The Journal of Supercomputing manuscript No.
(will be inserted by the editor)

Multi-level parallel chaotic Jaya optimization algorithms for solving constrained engineering design problems

H. Migallón · A. Jimeno-Morenilla · H. Rico · J.L. Sánchez-Romero · A. Belazi

Received: date / Accepted: date

Abstract Several heuristic optimization algorithms have been applied to solve engineering problems. Most of these algorithms are based on populations that evolve according to different rules and parameters to reach the optimal value of a function cost through an iterative process. Different parallel strategies have been proposed to accelerate these algorithms. In this work, we combined coarse-grained strategies, based on multi-populations, with fine-grained strategies, based on a diffusion grid, to efficiently use a large number of processes, thus drastically decreasing the computing time. The Chaotic Jaya optimization algorithm has been considered in this work due to its good optimization and computational behaviors in solving both the constrained optimization engineering problems (seven problems) and the unconstrained benchmark functions (a set of 18 functions). The experimental results show that the proposed parallel algorithms outperform the state-of-the-art algorithms in terms of

This research was supported by the Spanish Ministry of Science, Innovation and Universities and the Research State Agency under Grant RTI2018-098156-B-C54 co-financed by FEDER funds, and by the Spanish Ministry of Economy and Competitiveness under Grant TIN2017-89266-R, co-financed by FEDER funds.

H. Migallón
Department of Computer Engineering. Miguel Hernández University, E-03202 Elche, Spain.
Tel.: +34-966658390
Fax: +34-966658814
E-mail: hmigallon@umh.es

A. Jimeno-Morenilla
Department of Computer Technology, University of Alicante, E-03071, Alicante, Spain.

H. Rico
Department of Computer Technology, University of Alicante, E-03071, Alicante, Spain.

J.L. Sánchez-Romero
Department of Computer Technology, University of Alicante, E-03071, Alicante, Spain.

A. Belazi
Laboratory RISC-ENIT (LR-16-ES07), Tunis El Manar University, Tunis 1002, Tunisia.

optimization behavior, according to the quality of the obtained solutions, and efficiently exploit shared memory parallel platforms.

Keywords optimization · constrained engineering problem · Jaya algorithm · chaotic map · parallel algorithms · OpenMP

1 Introduction

In many fields of science and engineering, and many procedures in each area, the optimization of a function may be required. These functions are called objective or cost functions. The particular characteristics of such a function depend on the specific process it models. Depending on these characteristics, the optimization process can be a complex one, which may lead, for example, to high computational costs, or may even mean that it cannot be optimized using a deterministic mathematical procedure.

When an optimization problem can be solved using a deterministic method, it ensures that the optimization process generates a sequence of points that tend to the optimal value. That is, the optimization problem is solved as a linear algebra problem, in which the gradient of the objective function is usually used. Although these methods make it possible to solve a large number of optimization problems, even for large-scale problems, they are not always effective, either because they require excessive computing time to obtain the solution, or because the result obtained in the available time is not of the required quality.

When deterministic methods do not meet the requirements, or there is no deterministic method to solve an optimization problem, they can be replaced by meta-heuristic optimization methods, whereby the solutions obtained are acceptable, and the computational cost is reasonable. These methods aim to reach the optimum value of the function by meeting the computing time requirement. If the solution obtained is not the optimal value of the function, the solution obtained in the available computing time must be acceptable.

A wide range of meta-heuristic optimization algorithms is based on populations. These populations are modified iteratively to obtain new generations, whose objective is that they evolve towards the optimum of the function. Both the speed of approaching the optimum and how close to that optimal value it can get, depends on the process of the generation of new populations, i.e. on the heuristic optimization algorithm used.

Heuristic optimization methods have been proposed in which rules govern the generation of new populations based on physical processes, natural phenomena or mathematical functions, among others. Although it cannot be formally demonstrated that the optimum values thus obtained are the real solution to the problem, they have been experimentally proven to be robust.

Some of the proposed algorithms that have proven to be effective in several areas of science and engineering are: genetic algorithms (GA) [26] which reflect the process of natural selection; differential evolution (DE) [49] which attempts to iteratively improve a candidate solution for a given measure of quality; the evolutionary strategy (ES) algorithm [59] which was based on the processes of mutation and selection seen in evolution; genetic programming (GP) [31] and evolutionary programming (EP) [7]

which were based on the choice of individuals for reproduction (crossover) and mutation; the biogeography-based optimisation (BBO) algorithm [35] which improves solutions stochastically and iteratively; the gravitational search algorithm (GSA) [54] based on Newtons law of gravity; the grenade explosion method (GEM) algorithm [1] based on the characteristics of the explosion of a grenade; the ant colony optimization (ACO) algorithm [16] which imitates the foraging behaviour of ant colonies; the particle swarm optimization (PSO) algorithm [48] based on the social behavior of fish schooling or bird flocking; the artificial bee colony (ABC) algorithm [30] inspired by the foraging behavior of honey bees; the firefly (FF) algorithm [64] inspired by the flashing behavior of fireflies; the shuffled frog leaping (SFL) algorithm [17] which imitates the collaborative behavior of frogs; the teaching-learning-based optimization (TLBO) algorithm [52] based on the processes of teaching and learning; the Jaya algorithm [51] based on geometric distances and random processes; and the SCA algorithm [41] based on the sine and cosine trigonometric functions; among others.

The success of these algorithms depends on the ability to avoid being trapped in local minima, the speed of approaching the optimum, and how close to the optimum they can get. Depending on the function to be optimized, the optimization behavior of these methods varies. Besides, many of these methods require the correct setting of the control parameters of the optimizing method to ensure appropriate behavior, and this setting is not general but dependent on the function to be optimized. The non-optimal setting of these parameters can either cause a weak quality solution or increase the computational cost drastically, as more generations are required to improve the quality of the solution. Some algorithms such as TLBO, Jaya or SCA are free of control parameters.

These methods have shown their effectiveness in real-world applications. For example, in [46], GA was used in the path planning strategy for mobile robot navigation in static and dynamic environments. In [63] DE was used to develop a parallel machine scheduling that minimizes both the makespan and total energy consumption. A strategy to perform batch-mode active learning on multi-label data through an evolutionary approach, formulated as a multi-objective problem, was presented in [55]. In [60], a feature selection strategy based on GP was proposed and worked well with both balanced and unbalanced data. A learning mechanism for radial basis function networks based on BBO was presented in [5]. In [43] GSA is used for exergy optimization of the Bushehr nuclear power plant. A data clustering technique to divide a dataset into a few unsupervised data analysis partitions based on GEM was presented in [22]. The ACO was used in [42] to solve an advanced version of the vehicle routing problem called the fleet management system. In [28] PSO was used to improve the quality of the video for giving effective results in the video analysis process. The quadratic assignment problem was solved in [15] using ABC. In [32] a constrained multi-objective function is defined for privacy-preserving in social networks, and this function was solved by combining fuzzy clustering and FF. In [9] SFL is proposed to minimize the makespan when solving the distributed hybrid flow shop scheduling problem. In [45] both TLBO and Jaya were used to obtain the minimum cost of a reinforced concrete counterfort retaining wall. And, in [13] SCA is used to solve a short-term hydrothermal scheduling problem, among others.

All these algorithms include randomness in the generation of new populations, so chaos theory can be used to improve these algorithms. The chaos systems can replace the pseudo-random number generators (PRNGs) in producing the required random number sequences, or to perform a local search, or to generate the control parameters of the optimizing method [56]. Some meta-heuristic optimization algorithms have been successfully improved using chaos, for example in [20,23], chaos was applied to the FF algorithm; in [36,65] it was applied to GA algorithms; in [27,40], it was applied to the SA algorithm; in [58,62], it was applied to the BBO algorithm; in [29, 47], it was applied to the DE algorithm; in [2], it was applied to ABC algorithm; and in [21], it was applied to the GSA algorithm.

This paper's main contribution is to develop a suitable algorithm that improves the quality of the obtained solution when solving engineering design problems, fulfilling on the one hand that it is free of configuration parameters and on the other hand that it is algorithmically as simple as possible. These two assumptions may lead to an increase in the number of iterations required to achieve these objectives. Since this increase of iterations could cause an increase in the required computational cost, we designed improved multi-level parallel algorithms, combining coarse and fine grain strategies, based on the chaotic C-Jaya algorithm [18] and a diffusion grid (see, for example, [3,4]). On the one hand, multi-level parallel algorithms increase the parallel scalability of the proposed parallel algorithms and the number of concurrent processes used efficiently. To meet this objective, the parallel behavior of the multi-level parallel algorithm is optimized by working with process teams at the inner level instead of using nested parallelism. On the other hand, using both the chaotic map to replace randomness and a diffusion grid is intended to boost the optimization's performance, attending to the solution's quality in solving real-world engineering design problems.

The remainder of this paper is organized as follows. Section 2 presents a brief description of the Jaya algorithm and the 2D chaotic map used here. In Section 3, the multi-level parallel algorithms are explained in detail. In Section 4, we analyze the performance of the proposed parallel algorithms in terms of both optimization and parallel behavior, and finally, Section 5 concludes the paper.

2 Preliminaries

In this section, we briefly describe the Jaya algorithm, including its modification when using the 2D chaotic map, and also present the 2D chaotic map.

2.1 Jaya algorithm

The Jaya algorithm was first introduced and analyzed in the optimization of constrained and unconstrained functions in [51]. The outstanding behavior of this algorithm was shown in the first comparative studies presented in [51] and [53], and they have been confirmed in many subsequent papers. It is worth noting that this algorithm does not require the pre-setting of any specific parameters. Since the Jaya algorithm

is a population-based optimization algorithm, both the population size and the stop criterion are to be set.

