The final publication is available at Elsevier via https://doi.org/10.1016/j.asoc.2019.01.012

Enhanced global optimization methods applied to complex fisheries stock assessment models

David R. Penas[a], Andrés Gómez[b], Basilio B. Fraguela[c], María J. Martín[c], Santiago Cerviño[d]

[a]MODESTYA research group, Department of Statistics, Mathematical Analysis and Optimization, and Institute of Mathematics, IMAT, Universidade de Santiago de Compostela, Santiago de Compostela, Spain
[b]Galician Supercomputing Center (CESGA), Santiago de Compostela, Spain
[c]Computer Architecture Group (GAC), Universidade da Coruña, Spain
[d]Spanish Institute of Oceanography (IEO), Vigo, Spain

Abstract

Statistical fisheries models are frequently used by researchers and agencies to understand the behavior of marine ecosystems or to estimate the maximum acceptable catch of different species of commercial interest. The parameters of these models are usually adjusted through the use of optimization algorithms. Unfortunately, the choice of the best optimization method is far from trivial. This work proposes the use of population-based algorithms to improve the optimization process of the Globally applicable Area Disaggregated General Ecosystem Toolbox (Gadget), a flexible framework that allows the development of complex statistical marine ecosystem models. Specifically, parallel versions of the Differential Evolution (DE) and the Particle Swarm Optimization (PSO) methods are proposed. The proposals include an automatic selection of the internal parameters to reduce the complexity of their usage, and a restart mechanism to avoid local minima. The resulting optimization algorithms were called PMA (Parallel Multirestart Adaptive) DE and PMA PSO respectively. Experimental results prove that the new algorithms are faster and produce more accurate solutions than the other parallel optimization methods already included in Gadget. Although the new proposals have been evaluated on fisheries models, there is nothing specific to the tested models in them, and thus they can be also applied to other optimization problems. Moreover, the PMA scheme proposed can be seen as a template that can be easily applied to other population-based heuristics.

Keywords: Global optimization, Parallel programming, Marine ecosystem models, Particle Swarm Optimization, Differential Evolution

1. Introduction

Statistical fisheries models are an essential tool to understand the behavior of marine ecosystems and to develop simulations of future scenarios that allow the adoption of management measures for the sustainable exploitation of marine resources. Developing this kind of models is usually an iterative task that involves a parameter estimation process in which the output provided by the model is compared with observed real data, obtaining a fitness value that represents how well the model simulates the target ecosystem. Thus, the parameters of the model are iteratively adjusted to improve the fitness value and obtain a more accurate model.

The search for the best parameters can be formulated as a mathematical nonlinear optimization problem. Identifying the best optimization algorithm in this context is not trivial, as the search space usually contains non-linearities, multimodality and non-convexity, being computationally expensive to reach a good solution [1, 2]. Stochastic methods such as heuristics or metaheuristics can be used to return a solution near to the global optimum in a reasonable computational time. Even so, usually these types of methods are still time-consuming for some cases, or they have too many tunable parameters with a large impact on the behavior of the algorithm. To solve this problem, combinations of optimization methods have been tried to identify an optimal search procedure [3, 4, 5], although these works show that the best particular combination depends on the problem considered.

Gadget [6, 7, 8] (Globally applicable Area Disaggregated General Ecosystem Toolbox) is a flexible framework to aid in the development of statistical models of marine ecosystems. It is an open source tool, written in C++ that considers the impact related both to the fisheries harvesting and to the interactions between different species in a specific ecosystem. Gadget can take into account a great number of features such as: the existence of one or more species, migration between areas, predation, growth, maturation, reproduction, recruitment and commercial and scientific catches. Thus, it can be used to develop highly complex ecosystem models.

Gadget has been successfully applied in many case studies such as the Icelandic continental shelf area for the cod and redfish stocks [9, 10, 11], the Bay of Biscay in Spain to predict the evolution of the anchovy stock [12], the North East Atlantic to model the porbeagle shark stock [13], or the Barents Sea to model dynamic species interactions [14]. Moreover, other successful works were also modeled using Gadget, providing tactical advice on sustainable exploitation levels of a particular resource, such as southern European Hake stock, Icelandic Golden redfish, Tusk and Ling stocks, and Barent sea Greenland halibut, which are formally assessed by the International Council for the Exploration of the Sea (ICES) [15, 16, 17, 18].

The Gadget framework consists of three parts: a parametric model to simulate the ecosystem, statistical functions to compare the model output to the observed data, and optimization algorithms to adjust the model parameters. Currently, there are three different optimization methods included in Gadget: Simulated Annealing (SA) [19, 20], Hooke & Jeeves (HJ) [21], and Broyden-Fletcher-Goldfarb-Shanno (BFGS) [22]. The typical usage of Gadget involves combining these optimization methods in order to obtain the best possible result. Recently in [23], the SA and HJ algorithms have been parallelized using OpenMP [24] and speculative parallelization techniques, with the purpose of taking advantage of existing resources in current multi-core systems, and achieving improved results both in terms of execution time and accuracy of the solution.

Unfortunately, the optimization methods implemented in Gadget can still get stuck in local optima, which gives place to a large dispersion in the convergence time and in the quality of the results, being hard to obtain a solution near to the global optimum. This work proposes the use of two well-known optimization algorithms to solve this problem: an evolutionary method called Differential Evolution (DE) [25], and a particle-based method known as Particle Swarm Optimization (PSO) [26]. Both of them are population algorithms, which are more suitable for parallelization than the ones previously available in Gadget, and thus, they can more easily exploit the characteristics of the today's ubiquitous multicore systems. In addition, due to the large number of configuration parameters of these methods, the new algorithms were modified to auto-tune themselves at runtime, thus preventing the user from spending time in finding the best heuristic-parameter configuration. Furthermore, an additional mechanism of diversity, consisting in a random restart of the population when the algorithms do not evolve, was implemented to avoid that the methods get stuck.

The resulting optimization algorithms were called PMA (Parallel Multirestart Adaptive) DE and PMA PSO respectively, and they were applied to the optimization of three different models developed with Gadget, obtaining improved results both in terms of the quality of the solutions and the computational time required. The good performance of the proposals is due to the combination of the adaptation strategies with the restart mechanisms implemented.

This paper is organized as follows. Section 2 covers related work. Then, Section 3 describes the optimization problem to solve. Section 4 explains the original DE and PSO algorithms. Section 5 describes the modifications proposed, that is, the PMA DE and PMA PSO methods. Section 6 describes and discusses the results obtained when the proposed methods are applied to the optimization of three different Gadget models. This Section also includes comparisons with other DE and PSO algorithms. Finally, conclusions and future work are discussed in Section 7.

2. Related work

Works in the field of marine ecosystems modeling use different optimization techniques to obtain good predictions with the aim of guiding important decisions. An example is [27], where the parameters of nonlinear dynamic models of marine ecosystems were calibrated using simulated annealing. Another example is a more recent work [28], where also dynamic global optimization problems were handled by an optimization environment named MarMOT (Marine Model Optimization Testbed) to develop different plankton ecosystem models. In this work, a genetic algorithm called GFAn Optimizer was used to estimate the model parameters. It is also worth to mention the Stock Synthesis tool [29], which is based on the automatic differentiation framework called ADMD [30]. This tool is broadly used to solve stock assessment models, although depending on the model complexity the time to achieve convergence ranges from seconds to many hours.

As for the PSO and DE algorithms, Tashkova et al. [31] have performed a comparison among a local derivative-based method and four global meta-heuristic algorithms, including PSO and DE, for estimating the parameters of a non-linear model of an aquatic ecosystem. The study concludes that the meta-heuristic methods are clearly superior. Although the differences in performance among the different meta-heuristic methods are not significant, DE is the best solution for this problem.

Relatedly, many works propose self-adaptation schemes to optimize the PSO and DE algorithms [32, 33, 34, 35, 36, 37, 38], as the time spent in calibrating the configuration parameters in these heuristics methods is one of their most critical issues. Many methods as jDE [39, 40], SaDE [41], APSO [42] or CLPSO [43] are popular proposals of self-adaptive heuristics, which have great success and influence in the development of current evolutionary algorithms. These strategies usually implement a memory with the goal of self-learning what configuration parameters are the more suitable during the optimization process.

Regarding the parallelization, there are many proposals to parallelize via a distributed-memory paradigm both the DE [44, 45, 46, 47] and the PSO [48, 49, 50] algorithms. Besides, the parallelization in shared-memory, focusing on the case of OpenMP, has also been widely used in these optimization methods, such as in [51, 52, 53].

In this work we adapt the DE and PSO global optimization algorithms to work as optimization methods in Gadget, proposing parallelization, self-adaptation and multi-restart mechanisms to improve both the quality of the obtained solutions and the execution times.

3. Problem statement

Gadget allows to develop a parametric simulation model of a marine ecosystem that can explain the changes in the fish populations produced in a geographic area, such as growth or predation rate. After this, statistical functions can be used to compare the simulated data obtained with the observed data available to get a goodness-of-fit likelihood score that indicates how well the distribution from the model fits the distribution observed in the sampling process. Then, the parameters can be iteratively adjusted to improve the fit of the model with the observed data until an optimum is reached, this value corresponding with the lowest likelihood score in the model.

There are different kinds of statistical functions to calculate the fitness of the model with respect to the observed data, depending on the component of the ecosystem to analyze. Thus, in order to consider all the data together, the overall likelihood score is computed as a weighted sum of several likelihood scores. Therefore, the process of iteratively adjusting the model parameters can be formulated as the following global optimization problem:

$$minimize : L = \sum_{i=0}^{N} \ell_i(\{x_n\})\ \omega_i$$

where L is the overall likelihood function, ℓ_i is the likelihood function for the component i, ω_i is its associated weight, and $\{x_n\}$ is a set of parameters to be estimated in this likelihood component. Thus, each likelihood component has its own cost function and parameters to be estimated, which are subject to bound constraints.

Gadget also allows the use of likelihood components that apply a penalty to the overall likelihood score when an impossible situation arises. This happens for instance when "understocking" occurs, that is, when the combination of estimated parameters gives an insufficient number of a certain type of prey with respect to the requirements of predators.

The number of parameters of the models developed in Gadget is a design decision to be taken by the modelers, being closely related to the complexity of the problem to map: how many species interact in the ecosystem, how big is the area to study, or other similar details. In this work we have worked with models that contain 62, 47 and 38 parameters, covering only one species per model. Optimization problems obtained are already challenging. Thus, we have in mind that more challenging problems will appear as more complex models are developed, being important to design efficient optimization mechanisms to reduce the asociated computational cost.

A full description of each one of the available likelihood components can be found in [8]. The choice of the likelihood components used to develop a given model depends on the characteristics of the model and the data available. It is also worth to mention that Gadget supports the use of datasets from both commercial fisheries and scientific surveys.

4. Global optimization methods

As stated before, Gadget already has a set of optimization algorithms, such as Simulated Annealing (SA) or Hooke and Jeeves (HJ), from which excellent results have been obtained [23]. However, it is possible to obtain better results by using population-based heuristics due to properties such as their ability to operate with several solutions at once or their easily parallelizable scheme. In this work we consider two well-known population methods: Differential Evolution and Particle Swarm Optimization. The next subsections describe their algorithms in turn, followed by a brief discussion of two common issues in both methods: parameters selection and local optima.

