Spark. Concurr. Comput. 30 (14), e4429. doi:10.1002/cpe.4429.

Yang, X.S., 2015. Recent Advances in Swarm Intelligence and Evolutionary Computation. Springer International Publishing.