The global optimum search strategy deployed in the Jaya algorithm is based on moving towards the current population optimum while avoiding the worst individual in the population. For the population evolving, new individuals are created following Eq. (1). In that equation, x is the individual to be replaced if the new individual (x^{new}) improves x, x_{best} and x_{worst} are the best and worst individual in the population, and $rand_1$ and $rand_2$ are two uniformly distributed random numbers in the range $[0, 1]$. The cost function to be optimized defines the number of variables for each individual in the population, and for each variable, two new random numbers ($rand_1$ and $rand_2$) must be obtained.

$$x^{new} = x + rand_1 \left(x_{best} - |x|\right) - rand_2 \left(x_{worst} - |x|\right) \tag{1}$$

In the iterative procedure of the Jaya algorithm, shown in Algorithm 1, Eq. (1) is applied to all individuals in the population. Note that the dimension of $MinValue$, $MaxValue, rand, rand_1, rand_2$ and individuals x^i, x_{best}, x_{worst} and x^{new} depends on the number of variables of the function to be optimized.

Algorithm 1 Jaya algorithm

Set population size
Set stopping criterion
for $i = 1$ to $PopulationSize$ {Create Initial Population X:} **do**
    $x^i = MinValue + (MaxValue - MinValue) * rand^{[0,1]}$
    Compute $F_{cost}(x^i)$
end for
while (NOT stopping criterion) **do**
    Search for x_{best} and x_{worst}
    for $i = 1$ to $PopulationSize$ **do**
        $x^{new} = x^i + rand_1^{[0,1]} \left(x_{best} - \left|x^i\right|\right) - rand_2^{[0,1]} \left(x_{worst} - \left|x^i\right|\right)$
        Check bounds $MinValue$
        Check bounds $MaxValue$
        Compute $F_{cost}(x^{new})$
        if $F_{cost}(x^{new}) < F_{cost}(x^i)$ **then**
            $x^i = x^{new}$
        end if
    end for
end while

A possible problem with heuristic optimization algorithms is getting trapped in a local minimum, which is minimized by increasing the population diversity. The use of a chaotic map increases this diversity, accelerating the speed of convergence on the one hand, and improving the exploration of the search domain on the other. The chaotic map used in this work is briefly described below, the chaotic proof of which is reported in [18].

2.2 2D chaotic map

The process of searching for the optimum in heuristic optimization algorithms is based on two phases, the exploitation phase and the exploration phase. The former is related to the radius of convergence and the latter to the ability to explore different regions of the search space. To balance both phases, the chaotic map presented in [18] generates new individuals using three individuals instead of using only the best and worst individual in the population, as generated in the original Jaya algorithm (see Algorithm 1). The additional individual is randomly selected from the current population. On the other hand, each new individual is generated by choosing an equation from three possible options (see Eqs. (2), (3) and (4)). In these equations x_{best}, x_{worst} and x_{rand} are the best individual, the worst individual and the random individual respectively, S_F (scaling factor) is an integer value equal to one or two, and $\{ch_i, i = 1, 2, \ldots 5\}$ are chaotic variables obtained from the 2D cross chaotic map and fall in the range $[0, 1]$.

$$x^{new} = ch_1 x_{rand} + ch_2 \left(x^{old} - ch_3 x_{rand}\right) + ch_4 \left(x_{best} - ch_5 x_{rand}\right) \quad (2)$$

$$x^{new} = ch_1 x_{rand} + ch_2 \left(x^{old} - ch_3 x_{rand}\right) + ch_4 \left(x_{worst} - ch_5 x_{rand}\right) \quad (3)$$

$$x^{new} = ch_1 x_{best} + ch_2 \left(x_{rand} - S_F x_{best}\right) \quad (4)$$

Algorithm 2 describes the process of generating the 2D chaotic map, where $r_1 = 0.2$, $s_1 = 0.3$, $m = i$ and $maxDim = 500$. Note that the 2D map values are in $[-1, 1]$, thus the absolute value of the numbers extracted is calculated to obtain values in $[0, 1]$.

Algorithm 2 2D Chaotic map

1: Initialise r_1, s_1 and $maxDim$
2: **for** $i = 1$ to $maxDim$ **do**
3: $\quad r_{i+1} = cos(m * arccos(s_i))$
4: $\quad s_{i+1} = 16r_i^5 - 20r_i^3 + 5r_i$
5: **end for**

The partially random process for selecting the equation to generate a new individual is detailed in Algorithm 3, in which the number of elements of each $ch_i, i = 1, 2, \ldots 5$ depends on the number of variables in the cost function.

3 Multi-level parallel algorithms

The multi-level parallel proposals presented in this work are based on both the Jaya optimization algorithm presented in Section 2.1 and the proposed use of the chaotic

Algorithm 3 Generation of new individuals

rnd^1_{int} and rnd^2_{int} are integer random numbers.
$a = min(rnd^1_{int}, rnd^2_{int})$
$b = max(rnd^1_{int}, rnd^2_{int})$
ch_6 are randomly selected chaotic values.
Select between Eqs. (2), (3) or (4):
if $ch_6 < a$ **then**
    $x^{new} = ch_1 x_{rand} + ch_2 \left(x^{old} - ch_3 x^{rand}\right) + ch_4 \left(x_{best} - ch_5 x_{rand}\right)$
end if
if $a < ch_6 < b$ **then**
    $x^{new} = ch_1 x^{rand} + ch_2 \left(x^{old} - ch_3 x_{rand}\right) + ch_4 \left(x_{worst} - ch_5 x_{rand}\right)$
end if
if $ch_6 > b$ **then**
    $x^{new} = ch_1 x_{best} + ch_2 \left(x_{rand} - S_F x_{best}\right)$
end if

map explained in Section 2.2. The first parallel level (outer-level) exploits the coarse-grained parallelism, while the second parallel level (inner-level) exploits the fine-grained parallelism.

A general flowchart of the multi-level parallel algorithms is shown in Fig. 1. Note that the total number of processing threads used depends on both the number of outer processing threads ($outNt$) and the number of inner processing threads ($innNt$). For example, if $outNt$ in Fig. 1 equals 6, and $innNt$ in the same figure equals 3, the total number of concurrent processing threads will be 18. That is, once the outer parallel region has been generated, each outer processing thread spawns its own team of processing threads in its own inner parallel region.

3.1 Outer-level parallel algorithms

Two parallel coarse-grained algorithms based on multi-population (called CP-CJaya and NCP-CJaya) have been used as outer-level parallel algorithms. These algorithms were presented in [38]. In both parallel algorithms, the whole population was split between the processing threads. Therefore, the size of the initial population limits the maximum number of processing threads to be used, as the sub-populations must meet the required minimum size.

In the parallel algorithm CP-CJaya, synchronization points are used to share information between sub-populations. In contrast, the algorithm NCP-CJaya has no synchronization points, i.e. the synchronization point P_OUTER in Fig. 1 is implemented only in the CP-CJaya algorithm, while this synchronization point is removed for the NCP-CJaya. The results presented in [38] showed that the NCP-CJaya algorithm significantly improves the CP-CJaya algorithm regarding parallel behavior. In contrast, the CP-CJaya algorithm slightly improves NCP-CJaya regarding optimization behavior. However, the NCP-CJaya algorithm requires the use of large populations to increase the number of processing threads. Some slight improvements have been applied to the coarse-grained algorithms presented in [38]. For example, as shown in Fig. 1, the sub-population of each (outer-level) thread is generated by the thread itself, instead of the sequential thread which used to calculate the size of

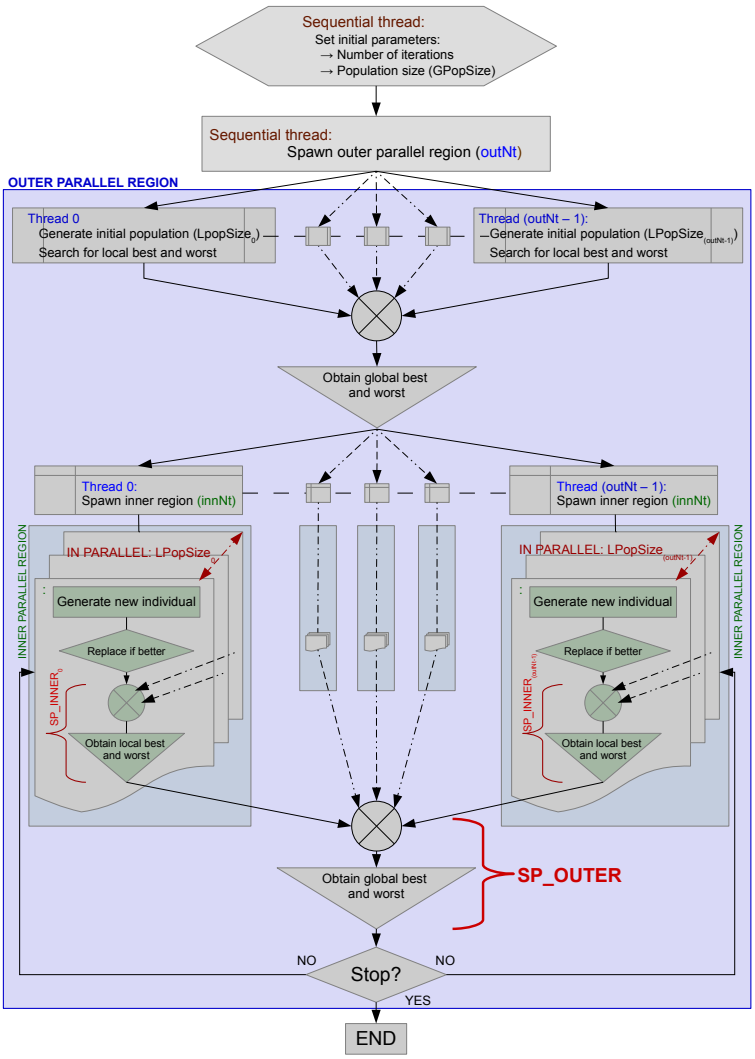


Fig. 1 General flowchart of the proposed multi-level parallel algorithms.

the sub-populations of all the threads. Note that the size of the sub-population may vary for each thread. To avoid load imbalance, the size of all sub-populations should be the same, or similar, for all outer-level processes. A detailed description of both methods, CP-CJaya and NCP-CJaya, can be found in [38], which are the methods used at the outer-level of the multi-level parallel algorithm.