4.1. Differential Evolution (DE)

DE is an evolutive method where, for each iteration, new solutions are created through difference operations. Starting from a population of pop_size randomly generated $nvars$-dimensional solutions within the bound constraints of the problem, the method creates, using a Mutation Strategy (MStr), a set of candidate solutions for each iteration, which are intended to replace the solutions stored in the population. These mutation operations represent the difference among two or more solutions randomly chosen from the population, multiplied by the so-called Mutation Factor (F). There are many types of mutation strategies, this work uses some of them:

- `DE/best/2/bin`:

$$new\_point_i \leftarrow xbest + F(p_a - p_b) + F(p_c - p_d) \tag{1}$$

- `DE/current-to-rand/1/bin`:

$$new\_point_i \leftarrow p_i + F(p_a - p_i) + F(p_b - p_c) \tag{2}$$

- `DE/current-to-best/1/bin`:

$$new\_point_i \leftarrow p_i + F(xbest - p_i) + F(p_a - p_b) \tag{3}$$

- `DE/rand-to-best/1/bin`:

$$new\_point_i \leftarrow p_a + F(xbest - p_a) + F(p_b - p_c) \tag{4}$$

where p_i is the current solution, p_a, p_b, p_c and p_d are solutions randomly selected belonging to the current population, $xbest$ is the current best solution of the population, and new_point_i is the candidate solution created.

Algorithm 1 shows the basic scheme of DE using the mutation strategy `DE/best/2/bin`. The mutation strategy is applied element by element in some of the $nvars$ positions of the solution. The crossover constant (CR) is a configuration parameter to decide how many positions of the candidate solution come from the original solution, and how many are the result of the mutation process. For each existing solution in the population, a new candidate is generated, but this candidate only replaces the solution it has been derived from if it is better. This process will be repeated until a specific stopping criterion is met. At that point, the best member of the pop_size population is obtained as output of the method.

Therefore, there are a lot configuration parameters in DE algorithms: population size, crossover constant, mutation factor and mutation strategy are the basic settings in this type of metaheuristic. Different combinations of these settings increase or decrease the performance depending of the optimization problem nature. Thus, a set of modifications of the classic scheme of DE have been emerged along last years, trying to improve the behaviour of this algorithm via implementing adaptive strategies. An example of them is jDE [39, 40], which proposed a self-tuning mechanism to control mutation and crossover parameters, assigning a pair F and Cr for each population member and adapting individually their value through adding random values. Other alternative very popular in DE community is SaDE [41], where the mutation strategies and an associated CR and F settings are gradually tuned, using the acquired knowledge in an initial learning state.

Algorithm 1: Differential Evolution.

input : P, a population of pop_size solutions
output: $xbest$, the best solution $\in$ P

repeat
  Select $xbest \in$ P;
  for $\forall$ *solution* $p_i \in P$ **do**
    p_a, p_b, p_c, $p_d \leftarrow$ different random individuals from P;
    $new\_p_i = p_i$;
    for *each dimension* $k \in p_i, 0 \leq k < nvars$ **do**
      $rand =$ random point between (0,1);
      if $rand < CR$ **then**
        `/* Mutation strategy DE/best/2/bin */`
        $new\_p_{i,k} = xbest_k + F(p_{a,k} - p_{b,k}) + F(p_{c,k} - p_{d,k})$;
      end
    end
    if *Evaluation*(new_p_i) *is better than Evaluation*(p_i) **then**
      $p_i = new\_p_i$;
    end
  end
until *Stop conditions*;

4.2. Particle Swarm Optimization (PSO)

The PSO algorithm also operates on a population of solutions, called swarm in this case. For each iteration new candidate solutions, named here particles, are generated applying movements through the search-space using both local knowledge of the particles and global information of the method.

Algorithm 2 shows the general scheme of PSO. At the beginning, *pop_size* $nvars$-dimensional particles that conform the initial swarm are randomly generated within the bound constraints of the problem. Then, in the main loop, a certain movement is applied to each particle through the following expressions:

$$v_i \leftarrow \omega v_i + \varphi_l r_l (pbest_i - p_i) + \varphi_g r_g (gbest - p_i) \tag{5}$$

$$p_i \leftarrow p_i + v_i \tag{6}$$

where p_i is the current solution for the particle i of the swarm, v_i is the vector velocity, ω is the inertia constant, φ_g is the social coefficient, φ_l is the cognitive coefficient, r_g and r_l are two random numbers between zero and one, $pbest_i$ is the local best solution found by the current particle up to this iteration, and $gbest$ is the best global solution found by any particle during the execution of the method. Using these two formulas, a new solution is created through the sum of the current position with the new velocity vector, which is generated taking into account the following three factors in Equation (5) in this order:

- Previous velocities and inertia weight: the velocity calculated for a particle "i" in past iterations is taken into account to compute the velocity in the current iteration. The inertial weight (ω) is a configurable parameter of the heuristic, typically with values between zero and one, to adjust the importance of this accumulation of speeds.

- Cognitive component: It is a vector calculated as the difference between the best value reached by the considered particle i and the current value of that particle. This vector takes into account the local knowledge about previous visited points. To adjust the impact of the influence of this cognitive component, it is multiplied by a random value between zero and one (this is what gives stochasticity to the method), and also by a configurable parameter called cognitive coefficient (φ_l).

- Social component: It is a vector computed as the difference between the global best solution found among all the particles from the swarm and the current particle solution. Therefore, the social component takes into account the influence of the global solution when new solutions are created. Also, for this case, the vector generated is multiplied by a random number between zero and one, and by a configurable parameter known as social coefficient (φ_g).

Then, if the new solution p_i improves over $pbest_i$, this best local solution is updated to p_i. Similarly, if p_i is better than the global best known solution $gbest$, the latter is updated to p_i. This process will be repeated until a specific stopping criterion is met. At that point, $gbest$ is obtained as output of the method.

Moreover, as in the case of the DE, there are many popular implementations of PSO, being the most promising that where the configuration parameters, such as swarm size, cognitive/social coefficient or inertia, are able to self-adapt their settings during the runtime. An example is CLPSO [43], a very popular modification of the algorithm, where a learning strategy is used to decide what local best positions placed in different particles are candidates to update the velocity of each solution, instead of the best local solution. Other option is APSO [42], a adaptive method able to learn about

Algorithm 2: Particle Swarm Optimization.

```
input  : P, a set of pop_size particles
output: gbest, the best particle ∈ P
repeat
    initialize P ← random generation of the particles;
    ∀ i, v_i = 0;
    pbest_i ← local best solution in position i;
    gbest ← global best solution;
    for ∀ particle p_i ∈ P do
        for each dimension k ∈ p_i, 0 ≤ k < nvars do
            r_p, r_g = random numbers generated between (0,1);
            v_{i,k} = ω v_{i,k} + φ_p r_p(pbest_{i,k} - p_{i,k}) + φ_g r_g(gbest_k - p_{i,k});
            p_{i,k} ← p_{i,k} + v_{i,k};
            if bound constraints are violated in p_{i,k} then v_{i,k} = 0;
        end
        if Evaluation(p_i) is better than Evaluation(pbest_i) then
            pbest_i = p_i;
            if Evaluation(p_i) is better than Evaluation(gbest) then
                gbest = p_i;
            end
        end
    end
until Stop conditions;
```

the state of the search, and modifying the inertia or the coefficients according to the state in which the search is..

4.3. Parameters selection and local optima avoidance

Both the DE and the PSO methods have some parameters with a high impact on the performance of the optimization process that need to be configured by the users. This way, PSO has the social and cognitive coefficients that intensify the search, as well as the inertia weight to diversify it. Regarding DE, it has the crossover constant, the mutation strategy and the mutation factor, which, depending on their values, will make the search more intensive or more diverse. Moreover, the size of the population (*pop_size*) is an important factor to take into account in both cases. The choice of values for the aforementioned configuration parameters is not trivial [54, 55], and it will be problem-dependent. In next section we propose an autotuning strategy to solve this issue.

Another potential problem of these methods is that they can get stuck in a local optimum during the global optimization. The selftuning of the parameters can help to leave these areas, but it may not be enough to obtain a solution of the desired quality. A restart mechanism is thus also proposed to avoid local optima.

5. Improving the optimization methods

The use of the *classic* versions of DE or PSO does not guarantee better results than the optimization methods already implemented in Gadget. However, thanks to the population-based scheme of these two algorithms, it is easy to apply a set of enhancements that improve their behavior for the optimization of the models developed using Gadget.

In this work we propose an enhancement template, which has been called PMA (Parallel Multi-restart Adaptive), that helps to obtain a good solution in the non-linear, multimodal and non-convex optimization problems generated by Gadget. It is based on the following new functionalities: (1) a parallel computation of the evaluations of the cost function in each iteration of the main loop of the methods; (2) a self-tuning of the configuration parameters to intensify the search when there is a promising result and obtain a good solution in a short time or, otherwise, to diversify it with a conservative parameter configuration when the algorithm is stuck in the proximity of a local optimum; and (3) an additional mechanism to restart the solutions of the algorithm, without losing the best solution found, to explore other regions when the search is stagnated.

5.1. Parallel algorithms

Regarding the first enhancement, the most time-consuming task in the two optimization methods considered is the evaluation of the cost function for each solution of the population (line 12 in Algorithm 1 and line 13 in Algorithm 2). The evaluation of each new solution does not depend on any other new solution generated. Thus, the loop in line 3 in Algorithm 1 and the loop in line 6 in Algorithm 2 can be parallelized. We have decided to implement the parallel versions using OpenMP, in which the execution of a parallel loop is based on the fork-join programming model. By using the directive shown in line 1 of Algorithm 3, a group of threads is created at the beginning of the parallel loops so that each thread evaluates concurrently a subset of the solutions, obtaining at the end a shared matrix with the results of the fitness values for each new solution.

There is a synchronization point at the end of the parallel loop: the different threads join again into a single thread and the computation does not

Algorithm 3: Parallel evaluation of the solutions using openMP

```
# pragma omp parallel for schedule(dynamic,1)
for ∀ solution p_i ∈ P do
    /* The generation of solutions was shown in Algorithms 1 and 2 */
    new_p_i = creation_new_solution(p_i)
    shared_fit_vector_i = function_evaluation(new_p_i)
end
```

continue until all of them have reached that point. Thus, if the workload has not been evenly distributed, many of the threads will remain idle waiting for the slowest one. Since there are variations in the computational load associated to the evaluation of different solutions, in order to avoid imbalances and reduce idle times, a dynamic schedule clause of OpenMP is included. Therefore, the computational load is distributed among the threads at runtime, sending more work to them as they complete their previously assigned evaluation.

5.2. Parameters self-tuning and multi-restart

The self-tuning of the configuration parameters and the multi-restart mechanism can be explained through the generation of a set of states, so that the optimization method changes of state and performs different actions depending on its behavior at a specific moment.

As the optimization progresses, three types of configuration parameters are dynamically adjusted in both methods (DE and PSO):

- Population size: It is a critical parameter in the performance of these methods. When a large population is chosen, the methods converge slowly due to the exploration of unpromising search space areas. Conversely, with a small population the methods can get stuck very easily in a local optimum. Thus, at runtime, our PMA template decreases the size of the population in order to intensify the search, or increases it to add diversity.

- Range of F (DE) or ω (PSO): The modification of the mutation factor or the inertia weight produces similar effects. When large values are chosen, the global exploration of the search space is favored. With small values, the exploitation is promoted, moving the particles in a specific area as in a local search method. PMA generates these parameters randomly within certain preset bounds, the interval defined by the bounds being smaller or larger depending on whether the method needs to intensify or diversify the search, respectively.

- Modification of CR (DE) or φ_p/φ_g (PSO): The crossover constant and the social/cognitive coefficients have a great impact in the behavior of the optimization. These parameters measure the influence of past solutions in a new one generated. Our PMA template adjusts these configuration parameters at runtime. When the optimization progresses properly these parameters remain fixed, since it is assumed that their configuration is good. But if the method is stagnated, they are changed by oscillating their values within a previously fixed interval. For example, and only when the search requires it, the parameter CR of the DE algorithm initially increases gently its value for each iteration of the algorithm until it reaches the upper bound (0.9 in this case). Then, the same process is repeated, but now decreasing CR until it reaches the lower bound (0.1 in this case), changing again the sense of modification, and repeating this cycle of increases and decreases. In the case of PSO, the process is the same, but the interval is $[0.1, 2.5]$, and when the method decreases the social coefficient, the cognitive one is increased, and vice versa. It deserves to be mentioned that these parameters ranges have been chosen based on the recommendations found in the literature [56, 57, 58, 59].

Once the three adaptive configuration parameters have been described, the flow of the algorithm and its states can be explained. They are shown in an abstract way in Figure 1 together with the definition of the main variables that control the state changes. A more detailed definition of the PMA template is found in the pseudocode in Algorithm 4. The PMA template considers five different states:

- State 0 is attained when the method does not achieve significant improvements in consecutive iterations (it_consec_impr < 2), the number of consecutive iterations in which the current best solution does not improve is lower than 10 (it_stuck < 10), and the number of iterations, not necessarily consecutive, with little improvements is lower than 10 (it_little_impr < 10). We define a significant improvement as one in which the best solution found enhances the fitness at least by 0.1% with respect to the previous best one, while a little improvement is one in which the enhancement is below 0.1%. State 0 has a balanced

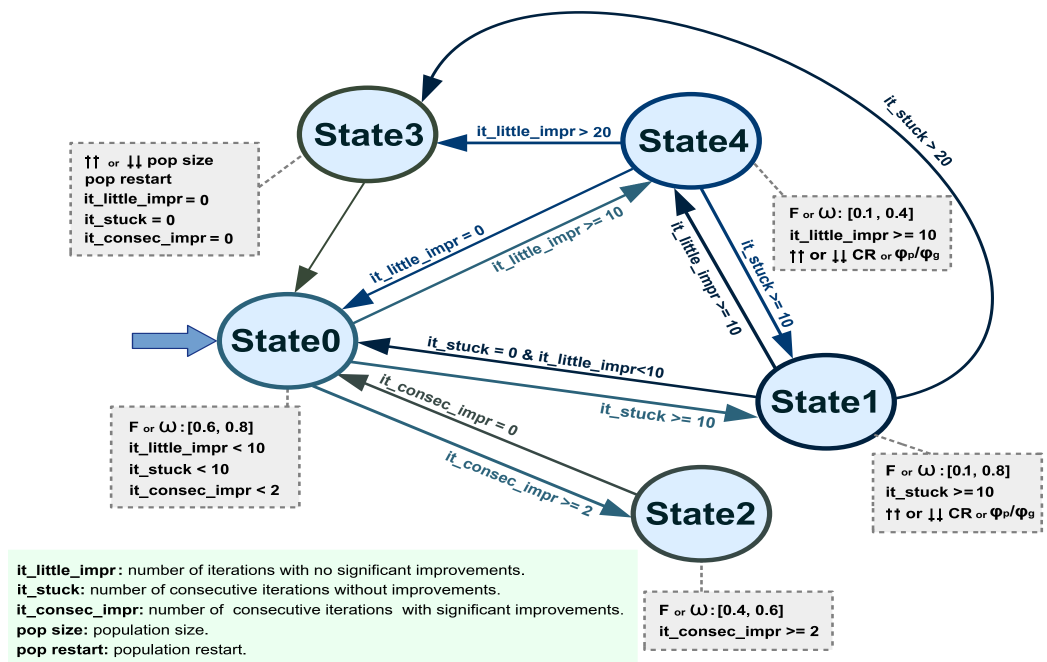


Figure 1: State diagram of the PMA template

behavior between intensification and diversification, producing values of F or ω according to this, and keeping the CR and φ coefficients constant because it is assumed that the configuration is good.

- State 1 is reached when the method is stuck, that is, when it has not been able to improve the best solution in at least the last 10 iterations (it_stuck ≥ 10). In this case, the method opens the bounds of F or ω and begins to swing the value of CR or the social/cognitive coefficients, with the goal of testing both intense and diverse configurations in order to look for a good solution, and stop being stuck. If a new best solution is obtained, the counter it_stuck is set to zero. If this solution does not improve significantly the previous best one, it_little_impr is also incremented. If this latter counter reaches the value 10, the flow of the algorithm is changed to state 4. Otherwise, the method moves to state 0 because it is assumed that the search is again in a good condition. Finally, if no improved solution is found in 10 additional iterations (it_stuck > 20), PMA moves to state 3.

- State 2 is obtained when the method has achieved to enhance the best known solution, by at least by 0.1% in consecutive iterations (it_consec_impr $\geq$ 2). We call that a significant improvement. Thus, it is in a promising area and the search is intensified decrementing smoothly the upper and lower bounds of F or ω and fixing the CR or φ_p/φ_g values. In the first iteration in this state in which no significant improvement is made, PMA returns to state 0.

- State 3 marks the beginning of the restart mechanism. Stagnation in the search can be caused by not obtaining significant improvements in the last 20 iterations (it_little_impr $>$ 20) or because the method has not obtained any improvement in the last 20 iterations (it_stuck $>$ 20), so it injects diversity in the population of the method modifying the population size and/or randomly restarting all their members. More details of this mechanism will be explained later. When the restart mechanism is fulfilled, the algorithm returns to the initial state 0 without losing the value of the best solution found.

- State 4 represents the situation in which the method improves repeatedly the best known solution, but with insignificant values (it_little_impr $\geq$ 10), that is, with improvement rates below 0.1%. The search in the current zone is strengthened, decreasing the bounds of F or ω and changing the values of CR or φ_p/φ_g. If the mentioned behavior persists, the method will move to state 3 and it will apply the multi-restart mechanism. If a significant solution is found, it will move to state 0 and it will continue to exploit this zone from this state. In the worst case, if it stops finding improved solutions during at least 10 consecutive iterations, the flow will move to state 1.

Using values of 10 or 20 iteration as thresholds to change state was motivated in the experience solving optimization problems produced by Gadget.

Algorithm 4 explains in detail the PMA template, where for each iteration in the main loop, the method in use (PSO or DE) follows these steps: (1) the value of ω or F is obtained from the current state; (2) the value of φ_p/φ_g or CR is also calculated, taking into account the flag par_trend_mod, which controls how these parameters must be modified; (3) if the current state is state 3, the Restart_Method procedure is called (Algorithm 5); (4) an iteration of the original method PSO or DE is carried on, obtaining a new best solution to compare with the current best known solution, updating the

Algorithm 4: Parallel Multirestart Adaptive (PMA) template

```
input  : P, a population of pop_size solutions with pop_size=nvars (number of variables)
output: xbest, the best solution
STATE, growth_trend_popul, par_trend_mod, type_restart=0;
it_consec_impr, it_stuck, it_little_impr=0;
PSO case: φ_p, φ_g=1.0 / DE case: CR=0.5;
repeat
    /* (1) CALC THE VALUE OF ω (PSO) OR F (DE)                                    */
    if STATE = 0 then  ω or F ← random number between [0.6 , 0.8] ;
    else if STATE = 1 then  ω or F ← random number between [0.1 , 0.8] ;
    else if STATE = 2 then  ω or F ← random number between [0.4 , 0.6] ;
    else if STATE = 4 then  ω or F ← random number between [0.1 , 0.4] ;
    /* (2) CALC THE VALUE OF φ_p/φ_g (PSO) OR CR (DE)                             */
    if (STATE == 1) or (STATE == 4) then
        if (par_trend_mod = 0)&(maximum φ_p/φ_g or CR is reached) then  par_trend_mod=1;
        if (par_trend_mod = 1)&(minimum φ_p/φ_g or CR is reached) then  par_trend_mod=0;
        if (par_trend_mod == 0) then
            DE case: CR=CR + 0.1 / PSO case: φ_p=φ_p + 0.1; φ_g=φ_g − 0.1;
        else
            DE case: CR=CR − 0.1 / PSO case: φ_p=φ_p − 0.1; φ_g=φ_g + 0.1;
        end
    end
    /* (3) RESIZE AND/OR RESTART POPULATION. Algorithm 5 for details.             */
    if (STATE = 3) then
        Restart_Method(P, it_consec_impr, it_little_impr, it_stuck, pop_size, nvars,
          growth_trend_popul);
    end
    /* (4) CALL OPTIMIZATION METHOD (PSO or DE). Algorithm 1 or 2 for details.    */
    new_xbest ← Optimization_Method(P);
    if (new_xbest improves xbest) then
        if (((xbest - new_xbest)/xbest)*100 >= 0.1) then
            it_little_impr, it_stuck=0;
            it_consec_impr=it_consec_impr+1;
        else
            it_consec_impr, it_stuck=0;
            it_little_impr=it_little_impr+1;
        end
        xbest =new_xbest;
    else it_stuck=it_stuck+1; it_consec_impr=0;
    /* (5) CALCULATING THE CURRENT STATE                                          */
    if (it_little_impr > 20) or (it_stuck > 20) then
        if (STATE = 1)  then  growth_trend_popul=0;
        else if (STATE = 4)  then  growth_trend_popul=1;
        STATE=3;
    else if (it_stuck ≥ 10) then   STATE=1 ;
    else if (it_little_impr ≥ 10) then   STATE=4 ;
    else if (it_consec_impr ≥ 2) then   STATE=2 ;
    else STATE=0;
until Stop conditions;
```

Algorithm 5: Restart method scheme.

```
Function Restart_Method(P, it_consec_impr, it_little_impr, it_stuck, pop_size, nvars,
  growth_trend_popul):
    it_consec_impr, it_little_impr, it_stuck = 0;
    addpop = nvars;
    if (growth_trend_popul = 0) then
        restart_flag = 0;
        if (pop_size − addpop) ≥ nvars then
            pop_size = pop_size − addpop;
        end
    else
        restart_flag = 1;
        if (pop_size + addpop) ≤ nvars ∗ 5 then
            pop_size = pop_size + addpop;
        end
    end
    Resize population P creating or deleting solutions, according to new pop_size;
    if (restart_flag = 1) then
        Random restart of all solutions of P;
    end
```

already explained counters it_stuck, it_consec_impr, and it_little_impr, and also updating the best solution if necessary; (5) and finally, with the data obtained in the previous steps, it is decided whether the algorithm must make a transition to another state.

Algorithm 5 shows the restart procedure. PMA invokes this procedure passing the population and its size, the number of problem variables $nvars$, the counters on which the state changes depend, and a new flag called growth_trend_popul. This flag is a binary variable that indicates whether the population must be increased or decreased. The amount to increment or decrement, represented by variable *addpop*, is equal to the number of problem variables, this value being also the minimum population size. The maximum population size is set to five times $nvars$. Thus, the PMA method does not allow to exceed these size bounds.

If the population size is reduced, the solutions are randomly restarted to introduce diversity. On the other hand, if the population is increased, the solutions remain unchanged and the diversity is provided by the new population members.

Figure 2 tries to explain graphically with a very simple example how the reset mechanism can help the method when it is stuck. In iteration 1, a population method has different solutions scattered across the search space (pink points), and one of them is in a promising region (red point). In successive iterations, the best solution is stuck in that promising region, attracting the rest of the solutions. Later, in iteration N, all the pink points are in the vicinity of the best solution, and the search will appear to be stuck in the local optimum. The reason is that the method does not have the capacity to generate solutions to escape from that region. Thus, the restart mechanism comes into play in iteration N+1, randomly restarting the solutions without losing the best known solution. In the following iterations, the movements of the pink points are influenced again by the attraction of the local minimum where the red point is located, but this time in the iteration N+2, one of the points follows a different path and reaches a more promising area in the search space, becoming the new best known value, and attracting the rest of the solutions to this new region.

We refer to PMA as a template because, although in this work it has been applied to the DE and the PSO methods, the proposed features can be easily extended to other population-based heuristics such as genetic algorithms or scatter search methods. Thus, throughout the present work, PMA DE or PMA PSO actually refers to the application of this template to the DE and PSO methods respectively.

Furthermore, when adaptive scheme proposed is applied to DE requires an additional comment. The mutation strategy used for our PMA DE is `rand-to-pbest/1/bin`, being almost exactly equal to strategy defined in Expresion 4.1, but with the difference that *pbest* is a randomly chosen solution between the p best solutions. Thus, depending to state, the size of pbest set can be bigger (exploration), or smaller (specialization).