The main goals of the proposed work are: (i) to achieve the parallel performance of the NCP-CJaya algorithm; (ii) to achieve the optimization performance provided by the use of the chaotic map (cf. Section 2.2); and (iii) to be able in increasing the number of processing threads without hampering the rest of the objectives. To achieve these goals, we developed multi-level parallel algorithms, using a fine-grained strategy at the inner-level, as explained below.

3.2 Inner-level parallel algorithms

In the multi-level parallel methods developed, whether using the CP-CJaya method or the NCP-CJaya method as the outer-level algorithm, each outer thread spawns its inner parallel region. The threads of each inner parallel region process the sub-population of the external generating thread (the master thread of the inner parallel area), where the synchronization point SP_INNER$_{Tid}$ of Fig. 1, which is used to obtain the best and the worst individual of the sub-population, is imperative. Fine-grained parallel proposals, due to synchronisation processes, often present problems of parallel scalability, similar to the behavior of the CP-CJaya algorithm. Worthy of note is that the actual behavior also depends on the features of the cost function to be optimized.

It is mentioning that the best and worst individuals in the Jaya algorithm are employed to ensure the population evolution (see. Algorithm 1). When the 2D chaotic map is used, an additional, randomly selected individual is also required (see Algorithm 2). The use of the third individual improves the behavior of the exploration phase of the heuristic process during optimization.

It should be noted that when using only either the CP-CJaya algorithm or the NCP-CJaya algorithm, the third individual (random individual) used is the same for the whole population. In contrast, the proposed fine-grained algorithm uses a different individual when generating each new individual. Efficient development of the inner-level algorithm requires the sub-population to be stored in a grid structure. This grid structure is used to perform a diffusion process, in which the random individual used to generate a new element is different in the generation of each new individual. Fig. 2(a) shows the grid created for a sub-population of 60 individuals, organized in a grid of ten rows by six columns. The random individual used to generate a new individual is selected from the eight available neighbors, as shown in Fig. 2(b) for the individual $(1, 2)$. Individuals located at the edges of the grid do not have eight neighbors, as is the case with the element $(5, 0)$, as shown in Fig. 2(b).

The selection of a random element (among the eight neighbors) for the generation of each new individual can increase the computational cost of the algorithm without providing any advantage. To avoid this problem, we randomly select the relative position of the neighbor to be used for the whole population (cf. Algorithm 4).

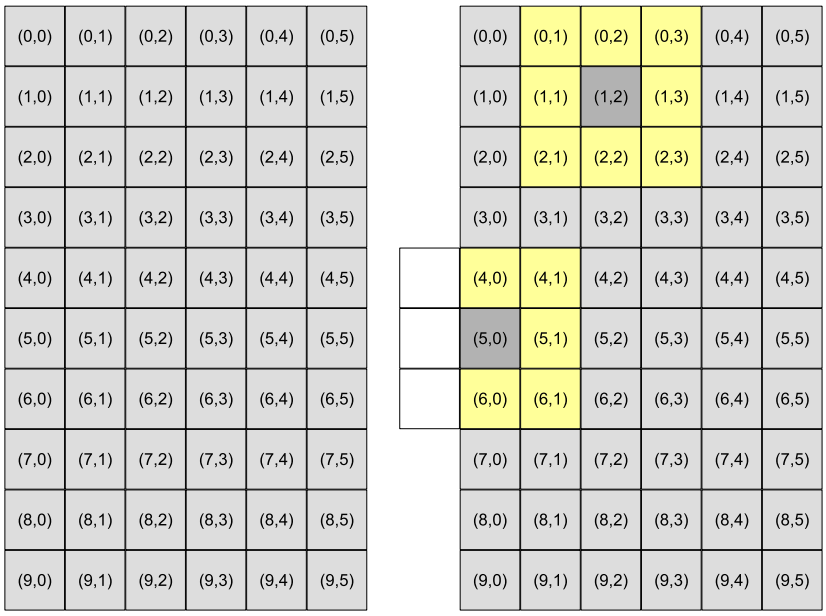


Fig. 2 Diffusion grid (10×6).

Algorithm 4 Neighbour Selection

1: Select a random number $r_i \in \{-1, 0, 1\}$
2: Select a random number $r_j \in \{-1, 0, 1\}$
3: **while** $((r_i = 0) \wedge (r_j = 0))$ **do**
4: Select $r_i \in \{-1, 0, 1\}$
5: Select $r_j \in \{-1, 0, 1\}$
6: **end while**

Fig. 3(a) shows the relative selection of the neighbor when the neighbor used as a random individual. The neighbor does not exist for some individuals; for example, the individuals $(2, -1)$ and $(10, 1)$. To solve this problem, a symmetric extension has been made, as shown in Fig. 3(b).

Algorithm 5 shows the inner-level parallel algorithm implemented. This algorithm is executed by all outer threads, each of them spawning a nested parallel region of $innNt$ threads.

4 Numerical experiments

In this section, both the parallel performance of the proposed multi-level parallel algorithms and the behavior of the optimization when solving constrained engineering problems are analyzed. The algorithms proposed here were implemented in the C programming language, using the GCC v.4.8.5 [19]. The multi-level parallel algorithms were designed for multicore platforms using the OpenMP API v3.1 [44]. The

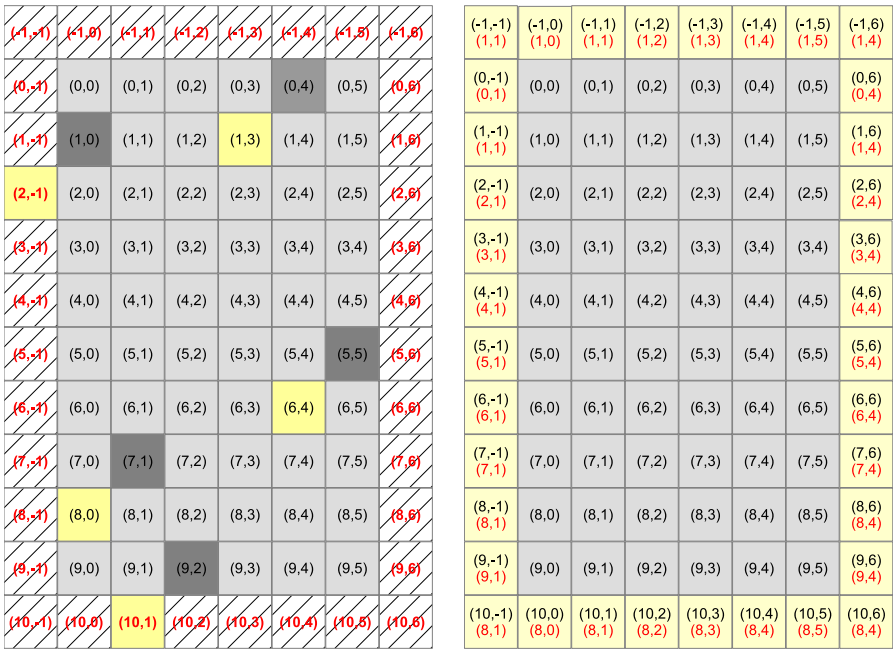


Fig. 3 Neighbour selection $r_i = 1$ and $r_j = -1$.

Algorithm 5 Inner-level algorithm.

Outer thread oT
Shared memory: sub-population^{oT}, r_i^{oT}, r_j^{oT}, i_{best}^{oT}, i_{best}^{oT}, i_{worst}^{oT} and j_{worst}^{oT}
Obtain r_i^{oT}, r_j^{oT} (cf. Algorithm 4)
Thread oT spawns inner parallel region ($innNt$ threads):
for $i = 0$ to $subpopSize_i^{oT} - 1$ **do**
  for $j = 0$ to $subpopSize_j^{oT} - 1$ **do**
    Compute the new individual x^{new} (cf. Algorithm 3)
    if $F(x^{new}) < F(x^{(i,j)})$ **then**
      $x^{(i,j)} = x^{new}$
      if $F(x^{new}) < F(x^{(i_{best}^{oT}, j_{best}^{oT})})$ **then**
        $i_{best}^{oT} = i$
        $j_{best}^{oT} = j$
      end if
    end if
    if $((i = i_{worst}^{oT}) \wedge (j = j_{worst}^{oT}))$ **then**
      Mark for search new worst $FlagWorst^{oT} = true$
    end if
  end for
end for
if $FlagWorst^{oT} = true$ **then**
  Update i_{worst}^{oT} and j_{worst}^{oT}
end if

multicore platform used was equipped with two Intel Xeon Gold 6140 processors, each of which contained eighteen 2.3 GHz processing cores. The operating system was CentOS Linux 7.6.

4.1 Benchmark sets

First, 18 well-known unconstrained functions are used to analyze the parallel behavior of the proposed parallel algorithms, as listed in Table 1 and described in Table 2. Note that the optimization behavior for unconstrained numerical functions has been analyzed in [18,39].

Table 1: Benchmarks, dimensions and domains.

Id.	**Name**	**Dim. (V)**	**Domain (Min, Max)**
F1	Sphere	30	$-100, 100$
F2	SumSquares	30	$-10, 10$
F3	Beale	2	$-4.5, 4.5$
F4	Easom	2	$-100, 100$
F5	Zakharov	10	$-5, 10$
F6	Schwefel problem 1.2	10	$-100, 100$
F7	Rosenbrock	30	$-30, 30$
F8	Branin	2	$x_1 : -5, 10; x_2 : 0, 15$
F9	Bohachevsky_1	2	$-100, 100$
F10	Booth	2	$-10, 10$
F11	Michalewicz_2	2	$0, \pi$
F12	Bohachevsky_2	2	$-100, 100$
F13	Bohachevsky_3	2	$-100, 100$
F14	GoldStein-Price	2	$-2, 2$
F15	Hartman_3	3	$0, 1$
F16	Ackley	30	$-32, 32$
F17	Langermann_2	2	$0, 10$
F18	Langermann_10	10	$0, 10$

Table 2: Benchmarks objective functions.

Id.	**Function**
F1	$f = \sum_{i=1}^{V} x_i^2$
F2	$f = \sum_{i=1}^{V} i x_i^2$
F3	$f = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2$ $+(2.625 - x_1 + x_1 x_2^3)^2$
F4	$f = -\cos(x_1)\cos(x_2)\exp\left(-(x_1 - \pi)^2 - (x_2 - \pi)^2\right)$

Table 3 Constrained engineering problems.

Id.	Name	Type
CEF1	Pressure vessel design	Minimisation
CEF2	Welded beam design	Minimisation
CEF3	Three bar truss design	Minimisation
CEF4	Tension-compression spring design	Minimisation
CEF5	Speed reducer design	Minimisation
CEF6	Belleville spring design	Minimisation
CEF7	Rolling element bearing design	Maximisation

F5 $f = \sum_{i=1}^{V} x_i^2 + \left(\sum_{i=1}^{V} 0.5 i x_i\right)^2 + \left(\sum_{i=1}^{V} 0.5 i x_i\right)^4$

F6 $f = \sum_{i=1}^{V} \left(\sum_{j=1}^{i} x_j\right)^2$

F7 $f = \sum_{i=1}^{V-1} \left(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right)$

F8 $f = \left(x_2 - \frac{5.1}{4\pi^2} x_1^2 + \frac{5}{\pi} x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$

F9 $f = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$

F10 $f = (x_1 - 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$

F11 $f = -\sum_{i=1}^{2} \sin x_i \left(\sin\left(\frac{i x_i^2}{\pi}\right)\right)^{20}$

F12 $f = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1)\cos(4\pi x_2) + 0.3$

F13 $f = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1 + 4\pi x_2) + 0.3$

F14 $f = \left[1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\right]$
$\left[30 + (2x_1 - 3x_2)^2 (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\right]$

F15 $f = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2\right]$

F16 $f = -20\exp\left(-0.2\sqrt{\frac{1}{V}\sum_{i=1}^{V} x_i^2}\right) - \exp\left(\frac{1}{V}\sum_{i=1}^{V}\cos(2\pi x_i)\right) + 20 + e$

F17
F18 $f = -\sum_{i=1}^{5} c_i \left[\exp\left(-\frac{1}{\pi}\sum_{j=1}^{V}(x_j - a_{ij})^2\right)\cos\left(\pi\sum_{j=1}^{V}(x_j - a_{ij})^2\right)\right]$

Optimization performance for constrained functions will be analyzed by solving the constrained engineering problems listed in Table 3, in the scenarios of minimization and maximization.

Pressure vessel design problem

The design problem of the pressure vessels, shown in Fig. 4, is a structural design problem. The design consists of determining four variables: the thickness of the shell

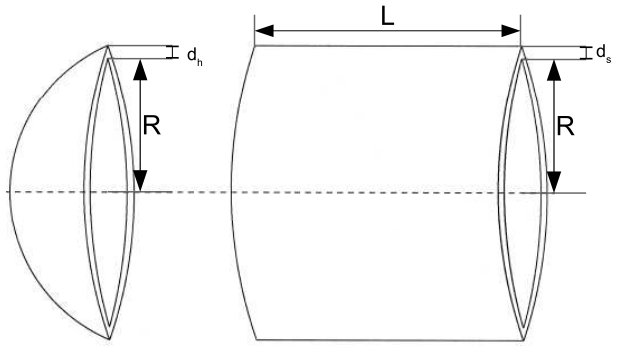


Fig. 4 Pressure vessel design problem.

(d_s), the thickness of the heads (d_h), the internal radius (R) and the length (L) of the cylindrical section. The optimization problem consists of minimizing the financial cost by meeting the non-linear stress constraints and yield criteria. Furthermore, both thicknesses (d_s and d_h) are not continuous variables, but integer multiples of 0.0625 inches. The optimisation problem can be defined as in Eq. (5).

$$
\begin{aligned}
&\text{Pressure vessel design problem:}\\
&F = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3\\
&x_1 = d_s, x_2 = d_h, x_3 = R, x_4 = L\\
&\text{Constraints:}\\
&g_1 = -x_1 + 0.0193x_3 \leq 0\\
&g_2 = -x_2 + 0.00954x_3 \leq 0\\
&g_3 = -\pi x_3^2x_4 - (4/3)\pi x_3^3 + 1296000 \leq 0\\
&g_4 = x_4 - 240 \leq 0\\
&0.0625 \leq x_1, x_2 \leq 99 * 0.0625\\
&10 \leq x_3, x_4 \leq 240
\end{aligned}
\tag{5}
$$

Welded beam design problem

This problem, shown in Fig. 5, consists of minimizing the cost of manufacturing and assembling the welded beams by considering the welding work, the cost of material and labor. The design consists of determining four variables; the thickness of the weld (h), the length of the welded joint (l), the width of the beam (t), and the thickness of the beam (b). The optimization problem definition is shown in Eq. (6), where $\tau(x)$ is the shear stress in the weld, τ_{max} the allowable shear stress of the weld, $\sigma(x)$ the normal stress in the beam, σ_{max} the allowable normal stress for the beam material, $P_c(x)$ the bar buckling load, P the load, $\delta(x)$ the beam end deflection and δ_{max} the

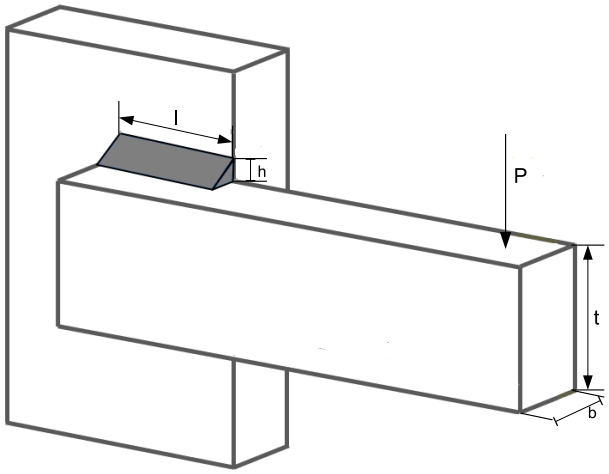


Fig. 5 Welded beam design problem.

allowable beam end deflection. Auxiliary functions and constant values are given in Eq. (7).

Welded beam design problem:

$$
\begin{aligned}
&F = 1.10471x_1^2x_2 + 0.04811x_3x_4(14.0 + x_2)\\
&x_1 = h, x_2 = l, x_3 = t, x_4 = b\\
&\text{Constraints:}\\
&g_1 = \tau(x) - \tau_{max} \leq 0\\
&g_2 = \sigma(x) - \sigma_{max} \leq 0\\
&g_3 = x_1 - x_4 \leq 0\\
&g_4 = 0.10471x_1^2 + 0.04811x_3x_4(14.0 + x_2) - 5.0 \leq 0\\
&g_5 = 0.125 - x_1 \leq 0\\
&g_6 = \delta(x) - \delta_{max} \leq 0\\
&g_7 = P(x) - P_c(x) \leq 0\\
&0.1 \leq x_1, x_4 \leq 2.0\\
&0.1 \leq x_2, x_3 \leq 10.0
\end{aligned}
\tag{6}
$$

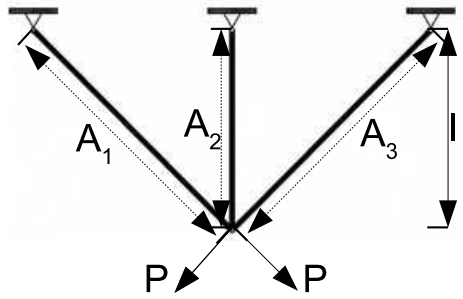


Fig. 6 Three bar truss design problem.

Auxiliary functions and constant values of welded beam problem:

$$
\begin{aligned}
\tau(x) &= \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2}; \tau' = \frac{P}{\sqrt{2}x_1x_2}; \tau'' = \frac{MR}{J} \\
M &= P\left(L + \frac{x_2}{2}\right); R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2} \\
J &= 2\left\{\sqrt{2}x_1x_2\left[\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\} \\
\sigma(x) &= \frac{6PL}{x_4x_3^2} \\
\delta(x) &= \frac{4PL^3}{Ex_x^3x_4} \\
P_c(x) &= \frac{4.013E\sqrt{\frac{x_3^2x_4^6}{36}}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right) \\
P &= 6000lb; L = 14in; \delta_{max} = 0.25in; E = 30e + 6psi; G = 10e + 6psi \\
\tau_{max} &= 13600psi; \sigma_{max} = 30000psi
\end{aligned}
\tag{7}
$$

Three bar truss design problem

When a three-bar truss structure is designed, the goal is the minimization of the volume of the structure, as shown in Fig. 6. The design variables of the cost function are the cross-sections of the structural members. Since, a symmetrical three-bar structure is considered, only cross-sections A_1 and A_2 are optimized, where $A_3 = A_1$. The optimization problem definition is shown in Eq. (8), where l is the length, P the load, and σ the stress limit.