Comparing the proposed adaptation strategy with that implemented in other methods as SaDE or jDE, where the algorithms have a learning time to try to known what settings are the more successful, our proposal changes their parameters basing on try to understand the current status of the search: e.g. trapped in local optima, moved fast convergence, or stucked in flatted surfaces.

6. Experimental results

In order to assess the efficiency of our proposal, different experiments have been performed to compare the PMA template applied to PSO and DE with the original versions of these heuristics as well as other optimization methods already implemented in Gadget. The rest of this section will use the acronyms introduced in Table 1 to simplify the explanations.

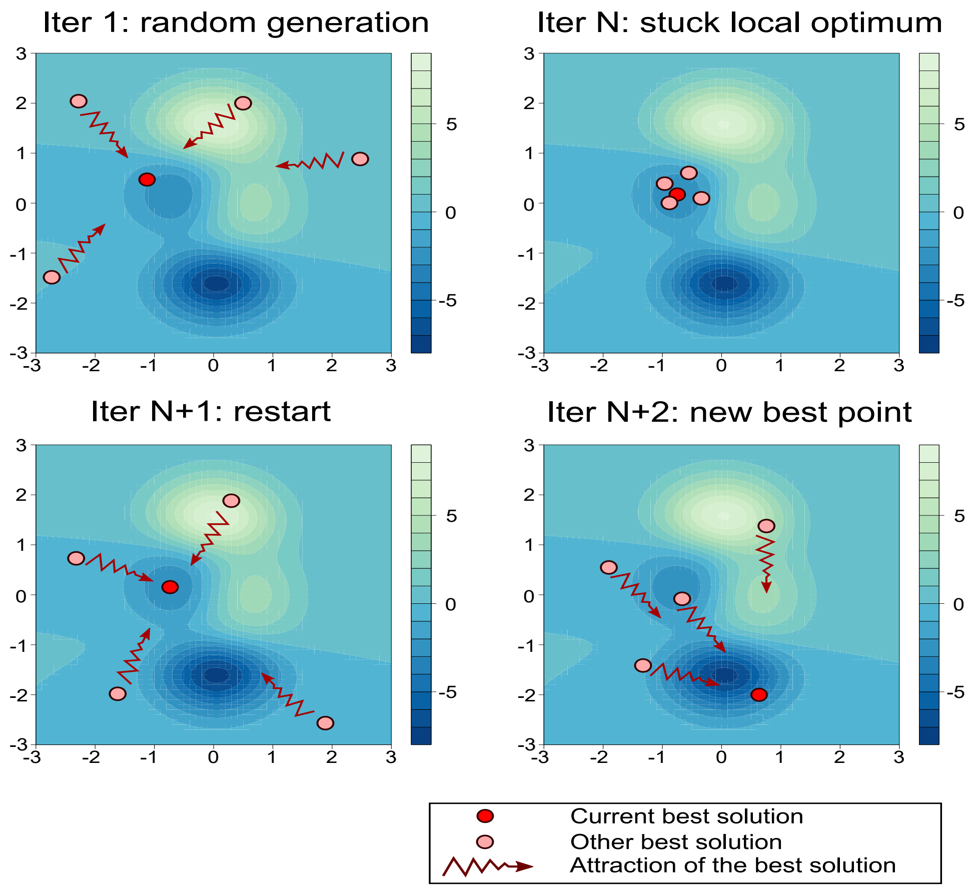


Figure 2: An example of the benefits of the multi-restart mechanism proposed. The example is minimizing the objective function.

The following three challenging optimization problems were used to check whether the enhancements proposed are adequate for the optimization of the models generated by Gadget:

- *Hake model* [60]: It was developed to assess the southern hake stock and give catch advice to EU through ICES (International Council for the Exploration of the Sea), examining complex fleet interactions and discards. It contains 62 continuous variables to be estimated.
- *Tusk model* [61]: It is an actual assessment model used to give tactical advices and it is based on over thirty years of data about a single-species of tusk (*brosme brosme*) in Icelandic waters. It was developed by the

Table 1: **Methods used in the experiments.**

Method	Description
PSO	Sequential Particle Swarm Optimization.
PMA PSO	Parallel Multi-restart Adaptive Particle Swarm Optimization.
APSO	Adaptive Particle Swarm Optimization [42].
CLPSO	Comprehensive Learning Particle Swarm Optimization [43].
DE	Sequential Differential Evolution.
PMA DE	Parallel Multi-restart Adaptive Differential Evolution.
SaDE	Self-adaptive Differential Evolution [41].
JDE	Self-adaptive Differential Evolution [39, 40].
SA	Sequential implementation of Simulated Annealing method.
SA specul	Speculative parallel version of Simulated Annealing proposed in [23].
HJ	Sequential implementation of Hooke and Jeeves method.
HJ specul	Speculative parallel version of Hooke and Jeeves proposed in [23].

MRI (Marine Research Institute, Iceland), and it is used by ICES as the basis for catch advice. It contains 47 continuous variables to be estimated.

- *Haddock model* [62]: It is a single-species, single-area model used to model the Icelandic haddock that is provided as an example with Gadget (it can be downloaded from the Gadget web [6]). It is a toy example used for illustrative purposes and its parameter space is fairly limited, containing 38 continuous variables to be estimated.

The experiments reported here have been performed in a multicore server whose main hardware and software features are described in Table 2. The optimization level `O3` was used in all the compilations. Moreover, due to the stochastic properties existing in the different heuristics used, each experiment reported in this section was performed 30 times in order to compare fairly the different optimization methods.

The experiments carried out are presented in three subsections. The first one is a comparative, using a single core, of the PMA template versus different configurations of the original implementations of the DE and PSO algorithms. The aim is to prove the reliability of the tuning and restart mechanisms provided with the PMA template. The second one evaluates the efficiency and scalability of the parallelization technique implemented in the PMA proposals using different number of cores. The third one compares the PMA PSO and DE algorithms with the SA and HJ methods, both sequentially and in parallel, under two complementary points of view [63]: horizontal and vertical view approaches.

Table 2: **Experimental environment.**

Feature	Value	Feature	Value
CPUs/Node	2 x Intel Xeon E5-2680 v3	**Memory/node**	64GB DDR4
#cores/CPU	12	**Compiler**	g++ 6.3.0
Total #cores	$2 \times 12 = 24$	**Compiler flags**	-O3
CPU Family	Haswell	**OS**	Red Hat Enterprise
CPU Frequency	2.5 GHz		Linux Server 6.7

Table 3: **List of configuration parameters used in PSO and DE implementations, selected according to [56, 57, 58, 59].**

method	φ_l	φ_g	popul. size	ω
PSO Configuration 1	2.0	2.0	nvars×3	0.7
PSO Configuration 2	2.0	2.0	nvars	0.7
PSO Configuration 3	0.75	0.75	nvars	0.7
PSO Configuration 4	2.0	2.0	nvars×3	adaptive[1]
APSO	adaptive	adaptive	nvars×3	adaptive
CLPSO	2.0	-	nvars×3	0.7
PMA PSO	adaptive	adaptive	adaptive	adaptive
method	**CR**	**F**	**popul. size**	**mutation Strategy**
DE Configuration 1	0.9	0.8	nvars×5	best/2/bin
DE Configuration 2	0.9	0.8	nvars×5	current-to-rand/1/bin
DE Configuration 3	0.9	0.8	nvars×5	current-to-best/1/bin
DE Configuration 4	0.9	0.8	nvars×5	rand-to-best/1/bin
JDE	adaptive	adaptive	nvars×5	rand/1/bin
SaDE	adaptive	adaptive	nvars×5	adaptive
PMA DE	adaptive	adaptive	adaptive	rand-to-pbest/1/bin

1 reducing ω along the nvars×3

6.1. *Comparison with other PSO and DE configurations*

There are many configurable parameters in DE and PSO whose selection may have a great impact in the speed of convergence. In order to evaluate the performance of the functionalities proposed by the PMA template, this has been compared sequentially, i.e. using a single core, with a set of versions of these methods configured with different typical values, which are shown in Table 3.

The results obtained appear in Table 4. Each experiment consisted of 30 independent runs, the stopping criterion being the execution time. The table shows for each optimization problem, the mean value and the standard deviation of the best cost function obtained by each optimization method executed during 1 hour. Let us remember that the lower the value, the better, because the goal is *minimize* the objective function. Moreover, the mean number of evaluations of the cost function performed is also reported.

According to the results obtained, both PMA PSO, and particularly PMA DE, achieve higher quality solutions than the rest of the configurations in the

Table 4: **Comparison of the proposals with other typical implementations of PSO and DE. Stopping criterion: 1 hour.**

method	Hake model mean fbest±std	Hake model mean evals	Tusk model mean fbest±std	Tusk model mean evals	Haddock model mean fbest±std	Haddock model mean evals
PMA PSO	1029.48±39.5	190364	6566.46±106.1	187249	0.88±0.00	293133
PSO Configuration 1	1376.95±122.1	181046	8978.73±516.5	180926	1.49±0.3	303156
PSO Configuration 2	1339.96±135.9	182474	9339.37±529.6	183661	1.75±0.6	322347
PSO Configuration 3	1527.93±99.3	179642	11195.56±601.1	179865	2.24±0.4	290082
PSO Configuration 4	1339.89±135.9	185952	9336.06±530.0	197323	2.26±0.5	292838
APSO	1124.80±66.5	197658	7631.23±346.0	190132	1.34±0.1	300127
LCPSO	1267.25±25.0	190131	8024.11±72.5	193422	1.17±0.0	299717
PMA DE	1004.05±9.2	180160	6540.71±65.3	191108	0.85±0.0	300568
DE Configuration 1	1461.42±40.2	180874	10529.41±171.5	181545	2.28±0.6	299459
DE Configuration 2	1445.37±33.2	180957	9470.96±156.3	181357	1.30±0.0	288971
DE Configuration 3	1453.17±35.3	171688	9516.29±157.7	172701	1.32±0.03	279515
DE Configuration 4	1128.02±82.7	178167	6984.08±218.6	185274	1.20±0.7	298623
JDE	1157.71±8.7	184160	7026.47±34.4	185540	0.86±0.0	290131
SaDE	1180.29±33.3	167286	7361.59±285.0	170141	0.88±0.0	280061

same threshold of time. Regarding the number of evaluations carried out, they were quite similar across all the cases for each model considered.

Figure 3 presents the same results, but emphasizing the distribution of the best fitness of the cost function obtained for each run and parameter configuration through violin plots. It can be observed that our proposal improves the quality of the achieved solution, being the median lower than for the other methods. Besides, the dispersion in the results is smaller, thus increasing the reliability of the method.

Due to the stochasticity existing in metaheuristics, it is good practice to validate the results using some type of statistical test [64]. Thus, the Wilcoxon Rank-sum test has been applied to check the statistical difference among the results set of our proposals versus the others the adaptive methods used.