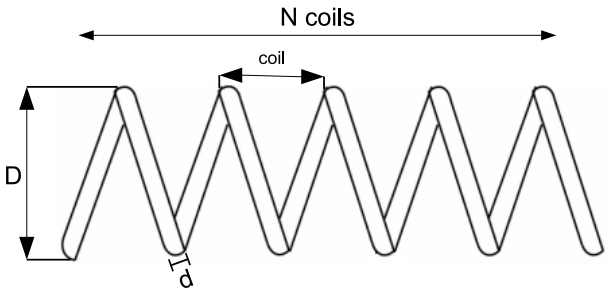


Fig. 7 Tension-compression spring design problem.

Three bar design problem:

$$
\begin{aligned}
&F = (2.0\sqrt{2.0}x_1 + x_2) * l;\\
&x_1 = A_1 = A_3, x_2 = A_2\\
&\text{Constraints:}\\
&g_1 = \left(\frac{\sqrt{2.0}x_1 + x_2}{\sqrt{2.0}x_1^2 + 2.0x_1x_2}\right) P - \sigma \leq 0\\
&g_2 = \left(\frac{x_2}{\sqrt{2.0}x_1^2 + 2.0x_1x_2}\right) P - \sigma \leq 0\\
&g_3 = \left(\frac{1.0}{\sqrt{2.0}x_2 + x_1}\right) P - \sigma \leq 0\\
&P = 2.0, l = 100.0, \sigma = 2.0\\
&0.0 \leq x_1, x_2 \leq 1.0
\end{aligned}
\tag{8}
$$

Tension-compression spring design problem

The design variables of the problem of minimizing the weight of the tension-compression spring, shown in Fig. 7, are the wire diameter (d), the mean coil diameter (D) and the number of active coils (N). The constraints are related to the minimum deflection, shear stress, surge frequency, diameter and design variables, as can be seen in Eq. (9).

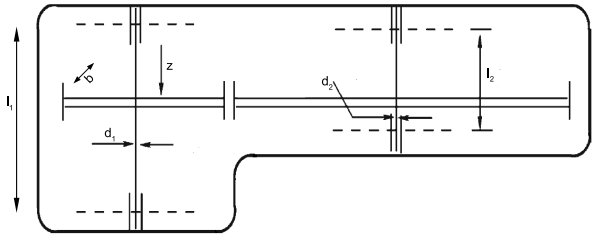


Fig. 8 Speed reducer design problem.

Tension-compression spring design problem:

$$
\begin{aligned}
&F = (x_3 + 2.0)x_2x_1^2 \\
&x_1 = d, x_2 = D, x_3 = N \\
&\text{Constraints:} \\
&g_1 = 1.0 - \left(\frac{x_2^3 x_3}{71785x_1^4}\right) \leq 0 \\
&g_2 = \frac{4.0x_2^2 - x_1x_2}{12566(x_2x_1^3 - x_1^4)} + \frac{1.0}{5108x_1^2} - 1.0 \leq 0 \\
&g_3 = 1.0 - \left(\frac{140.45x_1}{x_2^2x_3}\right) \leq 0 \\
&g_4 = \frac{x_2 - x_1}{1.5} - 1.0 \leq 0 \\
&0.05 \leq x_1 \leq 2.0 \\
&0.25 \leq x_2 \leq 1.3 \\
&2.0 \leq x_3 \leq 15.0
\end{aligned}
\tag{9}
$$

Speed reducer design problem

The goal of the speed reducer design problem, shown in Fig. 8, is to minimize the weights of the speed reducer. The constraints are related to the bending stress of the gear teeth, surface stress, transverse deflections of the shafts and stresses in the shafts. The design variables are the face width (b), the module of teeth (m), the number of teeth in the pinion (z), the length of the first shaft between bearings (l_1), the length of the second shaft between bearings (l_2), the diameter of the first shaft (d_1) and the diameter of the second shaft (d_2). The problem is formulated in Eq. (10). Note that the number of teeth in the pinion (z) is an integer, while the rest of the variables are continuous.

Speed reducer design problem:

$$
\begin{aligned}
&F = 0.7854x_1x_2^2\left(3.3333x_3^2x_3^2 + 14.9334x_3 - 43.0934\right) - 1.508x_1\left(x_6^2 + x_7^2\right)\\
&\quad + 7.4777\left(x_6^3 + x_7^3\right) + 0.7854\left(\left(x_4x_6^2\right) + \left(x_5x_7^2\right)\right)\\
&x_1 = b, x_2 = m, x_3 = z, x_4 = l_1, x_5 = l_2, x_6 = d_1, x_7 = d_2
\end{aligned}
$$

Constraints:

$$
\begin{aligned}
g_1 &= \frac{27.0}{x_1x_2^2x_3} - 1.0 \leq 0\\
g_2 &= \frac{397.5}{x_1x_2^2x_3^2} - 1.0 \leq 0\\
g_3 &= \frac{1.93x_4^3}{x_2x_3x_6^4} - 1.0 \leq 0\\
g_4 &= \frac{1.93x_5^3}{x_2x_3X - 7^4} - 1.0 \leq 0\\
g_5 &= \frac{\sqrt{\left(\frac{745.0x_4}{x_2x_3}\right)^2 + 16900000}}{110.0x_6^3} - 1.0 \leq 0\\
g_6 &= \frac{\sqrt{\left(\frac{745.0x_5}{x_2x_3}\right)^2 + 157500000}}{85.0x_7^3} - 1.0 \leq 0\\
g_7 &= \frac{x_2x_3}{40.0} - 1.00 \leq 0\\
g_8 &= \frac{5.0x_2}{x_1} - 1.00 \leq 0\\
g_9 &= \frac{x_1}{12.0x_2} - 1.00 \leq 0\\
g_{10} &= \frac{1.5x_6 + 1.9}{x_4} - 1.00 \leq 0\\
g_{11} &= \frac{1.1x_7 + 1.9}{x_5} - 1.00 \leq 0\\
&2.6 \leq x_1 \leq 3.6; 0.7 \leq x_2 \leq 0.8\\
&17 \leq x_3 \leq 28\\
&7.3 \leq x_4 \leq 8.3; 7.8 \leq x_5 \leq 8.3\\
&2.9 \leq x_6 \leq 3.9; 5.0 \leq x_7 \leq 5.5
\end{aligned}
\tag{10}
$$

Belleville spring design problem

A Belleville spring, shown in Fig. 9, should be designed with minimum weight. In this problem, the design variables are the thickness of the spring (t), the height of the

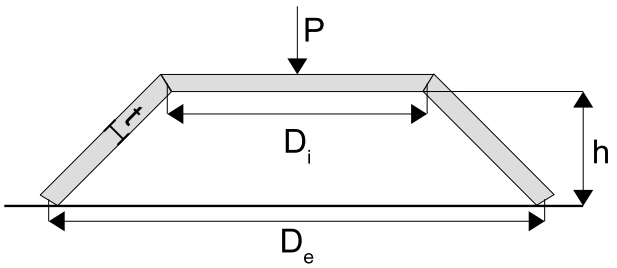


Fig. 9 Belleville spring design problem.

spring (h), the external diameter of the spring (D_e), and the internal diameter of the spring (D_i). The problem is formulated in Eq. (11), where S is the allowable strength, E is the modulus of elasticity for the spring material, μ is the Poissons ratio, δ_{max} is the maximum deflection, P_{max} is the maximum load acting on the spring, H is the overall height of the spring and D_{max} is the maximum outside diameter of the spring. Some functions definitions and constant values are given in Eqs. 12 and 13.

Belleville spring design problem:

$$
\begin{aligned}
&F = 0.07075\pi x_1^2 - x_2^2 x_3 \\
&x_1 = D_e, x_2 = D_i, x_3 = t, x_4 = h \\
&\text{Constraints:} \\
&g_1 = S - \left(\frac{4.0E\delta_{max}}{1.0-\mu^2\alpha(x)x_1^2}\right)\left(\gamma(x)x_3 + \beta(x)\left(x_4 - \frac{\delta_{max}}{2.0}\right)\right) \geq 0 \\
&g_2 = \left[\left(x_4 - \frac{\delta_{max}}{2.0}\right)(x_4 - \delta_{max})x_3 + x_3^3\right]\left(\frac{4.0E\delta_{max}}{1.0-\mu^2\alpha(x)x_1^2}\right) - P_{max} \geq 0 \\
&g_3 = \delta(x) - \delta_{max} \geq 0 \\
&g_4 = H - x_4 - x_3 \geq 0 \\
&g_5 = D_{max} - x_1 \geq 0 \\
&g_6 = x_1 - x_2 \geq 0 \\
&g_7 = 0.3 - \frac{x_4}{x_1 - x_2} \geq 0 \\
&2.0 \leq x_1, x_2 \leq 12.1 \\
&0.2 \leq x_3, x_4 \leq 2.0
\end{aligned}
\tag{11}
$$

Auxiliary functions and constant values of belleville spring design problem:

$$
\begin{aligned}
K(x) &= \frac{x_1}{x_2} \\
\beta(x) &= \left(\frac{6.0}{\pi \ln(K(x))}\right)\left(\frac{K(x)-1}{\ln(K(x)} - 1.0\right) \\
\alpha(x) &= \left(\frac{6.0}{\pi \ln(K(x))}\right)\left(\frac{K(x)-1}{K(x)}\right)^2 \\
\gamma(x) &= \left(\frac{6.0}{\pi \ln(K(x))}\right)\left(\frac{K(x)-1}{2.0}\right)^2 \\
a(x) &= \frac{x_4}{x_3} \\
\delta(x) &= f_a(a(x))a(x) \\
S &= 200000.0; E = 2.0; \mu = 0.3; D_{max} = 12.01; E = 30e \\
P_{max} &= 5400.0; \delta_{max} = 0.2
\end{aligned}
\tag{12}
$$

$$
\begin{aligned}
a = &< 1.4\ 1.50\ 1.60\ 1.70\ 1.80\ 1.90\ 2.00\ 2.10\ 2.20\ 2.30\ 2.40\ 2.50\ 2.60\ 2.70 > \\
f_a(a) = &\ 1.0\ 1.0\ 0.85\ 0.77\ 0.71\ 0.66\ 0.63\ 0.60\ 0.58\ 0.56\ 0.55\ 0.53\ 0.52\ 0.51\ 0.51\ 0.5
\end{aligned}
\tag{13}
$$

Rolling element bearing design problem

The rolling element bearing design problem is a maximization problem aimed at maximizing the dynamic load capacity of a rolling element bearing. This problem, shown in Fig. 10, has five decision variables, namely pitch diameter (D_m), ball diameter (D_b), number of balls (Z), curvature radius coefficient of inner raceway groove ($f_i = r_i/D_b$), curvature radius coefficient of outer raceway groove ($f_o = r_o/D_b$), and r_i and r_o are the inner and outer ring groove curvature ratio. It also has five constraints constants, K_{Dmin}, K_{Dmax}, ϵ, e and ψ, as formulated in Eq. (14).

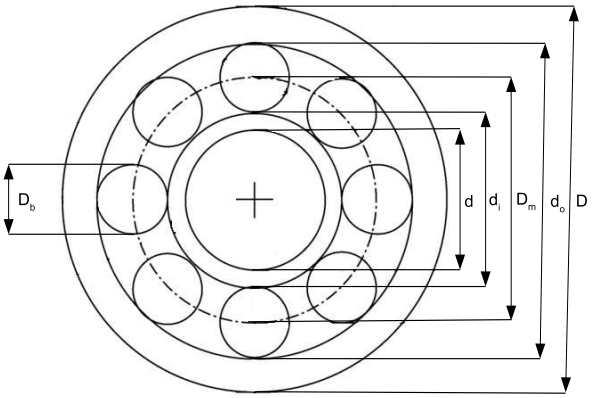


Fig. 10 Rolling element bearing design problem.

Rolling element bearing design problem:

$$
\begin{aligned}
&F = f_c x_3^{2/3} x_2^{1.8}; \text{if } x_2 \leq 25.4 \\
&F = 3.647 f_c x_3^{2/3} x_2^{1.4}; \text{if } x_2 > 25.4 \\
&x_1 = D_m, x_2 = D_b, x_3 = Z, x_4 = f_i, x_5 = f_o \\
&\text{Constraints:} \\
&g_1 = \frac{\phi_0}{2\sin^{-1}\frac{x_2}{x_1}} - x_3 + 1 \geq 0 \\
&g_2 = 2.0x_2 - x_6(D-d) \geq 0 \\
&g_3 = x_7(D-d) - 2.0x_2 \geq 0 \\
&g_4 = x_{10}B_w - x_2 \geq 0 \\
&g_5 = x_1 - 0.5(D+d) \geq 0 \\
&g_6 = (0.5+x_9)(D+d) - x_1 \geq 0 \\
&g_7 = 0.5(D-x_1-x_2) - x_8 x_2 \geq 0 \\
&g_8 = x_4 - 0.515 \geq 0 \\
&g_9 = x_5 - 0.515 \geq 0 \\
&x_6 = K_{Dmin}, x_7 = K_{Dmax}, x_8 = \epsilon, x_9 = e, x_5 = \psi
\end{aligned}
\tag{14}
$$

Auxiliary functions and constant values of rolling element bearing design problem:

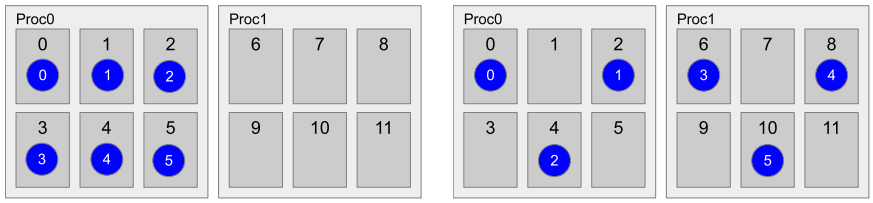
$$
\begin{aligned}
&\gamma = \frac{D_b cos\alpha}{D_m} \\
&f_c = 37.91 \left\{ 1 + \left[1.04 \left(\frac{1-\gamma}{1+\gamma} \right)^{1.72} \left(\frac{f_i(2f_o - 1)}{f_o(2f_i - 1)} \right)^{0.41} \right]^{10/3} \right\}^{-0.3} \\
&\quad \left\{ \left(\frac{\gamma^{0.3}(1-\gamma)^{1.39}}{(1+\gamma)^{1/3}} \right) \left(\frac{2f_i}{2f_i - 1} \right)^{0.41} \right\} \\
&T = D - d - (2.0x_2) \\
&\phi_0 = 2\pi - 2\cos^{-1} \left(\frac{\left(\frac{D-d}{2} - \frac{3T}{4} \right)^2 + \left(\frac{D}{2} - \frac{T}{4} - x_2 \right)^2 - \left(\frac{d}{2} + \frac{T}{4} \right)^2}{2\left(\frac{D-d}{3} - \frac{3T}{4} \right)\left(\frac{D}{2} - \frac{T}{4} - x_2 \right)} \right) \\
&D = 160; d = 90; B_w = 30; \alpha = 0 \\
&\quad 90.0 \leq x_1 \leq 150.0 \\
&\quad 10.5 \leq x_2 \leq 31.5 \\
&\quad 4 \leq x_3 \leq 50 \\
&\quad 0.515 \leq x_4, x_5 \leq 0.6 \\
&\quad 0.4 \leq x_6 \leq 0.5 \\
&\quad 0.6 \leq x_7 \leq 0.7 \\
&\quad 0.3 \leq x_8 \leq 0.4 \\
&\quad 0.02 \leq x_9 \leq 1.0 \\
&\quad 0.6 \leq x_{10} \leq 0.85
\end{aligned}
\tag{15}
$$

4.2 Thread affinity of multi-level parallel algorithms

Thread affinity can be defined as the method used to determine the policy for assigning the threads to the processing units (or cores). Since a multi-level parallel algorithm has been designed using nested parallelism, it is crucial to explicitly define the thread affinity to exploit the parallel platform efficiently.

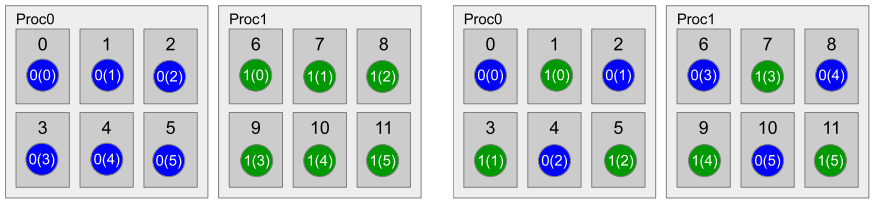
As earlier mentioned, the used parallel platform was equipped with two processors, both being multi-core. Fig. 11 shows the thread placement when the thread binding affinity is set to $close$ (Fig. 11(a)), and $spread$ (Fig. 11(b)), considering a platform with two processors of only six cores. There are applications where there is no significant difference in computing performance between the two strategies, but, depending on the application, in particular, the pattern of memory usage and synchronizations, one strategy or the other may be advisable.

However, by using a parallel multi-level strategy (through nested parallelism), the use of a $close$ strategy on the outer level causes the inner level threads to be



(a) (b)

Fig. 11 Outer level affinity approaches.



(a) (b)

Fig. 12 Multi-level affinity approaches.

$separate$, as can be seen in Fig. 12(a). In Fig. 12, each thread is labeled by a tuple $i(j)$, where j is the identification of the outer thread and i is the identification of the inner one. Therefore, this figure shows a multi-level strategy with six outer threads and two inner threads in each outer parallel region, i.e., a total of 12 threads.

The most convenient thread placement for our parallel algorithms is shown in Fig. 12(b), in which the outer threads are placed following the $spread$ strategy, while the inner threads are placed following the $close$ strategy. OpenMP thread affinity policies have been used to set this thread placement.

4.3 Parallel performance analysis using nested parallelism

To analyze the parallel performance of the proposed multi-level parallel algorithms, first we make use of the 18 functions mentioned above. As stated in Section 3, the outer algorithms used in the proposed multi-level parallel were characterized in [38]. Tables 4 and 5 show the speed-up values when a total of 12 processes are used in the multilevel parallel algorithm, but varying the number of both the outer and inner processes. In both tables, the population size equals 240, and 30 independent executions are carried out with 50,000 iterations in each of them. As can be seen, the results are significantly worse than those obtained in [38], i.e. using the outer (one-level) parallel algorithm. This is mainly due to the use of the OpenMP nested regions, but also to the use of the diffusion grid, which changes the storage structures and therefore the computing performed. Note that each outer thread generates and destroys an internal region in each of the 50,000 iterations, increasing the parallel overhead. This also implies that the number of synchronizations grows, since the nested parallel areas indicate additional synchronization points at the end of it. In these algorithms, the

	$(outNt, innNt)$			
	(6,2)	(4,3)	(3,4)	(2,6)
F1	2.7	2.0	1.8	1.5
F2	2.9	2.0	2.1	1.5
F3	0.4	0.2	0.2	0.2
F4	0.4	0.3	0.2	0.2
F5	1.0	0.8	0.7	0.5
F6	3.7	2.4	2.4	1.8
F7	1.7	1.4	1.3	1.1
F8	0.3	0.2	0.2	0.1
F9	0.2	0.2	0.1	0.1
F10	0.2	0.1	0.1	0.1
F11	0.6	0.4	0.4	0.3
F12	0.2	0.2	0.1	0.1
F13	0.2	0.2	0.1	0.1
F14	0.2	0.1	0.1	0.1
F15	0.5	0.4	0.3	0.2
F16	1.2	0.9	0.8	0.7
F17	0.8	0.5	0.5	0.4
F18	1.3	1.0	1.0	0.7

Table 4 Speed-up for multi-level parallel algorithm using nested parallelism and CP-CJaya as the outer algorithm. Population size = 240, Runs = 30, and Iterations = 50000.