These non-parametric test were carried on by pairs obtaining a p-value, which should be smaller than the significance level (α=0.05) to probe that the two results set are different. Table 5 shown this comparative, proving there are not a statistically similar results than our proposals PMA DE and PMA PSO. Other parameters in the table are R^+ and R^-, being the sum of the positive/negative ranks of the comparison of the sets. If R^+ is greater, the first method has better results than the second one, and if R^- is lower, it means the opposite case. Thus, the ranges in the table show: (1) our proposals are superior to the rest of the methods, (2) PMA DE has worked

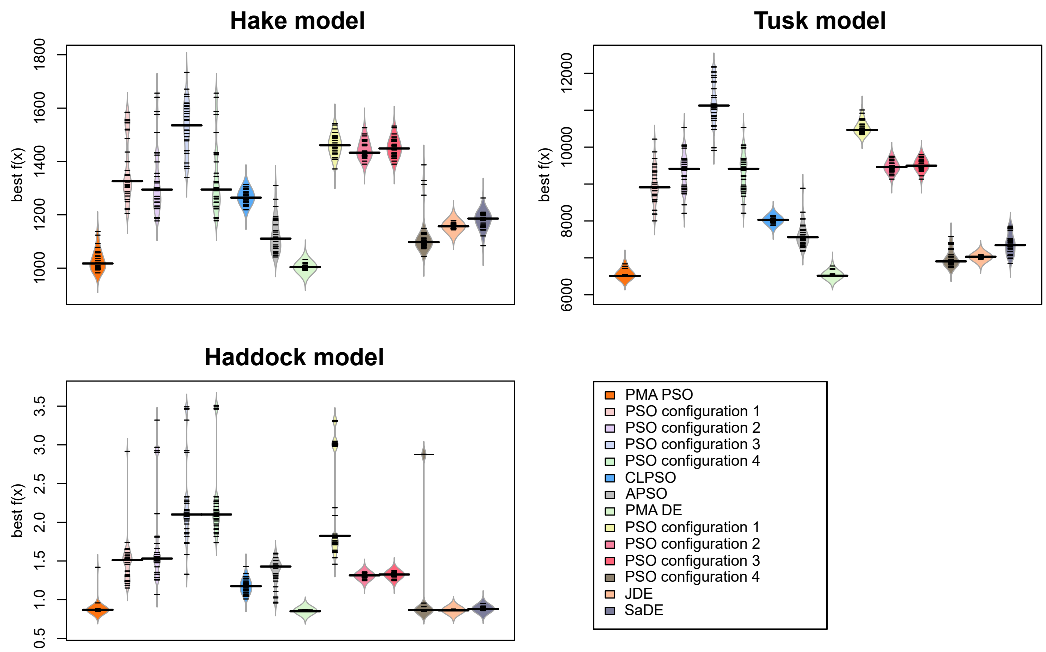


Figure 3: Violin plots corresponding with the results shown in Table 4.

better in Hake and Haddock models, and (3) PMA PSO has won in Tusk model.

Altogether, it can be concluded that our proposals have explored the search space better than the classic versions of DE and PSO, and also than other adaptive implementations, obtaining excellent results without spending time on tuning the configuration parameters.

6.2. Performance evaluation of the parallelization

The parallelization implemented in our proposal follows a shared memory scheme that relies on the OpenMP standard in which the work is distributed among the available processors (cores) when the search algorithm evaluates new generated solutions.

This subsection evaluates the scalability of the PMA template with the two search methods (DE and PSO) on the three Gadget models described. Since the PMA template introduces variations in the population size of the method through its adaptation mechanism, in order to make a fair study, 30 runs have been performed per test, using the mean value of the measured

Table 5: **Wilcoxon signed ranks test with a significance level $\alpha = 0.05$.**

Comparison	R^+	R^-	p-value	Comparison	R^+	R^-	p-value
Hake model							
Comparison	**R^+**	**R^-**	**p-value**	**Comparison**	**R^+**	**R^-**	**p-value**
PMA PSO vs APSO	450	15	1.25E-9	PMA DE vs APSO	465	0	1.69E-17
PMA PSO vs CLPSO	465	0	1.69E-17	PMA DE vs CLPSO	465	0	1.69E-17
PMA PSO vs jDE	465	0	1.69E-17	PMA DE vs jDE	465	0	1.69E-17
PMA PSO vs SaDE	465	0	3.21E-16	PMA DE vs SaDE	465	0	1.69E-17
PMA PSO vs PMA DE	97	368	0.017	PMA DE vs PMA PSO	368	97	0.017
Tusk model							
Comparison	**R^+**	**R^-**	**p-value**	**Comparison**	**R^+**	**R^-**	**p-value**
PMA PSO vs APSO	465	0	1.69E-17	PMA DE vs APSO	465	0	1.69E-17
PMA PSO vs LCPSO	465	0	1.69E-17	PMA DE vs LCPSO	465	0	1.69E-17
PMA PSO vs jDE	465	0	1.69E-17	PMA DE vs jDE	465	0	1.69E-17
PMA PSO vs SaDE	465	0	1.69E-17	PMA DE vs SaDE	465	0	1.69E-17
PMA PSO vs PMA DE	250	215	0.022	PMA DE vs PMA PSO	215	250	0.022
Haddock model							
Comparison	**R^+**	**R^-**	**p-value**	**Comparison**	**R^+**	**R^-**	**p-value**
PMA PSO vs APSO	464	1	1.20E-10	PMA DE vs APSO	465	0	2.99E-11
PMA PSO vs LCPSO	459	6	5.06E-10	PMA DE vs LCPSO	465	0	2.99E-11
PMA PSO vs jDE	145	320	0.025	PMA DE vs jDE	373	92	0.045
PMA PSO vs SaDE	366	99	1.67E-3	PMA DE vs SaDE	454	11	9.20E-9
PMA PSO vs PMA DE	22	443	4.42E-7	PMA DE vs PMA PSO	443	22	4.42E-07

execution time to compute the speedup. The stopping criterion was based on a maximum number of evaluations.

Figure 4 shows the speedup obtained when using from 1 to 24 threads, as this is the number of cores in the experimental environment. A good scalability is observed in the case of the Hake model, being a bit worse for the Tusk model, and finally limited for the Haddock model, which only evolves positively up to 8 threads. This behavior is strongly associated to the problem size: larger problems naturally scale better than smaller ones. Indeed, the larger the population, the size of the problem and its resolution complexity, the more noticeable the effect of the parallelization, since there will be more effort to be distributed among the processors in each iteration.

6.3. Comparison with other methods

The PMA template is compared now with other enhanced optimization methods implemented in Gadget, namely the speculative parallelization of Simulated Annealing and Hooke & Jeeves [23], and with a naive parallelisation of SaDE and APSO methods implemented for us. We develop a parallel version of these methods using openMP, following the same strategy than Algorithm 3, with the aim to make a fair comparison. It must be noticed that comparing different stochastic optimization methods is not a trivial task in

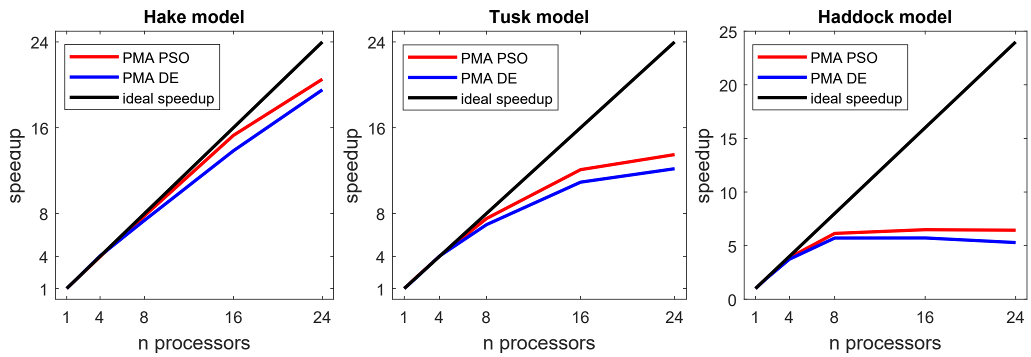


Figure 4: Speedup curves from PMA PSO and PMA DE. Stopping criteria: maximum number of evaluations reached = 10e5.

global optimization, particularly in real problems where the global solution is not known. For example, a solver can converge very fast to the vicinity of a good solution, but it can get stuck there for a long time. Another one may have a slower convergence at the beginning, but achieve more precise solutions without losing time in local optima. This way, deciding when the algorithm is stopped can determine whether one method is better than other. Therefore, in this subsection we use two complementary points of view [63] to compare the different methods, which helps better understand the improvements obtained in our proposal: (1) a vertical view approach, where all the experimental runs are carried on during a predefined time, using to compare them the quality of the solutions obtained, and (2) a horizontal view approach, where the experiments use a stopping criterion based in reaching a target value, the comparison between them being based on the time needed to achieve it.

Vertical approach

In this subsection the PMA template has been compared with the sequential and the speculative parallel implementation of SA and HJ algorithms using the vertical approach. Moreover, results of openMP parallelisation of SaDE and APSO were included. 30 runs were performed for each method, with a stopping criterion based on a predefined runtime equal to 1 hour. Hence, all the experiments had the same computational effort, and we can make a fair analysis of the quality of the solutions. Table 6 shows the obtained results. The mean and the standard deviation was reported for each experiment, and also the mean number of evaluations of the cost function

Table 6: **Analysis of solution quality in the proposed PMA PSO and PMA DE, with respect to the current methods used in Gadget, and popular versions of DE and PSO.** Stopping criteria: 1 hour of convergence time. SA settings: temperature = 1000, reduction factor = 0.85, step length = 1 and bound ratio = [0.3-0.7]. HJ settings: $\rho = 0.5$ and $\lambda = 0$.

		Hake model		**Tusk model**		**Haddock model**	
method	**num #thr**	**mean fbest±std**	**mean evals**	**mean fbest±std**	**mean evals**	**mean fbest±std**	**mean evals**
SA	1	6.66E13±3.6E14	182595	6518.27±6.0	186960	1.27±0.1	295505
SA specul.	2	1609.44±1604.2	358721	6519.40±13.1	367975	1.13±0.2	537013
	4	1170.68±429.0	495993	2.73E4±3.1E4	604012	1.14±0.1	683630
	8	1346.19±815.0	766663	3.54E4±3.3E4	980148	1.00±0.1	781493
	16	1116.23±253.3	1380884	3.77E4±3.8E4	1321137	1.12±0.2	711627
	24	1027.80±48.2	2098514	3.11E4±3.5E4	1308580	0.89±0.1	610720
HJ	1	1416.00±286.1	168992	1.73E4±1.9E4	176677	1.76±0.6	264419
HJ specul.	2	1413.97±287.1	326258	1.73E4±1.9E4	362748	1.75±0.6	505295
	4	1141.36±133.9	637125	1.56E4±1.6E4	687852	1.73±0.7	962520
	8	1102.55±92.3	1230323	1.42E4±1.6E4	1201591	1.80±0.9	1510076
	16	1070.85±100.1	2447248	1.47E4±1.6E4	1877228	1.82±0.9	1789527
	24	1061.54±101.2	3033093	1.48E4±1.6E4	2219249	1.56±0.8	1769972
PMA PSO	1	1029.48±39.5	190364	6581.89±111.9	187249	0.88±0.0	293133
	2	1016.12±31.5	357711	6542.49±83.8	396037	0.86±0.0	600122
	4	1007.34±29.5	713655	6533.78±78.5	778649	0.86±0.0	1153273
	8	996.60±23.2	1416943	6532.92±78.7	1477452	0.86±0.0	1941454
	16	988.71±20.2	2804966	6532.85±78.8	2546702	0.86±0.0	2016558
	24	984.81±19.0	4141767	6532.85±78.8	2759196	0.86±0.0	2012913
PMA DE	1	1004.05±9.2	180160	6540.71±65.3	191108	0.85±0.0	300568
	2	994.43±7.2	357697	6519.38±36.2	373525	0.85±0.0	576810
	4	986.16±6.4	719796	6507.47±0.8	735574	0.85±0.0	1154378
	8	979.84±4.9	1400523	6507.02±0.0	1787486	0.84±0.0	1991550
	16	974.69±3.7	2828591	6507.02±0.0	2488887	0.84±0.0	2085913
	24	972.06±14.0	4222821	6507.02±0.0	2787086	0.84±0.0	2129764
parallel APSO	1	1124.80±66.5	197658	7631.23±346.0	190132	1.34±0.1	300127
	2	1110.91±70.6	353988	7500.41±336.9	391014	1.34±0.1	585067
	4	1090.52±72.5	747078	7294.21±289.1	775245	1.33±0.1	1021268
	8	1073.77±71.2	1521684	7034.01±206.5	1467456	1.32±0.1	1874011
	16	1067.06±69.7	2584057	6906.29±218.0	2307418	1.32±0.1	2020037
	24	1065.17±68.0	4398336	6916.99±160.8	2801132	1.31±0.1	2230029
parallel SaDE	1	1180.29±33.3	167286	7361.59±285.0	170141	0.88±0.0	280061
	2	1045.43±23.8	348371	6615.73±15.7	366621	0.86±0.0	510081
	4	987.02±15.7	701042	6521.90±50.9	700566	0.85±0.0	1005161
	8	969.37±3.2	1201394	6515.63±47.1	1301221	0.84±0.0	1750125
	16	966.94±3.5	2644281	6507.02±0.0	2158616	0.84±0.0	2076131
	24	965.49±1.3	4035211	6507.02±0.00	2523166	0.84±0.0	2011131

performed during the execution time.

Analyzing the results, both in the sequential case and using different numbers of threads, PMA DE and PMA PSO obtain higher quality solutions than SA and HJ in the Hake and Haddock models. Moreover, our proposals always manage to improve the solution when more threads are added. This, however, does not always happen in the speculative parallelization versions of SA and HJ. In the Tusk model, SA has obtained very good results, sequentially and with its speculative version, using 2 threads, improving slightly the PMA implementations with the same resources. In contrast, the speculative parallelization in SA has a very poor performance when more threads are added. This type of parallelization tries to predict and compute in parallel the paths where the search could go, obtaining good results in Haddock and Hake models. However, it seems to have a very irregular behavior especially for the Tusk model, where a lot of runs get stuck in local optima. For more details of this parallel version of SA and HJ have been shown in [23].

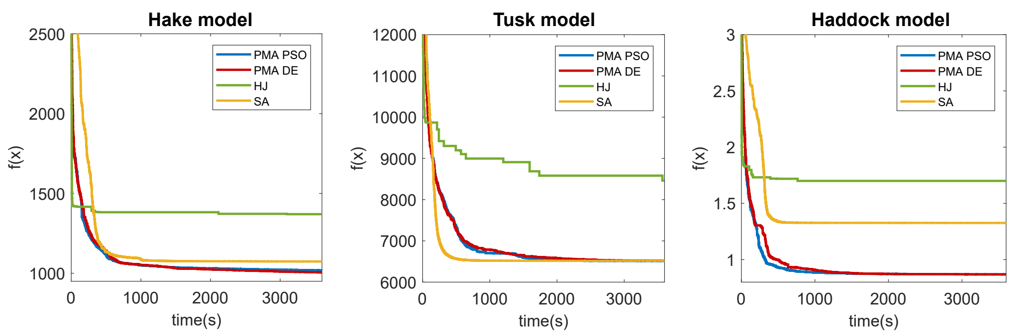
On the other hand, our proposals obtained a far superior results than the adaptive method APSO. PMA DE also reaches faster to the vicinity of the global optimum than SaDE in the cases of Tusk and Haddock models. In Hake model, in spite of PMA DE returned the best solution values with less computational resources, SaDE achieved a excelent result with more than 4 parallel processors. These results are due to the efficiency of the SaDE adaptation, which is based on learning memory, so that once many evaluations of the objective function have been carried out, the metaheuristic has been well tuned. However, a long numbers of iterations of the algorithm have had consumed to achieve this situation.

While Table 6 shows a summary of the final solution obtained in the different experiments, it is also interesting to analyze how the evolution of the search has been along the different runs carried out. Figures 5(a), 5(b), 5(c) and 5(d) show the convergence curves for the sequential and the parallel methods with 4, 8 or 24 threads, respectively. In both cases, each line of the graph represents the convergence curves for those experiments that fall in the median values of the results distribution for each instant of time. The results show that the PMA proposal has always a better convergence time, that is, a good quality solution is found in less time, except for the Tusk model using the sequential method, where the SA algorithm converges faster.

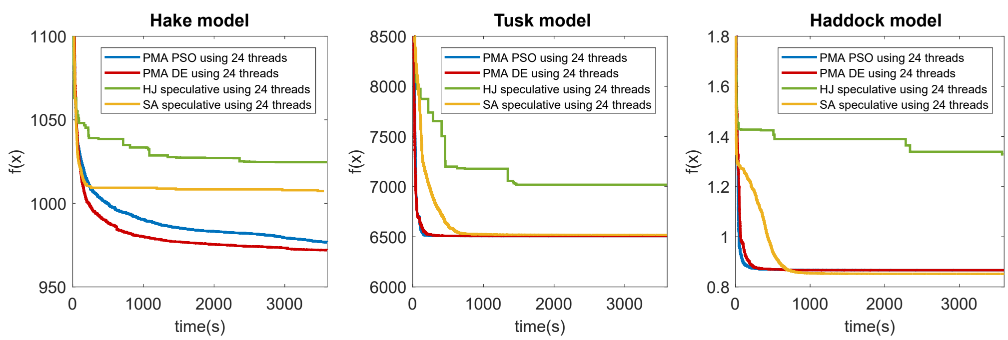
Horizontal approach

In this subsection an additional point of view, the horizontal approach, is included to complete the understanding of the benefits provided by our proposal. Thus, new experiments have been performed with a stopping criterion based on reaching a solution with an acceptable quality, so that the analysis focuses on the computational time needed to reach that point.

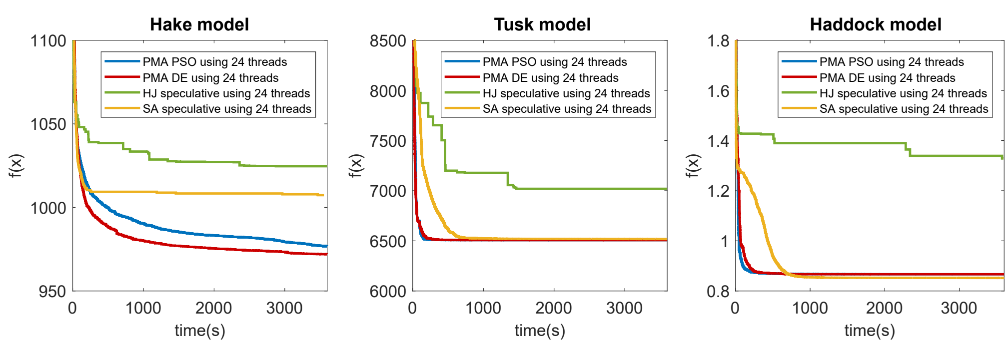
The selected target solutions, or VTR (value to reach), are related to the



(a) Convergence curves for the sequential methods



(b) Convergence curves for the methods using 8 threads



(c) Convergence curves for the methods using 24 threads

Figure 5: Convergence curves for those experiments that fall in the median values of the results distribution [ESTOUNAS XENERANDO]

best points obtained in the results using the vertical approach: 963.90 in the Hake model, 6507.02 in the Tusk model, and 0.849 in the Haddock model.

The aim in the experiments using the horizontal view is to reach values, valued by the developers of the models as with quality enough, that are at a distance of approximately 5% with respect to the best solutions previously obtained. Thus, the VTRs of these tests are the following: 1016.36 in the Hake model, 6832.37 in the Tusk model, and 0.891 in the Haddock model. Furthermore, since not all tests are able to reach these VTRs in a reasonable time, a new condition is added to the stop criterion: a maximum execution time equal to 3 hours.

Table 7 shows the mean and standard deviation of the runtime needed to reach the VTR for each model, using the different methods with a different number of threads. Besides, the column named "%hits" reports the percentage of the 30 runs that managed to reach the target value before the time threshold (3 hours).

For the Hake and Haddock models, it is observed that the PMA proposals need shorter times to reach acceptable solutions. In addition, the percentage of success of the different runs is higher, particularly when using the DE method, where all the runs achieve the VTR before the maximum time. Although in the Tusk model the SA algorithm has a better performance in the sequential version and with a reduced number of threads, it behaves badly when the number of threads grows, being overtaken by our proposals when the number of processors used is more than four. This is because, although PMA DE and PMA PSO have a slow behavior with few processors, they scale very well when more cores are added, thus reducing considerably the execution time in the parallel versions.

Figure 6 shows, using Violin plots, the distribution of the runtime of the different optimization methods for 24 threads. It can be observed that the proposed PMA methods obtain a better median than HJ and SA in all the models mainly because in these two latter methods many executions get stuck in local minima.

6.4. Application to other optimization problems

Finally, as PMA template can be applied to other types of bound-constrained optimization problems, we compared the performance of our proposals versus other adaptive version of DE and PSO as SaDE and APSO, using the popular benchmarks CEC 2013 [65], where there are implemented classical and hard to solve synthetic functions, presenting many of them multimodality as in the Gadget problems.

Table 7: **Dispersion of execution times to reach a minimum quality solution. Stopping criteria: (1) a predefined value to reach (VTR): hake=1016.364258, tusk=6832.373331) and haddock=0.891534, and (2) maximum time 3 hours.**

method	**num #thr**	**Hake model mean time(s)±std**	**Hake model %hits**	**Tusk model mean time(s)±std**	**Tusk model %hits**	**Haddock model mean time(s)±std**	**Haddock model %hits**
SA	1	9128±3684	20.0%	298±30	100.0%	10800±0	0.0%
SA specul.	2	7036±5048	36.6%	129±11	100.0%	8767±3914	20.0%
	4	5503±5264	56.6%	6513±5341	40.0%	8784±3908	20.0%
	8	5495±5339	53.3%	7885±4775	26.6%	5669±4895	56.6%
	16	5416±5335	53.3%	5198±5377	50.0%	8223±4444	30.0%
	24	4105±5106	66.6%	5068±5454	53.3%	1226±2609	93.3%
HJ	1	10444±1954	3.3%	9079±3603	20.0%	10800±0	0.0%
HJ specul.	2	10443±1959	3.3%	8942±3820	20.0%	10721±434	3.3%
	4	9940±2797	10.0%	7763±4757	30.0%	10800±0	0.0%
	8	8965±3726	23.3%	6466±5180	46.6%	10545±1244	10.0%
	16	7695±4634	33.3%	6381±5202	43.3%	10444±1952	3.3%
	24	7410±4763	36.6%	6195±5119	46.6%	10109±2635	6.6%
PMA PSO	1	5696±4362	63.3%	1027±1125	100%	1288±1090	100%
	2	4420±4268	76.7%	513±559	100%	680±587	100%
	4	3313±4034	83.3%	263±286	100%	356±307	100%
	8	2464±3722	86.7%	138±147	100%	211±177	100%
	16	1818±3346	90.0%	84±87	100%	190±162	100%
	24	1488±3006	93.3%	73±75	100%	189±164	100%
PMA DE	1	1264.10±755.8	100%	659.18±479.8	100%	602.95±242.4	100%
	2	714±610	100%	328±238	100%	299±120	100%
	4	354±300	100%	166±120	100%	154±61	100%
	8	194±164	100%	93±67	100%	92±36	100%
	16	102±86	100%	53±37	100%	74±29	100%
	24	71±58	100%	41±28	100%	74±29	100%
parallel APSO	1	9070±2924	26.6%	8931±2706	36.6%	10115±2089	10.0%
	2	6917±3278	60.0%	6925±3150	63.3%	9878±2812	10.0%
	4	6840±3225	63.3%	6620±3131	66.6%	9656±2993	13.3%
	8	6403±3410	63.3%	3779±124	100%	8776±3145	30.0%
	16	5845±3315	70%	3629±185	100%	8752±3182	30.0%
	24	5753±3237	73.3%	3622±306	100%	8286±3366	36.6%
parallel SaDE	1	7214.94±1669.5	93.3%	3675.32±871.0	100%	2102.79±391.4	100%
	2	3924.66±1732.3	100%	3804.20±1491.0	100%	1008.05±204.9	100%
	4	1801.60±574.6	100%	907.33±187.8	100%	546.990±136.3	100%
	8	1114.14±367.9	100%	546.75±144.1	100%	317.25±65.2	100%
	16	485.51±128.9	100%	293.91±73.2	100%	294.22±76.9	100%
	24	355.24±85.0	100%	213.01±46.8	100%	274.21±71.2	100%

The experiments carried out follow the typical procedure for `CEC2013` benchmarks: a number of different executions is performed for each function, using a stopping criterion based on reach to predefined number of evaluation of the objective function (being in this case equal to dimension of the problem $\times$ 1E4). Thus, once the experiments completed, a comparison can be made with the quality of results that has been reached in each method.