	$(outNt, innNt)$			
	(6,2)	(4,3)	(3,4)	(2,6)
F1	8.9	7.9	6.6	5.9
F2	9.0	7.9	6.6	5.9
F3	2.4	1.5	1.0	0.8
F4	2.5	1.5	1.0	0.8
F5	5.6	4.2	3.2	2.4
F6	9.4	8.6	7.3	6.5
F7	8.0	6.7	5.2	4.4
F8	1.9	1.2	0.8	0.6
F9	1.7	1.0	0.7	0.5
F10	1.5	0.9	0.6	0.5
F11	3.7	2.4	1.6	1.3
F12	1.6	1.0	0.6	0.5
F13	1.6	1.0	0.6	0.5
F14	1.6	1.0	0.6	0.5
F15	3.2	2.1	1.3	1.1
F16	6.5	4.9	3.4	2.8
F17	4.6	3.2	2.1	1.7
F18	6.7	5.2	3.9	3.1

Table 5 Speed-up for multi-level parallel algorithm using nested parallelism and NCP-CJaya as the outer algorithm. Population size = 240, Runs = 30, and Iterations = 50000.

parallel for of OpenMP has been used. When the NCP-CJaya algorithm is used as the outer algorithm, the speed-up values improve, but these parallel algorithms are useless for a considerable number of processes. It should be noted that the use of parallel nested regions in OpenMP has an intrinsic overhead [14], depending on the characteristics of those regions. If the computational load of these inner parallel regions is not costly enough, it may make this type of parallel development unfeasible.

Our parallel multi-level algorithms aim to increase the number of processes without drastically decreasing the size of the sub-populations. In this way, the size of the sub-populations to be used can be fixed, and the number of processes can be increased. This goal will be met if the parallel scalability of the multi-level algorithms behaves appropriately. Tables 4 and 5 show that the parallel performance of the multi-level parallel algorithms is not satisfactory when nested parallelism is used. As stated, this undesirable behavior is due to the overhead induced by a large number of nested parallel regions to be generated. This unwanted behavior will be solved by using process teams instead of nested parallel regions, as outlined below.

4.4 Parallel performance analysis using process teams

Parallel processing of parallel algorithms using process teams is similar to the processing of algorithms using nested parallel regions. The main difference is that when using process teams, all parallel processes will be generated only once, remaining active during all the processing. However, when using nested parallel regions, the outer threads generate and destroy the nested parallel regions for each new generation of individuals. The process mapping remains unchanged from the procedure explained in Section 4.2, when OpenMP thread affinity is used.

For this purpose, the parallel threads will have three identifiers: the global thread identifier, which is a unique identifier; the process team identifier, which matches the outer thread identifier; and the identifier within the process team, which matches the thread identifier inside the nested regions. Algorithm 6 shows the computing of these identifiers.

Algorithm 6 Thread identifiers computing

1: Number of process teams n^0Teams
2: Number of process per team $n^0ThsxTeam$
3: Number of threads $NoTs = n^0Teams \times n^0ThsxTeam$
4: Sequential thread spawns parallel region of $NoTs$ threads
5: Global thread identifier G_{Tid} (using $omp\_get\_thread\_num()$ function)
6: Team identifier $T_{Tid} = floor(n^0TeamsTeam \ / \ n^0ThsxTeam)$
7: ID inside Team $ID_{team} = (n^0Teams \ \% \ n^0ThsxTeam)$

As shown in Tables 6 and 7, when using process teams and the multi-level parallel algorithms, the parallel scalability improves significantly with respect to using nested parallel regions. Note that by using the NCP-CJaya algorithm (cf. Algorithm 7) instead of the CP-CJaya algorithm (cf. Algorithm 6) as the outer algorithm, the speed-up values are significantly improved. The two key results are that, on the one hand, the average cost of the efficiency (higher than 87%) allows the NCP-CJaya parallel algorithm, using both diffusion grid and process teams, to be applicable in real engineering problems. On the other hand, the parallel behavior does not depend on the number of process teams or the number of processes in each process team. Therefore, it is possible to set the size of the sub-populations without being related to the parallel behavior.

	$(n^0Teams\ ,\ n^0ThsxTeam)$			
	(6,2)	(4,3)	(3,4)	(2,6)
F1	9.2	9.1	9.1	9.1
F2	8.0	8.1	8.0	8.1
F3	4.4	6.0	5.6	5.8
F4	4.5	6.2	6.0	6.0
F5	6.9	6.3	6.4	6.5
F6	9.9	9.8	9.8	9.9
F7	7.6	9.4	8.0	8.9
F8	3.9	5.4	5.2	5.3
F9	3.5	4.9	5.1	5.0
F10	3.0	4.7	4.4	4.4
F11	5.8	6.7	6.6	6.8
F12	3.4	5.1	5.1	4.9
F13	3.3	4.9	5.0	4.9
F14	3.2	4.7	4.5	4.7
F15	7.1	7.1	6.9	7.1
F16	8.4	8.5	8.4	9.3
F17	7.1	8.3	8.2	8.3
F18	9.4	8.7	8.6	8.7

Table 6 Speed-up for multi-level parallel algorithm using process teams and CP-CJaya as the outer algorithm. Population size = 240, Runs = 30, Iterations = 50000, and 12 processes.

	$(n^0Teams\ ,\ n^0ThsxTeam)$			
	(6,2)	(4,3)	(3,4)	(2,6)
F1	10.8	10.8	10.8	10.8
F2	10.8	10.8	10.8	10.8
F3	10.5	10.1	9.1	9.8
F4	10.7	10.5	9.7	10.1
F5	9.9	10.3	10.0	11.1
F6	10.9	10.9	10.9	10.9
F7	10.9	10.9	10.7	10.8
F8	10.6	10.4	9.5	10.0
F9	10.5	10.3	10.3	10.4
F10	10.6	10.4	9.3	9.6
F11	10.5	10.3	9.8	10.3
F12	10.4	10.5	10.4	10.3
F13	10.2	10.4	10.3	10.2
F14	10.6	10.3	9.4	9.5
F15	10.7	10.7	10.0	10.4
F16	11.0	11.0	10.9	11.0
F17	10.9	10.8	10.4	10.6
F18	10.8	10.8	10.6	10.8

Table 7 Speed-up for multi-level parallel algorithm using process teams and NCP-CJaya as the outer algorithm. Population size = 240, Runs = 30, Iterations = 50000, and 12 processes.

Moreover, by using NCP-CJaya as the outer algorithm and using process teams to handle each sub-population, all synchronization points can be removed. This absence of synchronization points allows, if necessary, each sub-population to be of a different size, which causes an evident load imbalance, but provides load balancing by varying the number of generations associated with each sub-population.

Tables 8 and 9 show the acceleration when 20 and 30 processes are used for the most computationally expensive functions, efficiently exploiting the parallel ar-

	$((n^0Teams\ ,\ n^0ThsxTeam))$			
	(10,2)	(5,4)	(4,5)	(2,10)
F1	15.8	15.6	15.4	15.5
F2	15.5	15.3	15.3	15.3
F6	15.3	15.4	15.4	15.5
F7	15.0	14.8	14.7	14.6
F16	14.7	14.8	14.8	14.8
F17	14.4	14.2	14.1	13.6
F18	14.6	14.4	14.4	14.2

Table 8 Speed-up for multi-level parallel algorithm using process teams and NCP-CJaya as the outer algorithm. Population size = 240, Runs = 30, Iterations = 50000, and 20 processes.

	$((n^0Teams\ ,\ n^0ThsxTeam))$					
	(15,2)	(10,3)	(6,5)	(5,6)	(3,10)	(2,15)
F1	22.8	22.2	22.0	21.9	21.8	22.3
F2	23.7	23.0	22.8	22.7	22.8	23.3
F6	22.7	22.8	22.7	23.0	23.0	22.8
F7	22.1	22.0	21.5	21.8	21.3	21.2
F16	21.6	21.9	21.8	21.8	21.7	21.7
F17	20.9	21.0	18.6	19.4	18.5	18.3
F18	21.7	21.6	20.8	21.1	20.9	20.6

Table 9 Speed-up for multi-level parallel algorithm using process teams and NCP-CJaya as the outer algorithm. Population size = 240, Runs = 30, Iterations = 50000, and 30 processes.

chitecture. That is, the algorithm offers good parallel scalability, even though the processing associated with the inner level has some drawbacks to exploit the parallel platform efficiently. The main drawbacks are the memory storage structure of the sub-populations and that this memory must be shared (i.e., global memory), to be accessed by all the processes in the same equipment. Despite this, maximum efficiency values close to 80% are obtained when both 20 and 30 processes are used.

4.5 Optimisation performance analysis by solving constrained engineering problems

The optimisation behaviour will be analysed by applying the algorithms developed to the real design engineering problems listed in Table 3.