Regarding to the size of problems selected, we choose a dimension equal

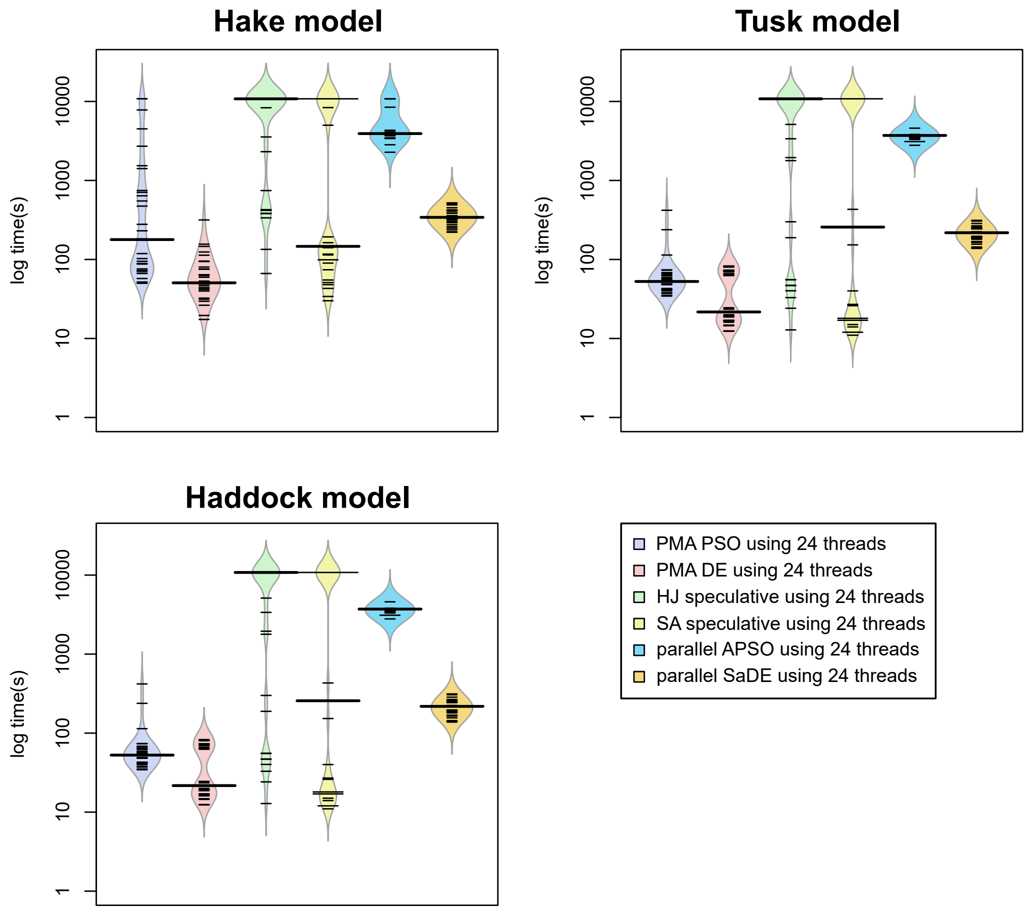


Figure 6: Violin plots corresponding with the execution time needed to obtain the target value for each method, using 24 threads.

to 50, because is the higher for this suite of benchmarks. It can be shown in previous sections, Gadget problems are complex problem to solve. For this reason, we are interested in evaluating the performance of our methods in the most difficult cases of CEC2013.

Table 8 shown the results for PMA DE and PMA PSO versus APSO and SaDE. For each function, the mean best quality solution obtained and their standard deviation is reported. The entries marked in bold show the method with the best performance for a specific benchmark. Therefore, the proposed

Table 8: **Best solution found in CEC2013 [65] benchmarks using dimension problem equal to 50 and a stopping criterion based on achieve to 500000 (dim×10000) evaluations of objective function.**

	PMA DE		PMA PSO		SaDE		APSO	
Fun.	fbest	±std	fbest	±std	fbest	±std	fbest	±std
f-1	4.7E-6	9.8E-6	**1.E-19**	**5E-19**	2.2E-2	6.5E-2	1.03	1.3
f-2	**7.3E+6**	**2.4E+6**	7.89E+7	2.3E+7	2.23E+7	7.5E+7	7.89E+7	2.3E+7
f-3	**4.96E+6**	**9.0E+6**	6.03E+8	7.8E+8	5.33E+8	1.7E+9	6.03E+8	7.8E+8
f-4	3.09E+4	5.0E+3	**2.72E+3**	**9.7E+2**	1.17E+5	1.4E+4	**2.72E+3**	**9.7E+2**
f-5	5.28E-6	1.6E-5	**4.18E-7**	**1.5E-6**	5.01E-2	1.21E-1	**4.18E-7**	**1.5E-6**
f-6	**45.6**	**1.7**	60.5	2.4E+1	46.6	1.0	1.28E+2	6.9E+1
f-7	**13.1**	**1.1E+1**	1.40E+02	2.8E+1	44.0	2.6E+1	2.92E+2	2.4E+2
f-8	2.11E+1	4.1E-2	2.11E+1	3.2E-2	**2.11E+1**	**3.0E-2**	21.1	3.1E-2
f-9	67.7	1.7	**55.6**	**6.8**	72.7	1.4	6.30E+1	3.8
f-10	1.28E-1	1.0E-1	**1.03E-1**	**5.5E-2**	1.62	2.2	2.15E+2	6.5E+1
f-11	1.07E+2	9.0	15.7	6.9	3.04E+2	2.5E+1	**3.53**	**1.4**
f-12	**3.08E+2**	**5.0E+1**	4.23E+2	1.2E+2	4.14E+2	2.6E+1	7.49E+2	2.4E+2
f-13	**3.23E+2**	**1.3E+1**	5.24E+2	9.7E+1	3.87E+2	1.8E+1	8.64E+2	1.5E+2
f-14	4.56E+3	2.9E+2	1.78E+3	4.7E+2	1.00E+4	6.2E+2	**7.86E+1**	**4.7**
f-15	1.36E+4	4.2E+2	**7.96E+3**	**1.0E+3**	1.42E+4	3.1E+2	8.90E+3	1.4E+3
f-16	3.41	2.2E-1	**1.80**	**5.0E-1**	3.38	2.2E-1	2.23	4.0E-1
f-17	1.74E+2	8.7	94.7	1.5E+1	3.75E+2	3.1E+1	**58.8**	**2.6**
f-18	**3.73E+2**	**1.6E+1**	4.62E+2	1.6E+2	4.40E+2	1.4E+1	1.20E+03	2.8E+2
f-19	18.0	2.5	**9.7**	**3.3**	30.6	2.2	13.8	5.3
f-20	**21.9**	**2.4E-1**	22.0	1.26	22.9	2.1E-1	23.0	1.11
f-21	**3.52E+2**	**3.0E+2**	9.04E+2	3.1E+02	9.49E+2	2.0	8.07E+2	3.95E+2
f-22	4.46E+3	4.4E+2	1.82E+3	5.5E+2	1.06E+4	6.5E+2	**1.32E+2**	**8.2E+1**
f-23	1.34E+4	3.9E+2	**9.15E+3**	1.5E+3	1.43E+4	3.8E+2	1.05E+4	1.58E+3
f-24	**2.34E+2**	**1.8E+1**	3.70E+2	2.1E+1	3.76E+2	1.7E+1	4.01E+2	1.3E+1
f-25	**2.92E+2**	**1.2E+1**	4.08E+2	1.6E+1	4.00E+2	1.8E+1	4.36E+2	1.7E+1
f-26	**2.66E+2**	**8.6E+1**	2.85E+2	1.1E+2	4.02E+2	1.0E+2	4.49E+2	4.7E+1
f-27	**7.98E+2**	**3.0E+2**	1.82E+3	1.4E+2	2.14E+3	3.7E+1	2.01E+3	1.2E+2
f-28	**7.08E+2**	**9.3E+2**	1.54E+3	1.8E+3	9.27E+2	1.2E+3	2.70E+3	2.6E+3

methods obtain the best results in most of the benchmarks, highlighting that in these experiments PMA PSO results are similar to PMA DE, something that did not happen in Gadget problems.

7. Conclusions and Future Work

The aim of this work was to propose enhanced optimization methods to improve the fit between the models generated by Gadget and the observed real data in a reasonable time.

Population-based metaheuristics such as DE and PSO were considered, and the PMA (Parallel Multi-restart Adaptive) template was proposed to improve the behavior of this kind of methods. The PMA template comprises three functionalities: a self-tuning mechanism that dynamically adjusts the configuration parameters of the population method; a parallel scheme that allows the parallel computation of the most compute intensive part of the optimization methods, which is the evaluation of the cost function; and a restart mechanism that avoids the stagnation of the algorithms on local minima.

The experimental results obtained using three different models generated by Gadget proved that the PMA DE and PSO methods: 1) enhance the behavior of the original DE and PSO algorithms thanks to the auto-tuning and restart mechanisms, reducing the dispersion of the results and improving the quality of the solutions; 2) provide a scalable solution when increasing the number of threads; and 3) obtain better solutions than the parallel methods already included in Gadget, the SA and the HJ algorithms. The PMA DE and PSO methods managed to reduce the execution times required to achieve a good solution, reducing the number of times the algorithm becomes stuck in a local minimum, and getting high quality solutions with less time-solution dispersion in the three models analyzed.

Therefore, the proposed PMA DE and PMA PSO methods have proven to be a good option to adjust the parameters of the Gadget models, not only because of their good performance, but also thanks to the absence of configuration parameters to be calibrated, which facilitates their use by non-experts users.

Although in this work the PMA template was only applied to the DE and the PSO methods, it can be easily adapted to other population-based heuristics such as genetic algorithms or scatter search methods.

Moreover, PMA template have been tested in a set of synthetic bound-constrained benchmark, obtaining a very promising results. However, multirestart mechanism implemented in our proposal is very aggressive, obtaining slow down the convergence in some optimization problems.

The self-adaptation proposed in this work is based on trying to know the situation of the global search, unlike others methods where the goal is learning which are the best configuration parameters through a memory during the runtime. Both points of view have their advantages and disadvantages: the learning phases can be slow, but they can achieve very good solutions, as in the case of the SaDE in the hake model or the good performance of APSO in some benchamrks of the `CEC2013`. On the other hand, in our proposal the adaptation is more aggressive and immediate, reaching to very good solutions in a short time in the most case studied.

As future work, the population-based methods could be combined with exact methods from mathematical programming with the purpose of creating matheuristics to further improve the quality of the solutions and reduce the execution times in these complex fisheries stock assessment models.

Other open issue can be expand the adaptive scheme proposed to addresses constrained optimization problem, trying to improve the handle of the infeasibilities from the knowledge of the state of the search proposed by our algorithm.

The new optimization methods developed in this work are publicly available under GPLv2 license at www.github.com/hafro/gadget (branch OpenMP).

Acknowledgments

This research was supported by the Galician Government (Xunta de Galicia) under the Consolidation Program of Competitive Research (refs. ED431C 2017/04 and R2016/045).

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no.613571.

We gratefully thank Galicia Supercomputing Center for providing access to the multicore server used for the experimental evaluation, and B. Elvarsson for assistance with the assessment of the implementation.

References

[1] J. J. Pella, P. K. Tomlinson, A generalized stock production model, Inter-American Tropical Tuna Commission Bulletin 13 (1969) 416–497.

[2] K. Patterson, R. Cook, C. Darby, S. Gavaris, L. Kell, P. Lewy, B. Mesnil, A. Punt, V. Restrepo, D. W. Skagen, et al., Estimating uncertainty in fish stock assessment and forecasting, Fish and fisheries 2 (2001) 125–157.

[3] G. Einarsson, Competitive coevolution in problem design and metaheuristical parameter tuning, Master's thesis, University of Iceland, Iceland, 2014.

[4] G. Einarsson, T. P. Runarsson, G. Stefansson, A competitive coevolution scheme inspired by de, in: Differential Evolution (SDE), 2014 IEEE Symposium on, IEEE, pp. 1–8.

[5] A. Punt, B. Elvarsson, Improving the performance of the algorithm for conditioning implementation simulation trials, with application to north atlantic fin whales, IWC Document SC/D11/NPM1 (7pp) (2011).

[6] J. Begley, Github Gadget, `http://www.github.com/hafro/gadget`, 2003.

[7] J. Begley, D. Howell, An overview of Gadget, the Globally applicable Area-Disaggregated General Ecosystem Toolbox, 2004.

[8] J. Begley, Gadget user guide, `http://www.hafro.is/gadget/files/userguide.pdf`, 2012.

[9] L. Taylor, J. Begley, V. Kupca, G. Stefansson, A simple implementation of the statistical modelling framework Gadget for cod in Icelandic waters, African Journal of Marine Science 29 (2007) 223–245.

[10] H. Björnsson, T. Sigurdsson, Assessment of golden redfish (sebastes marinus L) in Icelandic waters, Scientia Marina 67 (2003) 301–314.

[11] B. Elvarsson, L. Taylor, V. Trenkel, V. Kupca, G. Stefansson, A bootstrap method for estimating bias and variance in statistical fisheries modelling frameworks using highly disparate datasets, African Journal of Marine Science 36 (2014) 99–110.

[12] E. Andonegi, J. A. Fernandes, I. Quincoces, X. Irigoien, A. Uriarte, A. Pérez, D. Howell, G. Stefánsson, The potential use of a Gadget model to predict stock responses to climate change in combination with bayesian networks: the case of Bay of Biscay anchovy, ICES Journal of Marine Science 68 (2011) 1257–1269.

[13] S. McCully, F. Scott, L. Kell, J. Ellis, D. Howell, A novel application of the Gadget operating model to North East Atlantic porbeagle, Collect. Vol. Sci. Pap. ICCAT 65 (2010) 2069–2076.

[14] D. Howell, B. Bogstad, A combined Gadget/FLR model for management strategy evaluations of the Barents Sea fisheries, ICES Journal of Marine Science 67 (2010) 1998–2004.

[15] Report of the working group on the assessment of southern shelf stocks of hake, monk and megrim, Technical Report, CES CM 2010/ACOM, 2010.

[16] Report of the benchmark workshop on deep-sea stocks (wkdeep), Technical Report, ICES CM 2014/ACOM, 2014.

[17] Report of the benchmark workshop on redfish management plan evaluation (wkredmp), Technical Report, ICES CM 2014/ACOM, 2014.

[18] Report of the inter benchmark process on Greenland halibut in ICES areas I and Ii (ibphali), Technical Report, ICES CM 2015/ACOM, 2015.

[19] A. Corana, M. Marchesi, C. Martini, S. Ridella, Minimizing multimodal functions of continuous variables with the “simulated annealing” algorithm, ACM Transactions on Mathematical Software (TOMS) 13 (1987) 262–280.

[20] W. L. Goffe, G. D. Ferrier, J. Rogers, Global optimization of statistical functions with simulated annealing, Journal of Econometrics 60 (1994) 65–99.

[21] R. Hooke, T. A. Jeeves, “Direct search” solution of numerical and statistical problems, Journal of the ACM 8 (1961) 212–229.

[22] D. P. Bertsekas, Nonlinear programming, Athena scientific, Belmont, 1999.

[23] S. Vázquez, M. J. Martín, B. B. Fraguela, A. Gómez, A. Rodríguez, B. Elvarsson, Novel parallelization of simulated annealing and Hooke & Jeeves search algorithms for multicore systems with application to complex fisheries stock assessment models, Journal of Computational Science 17 (2016) 599 – 608.

[24] B. Chapman, G. Jost, R. van der Pas, Using OpenMP: portable shared memory parallel programming, Mit University Press Group Ltd, 2007.

[25] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, Journal of global optimization 11 (1997) 341–359.

[26] J. Kennedy, R. Eberhart, PSO optimization, in: Proceedings of the 1995 IEEE International Conference Neural Networks, volume 4, IEEE Service Center, Piscataway, NJ, pp. 1941–1948.

[27] R. J. Matear, Parameter optimization and analysis of ecosystem models using simulated annealing: a case study at Station P, Journal of Marine Research 53 (1995) 571–607.

[28] J. Hemmings, P. Challenor, Addressing the impact of environmental uncertainty in plankton model calibration with a dedicated software system: the marine model optimization testbed (marmot 1.1 alpha), Geoscientific Model Development 5 (2012) 471–498.

[29] R. D. Methot, C. R. Wetzel, Stock synthesis: A biological and statistical framework for fish stock assessment and fishery management, Fisheries Research 142 (2013) 86 – 99.

[30] D. A. Fournier, H. J. Skaug, J. Ancheta, J. Ianelli, A. Magnusson, M. N. Maunder, A. Nielsen, J. Sibert, Ad model builder: using automatic differentiation for statistical inference of highly parameterized complex nonlinear models, Optimization Methods and Software 27 (2012) 233–249.

[31] K. Tashkova, J. Šilc, N. Atanasova, S. Džeroski, Parameter estimation in a nonlinear dynamic model of an aquatic ecosystem with meta-heuristic optimization, Ecological Modelling 226 (2012) 36–61.

[32] J. Zhang, A. C. Sanderson, Jade: adaptive differential evolution with optional external archive, IEEE Transactions on evolutionary computation 13 (2009) 945–958.

[33] S. M. Islam, S. Das, S. Ghosh, S. Roy, P. N. Suganthan, An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 42 (2012) 482–500.

[34] J. Brest, B. Bošković, S. Greiner, V. Žumer, M. S. Maučec, Performance comparison of self-adaptive and adaptive differential evolution algorithms, Soft Computing 11 (2007) 617–629.

[35] X. Hu, R. C. Eberhart, Adaptive particle swarm optimization: detection and response to dynamic systems, in: Proceedings of the 2002 Congress on Evolutionary Computation (CEC), volume 2, IEEE, pp. 1666–1670.

[36] Y. Shi, R. C. Eberhart, Fuzzy adaptive particle swarm optimization, in: Proceedings of the 2001 Congress on Evolutionary Computation (CEC), volume 1, IEEE, pp. 101–106.

[37] Y. Wang, J. Zhou, C. Zhou, Y. Wang, H. Qin, Y. Lu, An improved self-adaptive PSO technique for short-term hydrothermal scheduling, Expert Systems with Applications 39 (2012) 2288–2295.

[38] R. Tanabe, A. Fukunaga, Success-history based parameter adaptation for differential evolution, in: Evolutionary Computation (CEC), 2013 IEEE Congress on, IEEE, pp. 71–78.

[39] J. Brest, S. Greiner, B. Boskovic, M. Mernik, V. Zumer, Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems, IEEE transactions on evolutionary computation 10 (2006) 646–657.

[40] J. Brest, V. Zumer, M. S. Maucec, Self-adaptive differential evolution algorithm in constrained real-parameter optimization, in: Evolutionary Computation, 2006. CEC 2006. IEEE Congress on, IEEE, pp. 215–222.

[41] A. K. Qin, V. L. Huang, P. N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, IEEE transactions on Evolutionary Computation 13 (2009) 398–417.

[42] Z.-H. Zhan, J. Zhang, Y. Li, H. S.-H. Chung, Adaptive particle swarm optimization, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 39 (2009) 1362–1381.

[43] J. J. Liang, A. K. Qin, P. N. Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE transactions on evolutionary computation 10 (2006) 281–295.

[44] D. Izzo, M. Rucinski, C. Ampatzis, Parallel global optimisation metaheuristics using an asynchronous island-model, in: Proceedings of the 2009 IEEE Congress on Evolutionary Computation (CEC), IEEE, pp. 2301–2308.

[45] J. Apolloni, J. García-Nieto, E. Alba, G. Leguizamón, Empirical evaluation of distributed differential evolution on standard benchmarks, Applied Mathematics and Computation 236 (2014) 351–366.

[46] M. Weber, F. Neri, V. Tirronen, Distributed differential evolution with explorative–exploitative population families, Genetic Programming and Evolvable Machines 10 (2009) 343.

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

[48] Y. Li, Y. Cao, Z. Liu, Y. Liu, Q. Jiang, Dynamic optimal reactive power dispatch based on parallel particle swarm optimization algorithm, Computers & Mathematics with Applications 57 (2009) 1835–1842.

[49] B.-I. Koh, A. D. George, R. T. Haftka, B. J. Fregly, Parallel asynchronous particle swarm optimization, International journal for numerical methods in engineering 67 (2006) 578–595.

[50] J. F. Schutte, J. A. Reinbolt, B. J. Fregly, R. T. Haftka, A. D. George, Parallel global optimization with the particle swarm algorithm, International journal for numerical methods in engineering 61 (2004) 2296–2315.

[51] R. Agrawal, A. Goyal, D. Sambasivam, A. K. Bhattacharya, Parallelization of industrial process control program based on the technique of differential evolution using multi-threading, in: Proceedings of the 2014 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), IEEE, pp. 546–550.

[52] Z.-H. Liu, X.-H. Li, W. Tan, Z. Zhang, OpenMP-based multi-core parallel cooperative PSO with ICS using machine learning for global optimization problem, in: Proceedings of the 2015 IEEE International Conference on Systems, Man, and Cybernetics (SMC), IEEE, pp. 2786–2791.

[53] D. Wang, C.-H. Wu, A. Ip, D. Wang, Y. Yan, Parallel multi-population particle swarm optimization algorithm for the uncapacitated facility location problem using OpenMP, in: Proceedings of the 2008 IEEE Congress on Evolutionary Computation (CEC), IEEE, pp. 1214–1218.

[54] R. Gämperle, S. D. Müller, P. Koumoutsakos, A parameter study for differential evolution, Advances in intelligent systems, fuzzy systems, evolutionary computation 10 (2002) 293–298.

[55] Y. Shi, R. C. Eberhart, Parameter selection in particle swarm optimization, in: Proceedings of the 1998 International Conference on Evolutionary Programming, Springer, pp. 591–600.

[56] J. Montgomery, S. Chen, An analysis of the operation of differential evolution at high and low crossover rates, in: Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8.

[57] S. Das, P. N. Suganthan, Differential evolution: A survey of the state-of-the-art, IEEE transactions on evolutionary computation 15 (2011) 4–31.

[58] A. Ratnaweera, S. K. Halgamuge, H. C. Watson, Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients, IEEE Transactions on Evolutionary Computation 8 (2004) 240–255.

[59] B. Mohammadi-Ivatloo, M. Moradi-Dalvand, A. Rabiee, Combined heat and power economic dispatch problem solution using particle swarm optimization with time varying acceleration coefficients, Electric Power Systems Research 95 (2013) 9–18.

[60] Report of the Working Group for the Bay of Biscay and the Iberian waters Ecoregion, Technical Report ICES CM/ACOM:11, International Council for the Exploration of the Sea, 2015.

[61] Report of the Benchmark Workshop on Deep-sea Stocks (WKDEEP), Technical Report ICES CM 2014/ACOM:44, International Council for the Exploration of the Sea, 2014.

[62] J. Begley, A brief guide to a simple Gadget example, `http://www.hafro.is/gadget/hadexample/hadexample.html`, 2003.

[63] N. Hansen, A. Auger, S. Finck, R. Ros, Real-Parameter Black-Box Optimization Benchmarking 2009: Experimental Setup, Technical Report RR-6828, INRIA, 2009.

[64] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm and Evolutionary Computation 1 (2011) 3–18.

[65] J. Liang, B. Qu, P. Suganthan, A. G. Hernández-Díaz, Problem definitions and evaluation criteria for the cec 2013 special session on real-parameter optimization, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, Technical Report 201212 (2013) 3–18.