Mainly three recent optimization algorithms will be used to perform the comparative analysis with our proposal as well as other well-known ones. These algorithms are also focused on resolving engineering design problems. These algorithms are a) SSBSA [61] that modifies backtracking search optimization algorithm based on the two strategies, the species evolution rule, and the simulated annealing principle; b) MSFWA [24] a hybrid algorithm based on moth search and fireworks algorithm; c) HMPA [8] a hybrid multi-population algorithm based on artificial ecosystem-based and Harris Hawks optimization algorithm.

It is worth noting that after efficiently accelerating our algorithm, the main goal is to analyze the improvement of our proposal's exploitation capability, i.e., the quality of the obtained solution. Therefore, we will make a comparative analysis of the obtained solutions by the reference algorithms used, then check if those solutions are

	CDE	GA	HMPA	SSBSA	CJAYA
x_1	0.812500	0.812500	~~0.778168~~	0.812500	0.812500
x_2	0.437500	0.437500	~~0.384649~~	0.437500	0.437500
x_3	42.098411	42.097400	40.319610	42.098497	42.098159
x_4	176.637690	176.654000	200.000000	176.635967	176.640714
g_1	-0.000001	-0.000020	0.000000	~~0.000001~~	-0.000006
g_2	-0.035881	-0.035891	0.000000	-0.035880	-0.035884
g_3	-3.705124	-24.759376	~~0.620248~~	-0.045299	-3.155467
g_4	-63.362310	-63.346000	-40.000000	-63.364033	-63.359286
f	**6059.734106**	6059.945646	5885.326000	6059.708265	6059.762489

Table 10 Comparison of the best solutions for the pressure vessel design problem.

feasible or not. That is, they meet all the constraints of the engineering design problem, including the constraints of the optimization problem and the allowed values of some of the variables. Accordingly, we have only included references that provide both the cost function and the solution's design variables to analyze the constraints. Moreover, only the design variables values have been taken from the references. The value of the cost function and the value of the restrictions have been calculated to validate the results and make them fully reliable correctly.

Table 10 compares the obtained best solutions for the pressure vessel design problem. The reference algorithms in Table 10 are SSBSA and HMPA. MSFWA does not provide results for pressure vessel problem. In addition, the best results presented in [61,33,25,12] are included in the same table. We note that in Table 10 and the following ones, the best feasible result of the cost function will be marked in bold, the infeasible values of both design variables and constraints will be crossed out. In those algorithms in which some of their design variables are not feasible, or some constraint has infringed, the cost function's value will be grayed out instead of black. In this case, our proposal does not obtain the best result, but it achieves the second-best feasible result after the CDE method. It is mentioning that our method uses a fixed penalty system and does not require configuration parameters. In contrast, the CDE method modifies the penalty method, leading to an increase in the tuning parameters of the original differential algorithm (DE).

In addition to the three main reference algorithms (SSBSA, MSFWA, and HMPA), the crow search algorithm (CSA) [6], the HPSO algorithm [25], and the GA algorithm [12] are compared to the CJaya algorithm in Table 11, when solving the welded beam design problem. The proposed algorithm (CJaya) achieves a feasible result that significantly outperforms all other methods, whether they are feasible.

Table 12 compares the obtained best solutions for the three-bar truss design problem. The reference algorithms in Table 12 are SSBSA and MSFWA. HMPA does not provide results for the three-bar truss design problem. Besides, the algorithms DEDS [66], PSO-DE [34], and PSOSCALF [10] are added to extend the comparative study. The proposed algorithm and the PSOSCALF algorithm reach the best feasible solution. At the same time, our proposal is more efficient since PSOSCALF proposes the hybridization of three different methods: the particle swarm optimization algorithm, the sine cosine algorithm, and the Levy flight algorithm. Moreover, the proposed algorithm is free of tuning parameters.

	CSA	HPSO	GA	HMPA	MSFWA	SSBSA	CJAYA
x_1	0.205730	0.205730	0.205986	0.205729	0.201966	0.205731	0.168005
x_2	3.470489	3.470489	3.471328	3.253120	3.338861	3.470467	4.067010
x_3	9.036624	9.036624	9.020224	9.036623	9.085440	9.036624	10.000000
x_4	0.205730	0.205730	0.206480	0.205729	0.205506	0.205730	0.168007
g_1	-0.025400	-0.025400	-0.103050	~~724.6282~~	~~632.5804~~	-0.023906	-0.214219
g_2	-0.053122	-0.053122	-0.231748	~~0.099340~~	-289.2168	-0.053122	-1.249948
g_3	0.000000	0.000000	-0.000494	0.000000	-0.003540	~~0.000001~~	-0.000002
g_4	-3.432981	-3.432981	-3.430044	-3.452431	-3.438235	-3.432983	-3.536721
g_5	-0.080730	-0.080730	-0.080986	-0.080729	-0.076966	-0.080731	-0.043005
g_6	-0.235540	-0.235540	-0.235514	-0.235540	-0.235757	-0.235540	-0.236934
g_7	-12036.68	-12036.68	-12196.35	-12036.42	-12090.17	-12036.68	-5017.43
f	1.724856	1.724856	1.728225	1.695241	1.707948	1.724854	**1.587138**

Table 11 Comparison of the best solutions for the welded beam design problem.

	DEDS	PSO-DE	PSOSCALF	MSFWA	SSSBA	CJAYA
x_1	0.7886751300	0.7886751000	0.7886622460	0.7886729700	0.7886750000	0.7886925585
x_2	0.4082482800	0.4082482000	0.4082847470	0.4082544300	0.4082480000	0.4081990117
g_1	~~0.00000018~~	~~0.000000143~~	-0.000000001	-0.000000013	~~0.000000509~~	-0.000000001
g_2	-1.464102	-1.464102	-1.464060	-1.464095	-1.464102	-1.464158
g_3	-0.535898	-0.535898	-0.535940	-0.535905	-0.535898	-0.535842
f	263.895841	263.895825	**263.895844**	263.895845	263.895776	**263.895844**

Table 12 Comparison of the best solutions obtained for the three-bar truss design problem.

	CDE	SDO	HMPA	MSFWA	SSBSA	CJAYA
x_1	0.05160900	0.05189800	0.05160800	0.05085632	0.05174300	0.05194400
x_2	0.35471400	0.36175200	0.35478800	0.36680200	0.35801700	0.36287300
x_3	11.41083100	10.74794600	11.40295000	9.83231005	11.21318700	10.93758000
g_1	-0.00003864	~~0.02293804~~	-0.00005104	-0.01050210	~~0.00000026~~	-0.00002290
g_2	-0.00018289	-0.00002921	~~0.00003907~~	~~0.07054573~~	~~0.00000055~~	-0.00001987
g_3	-4.04862664	-4.18232932	-4.04991100	-4.39942800	-4.05634834	-4.06554970
g_4	-0.79793000	-0.79343067	-0.79788000	-0.78936950	-0.79581733	-0.79271400
f	0.01267024	0.01242088	0.01266495	0.01122512	0.01242088	**0.01242088**

Table 13 Comparison of the best solutions obtained for the tension-compression spring design problem.

In addition to the three main reference algorithms (SSBSA, MSFWA, and HMPA), the supply-demand-based optimization (SDO) [67] and the coevolutionary differential evolution (CDE) [33] algorithms are compared to the CJaya algorithm in Table 13, when solving the tension-compression spring design problem. Our proposal significantly outperforms all other feasible results reached by the tested algorithms.

Table 14 compares the obtained best solutions for the speed reducer design problem. The reference algorithms in Table 12 are SSBSA and HMPA. MSFWA does not provide results for the three-bar truss design problem. Besides, the algorithms DEDS [66], PSO-DE [34], and the modified differential evolution (MDE) [37] are included to extend the comparison analysis. It is found that the proposed algorithm and the MDE algorithm have the best feasible solution.

The SSBSA and the main references included in [61], i.e., the crow search algorithm (CSA) [6], the mine blast algorithm (MBA) [57], the TLBO algorithm [52], and the GA algorithm [11] are compared to the CJaya algorithm in Table 15 when solving the Belleville spring design problem. On the one hand, our proposal achieves the best

	DEDS	PSO-DE	MDE	HMPA	SSBSA	CJAYA
x_1	3.50000000	3.50000000	3.50001000	3.50000000	3.49999700	3.50000275
x_2	0.70000000	0.70000000	0.70000000	0.70000000	0.70000000	0.70000000
x_3	17.00000000	17.00000000	17.00000000	17.00000000	17.00000000	17.00000000
x_4	7.30000000	7.30000000	7.30015600	7.30000000	7.30000600	7.30000000
x_5	7.71531900	7.80000000	7.80002700	7.80000000	7.71531100	7.80000000
x_6	3.35021400	3.35021400	3.35022100	2.90000000	3.35021400	3.35042053
x_7	5.28665400	5.28668320	5.28668500	5.28668000	5.28665300	5.28689438
g_1	-0.07391528	-0.07391528	-0.07391793	-0.07391528	-0.07391449	-0.07391601
g_2	-0.19799850	-0.19799850	-0.19800082	-0.19799850	-0.19799784	-0.19799916
g_3	-0.49917180	-0.49917180	-0.49914393	-0.10795460	-0.49917061	-0.49929533
g_4	-0.90464390	-0.90147170	-0.90147081	-0.90147150	-0.90464413	-0.90148744
g_5	~~0.00000060~~	~~0.00000060~~	-0.00000541	~~0.54178530~~	~~0.00000061~~	-0.00018432
g_6	~~0.00000026~~	~~0.00000002~~	-0.00000100	~~0.00000183~~	~~0.00000083~~	-0.00011981
g_7	-0.70250000	-0.70250000	-0.70250000	-0.70250000	-0.70250000	-0.70250000
g_8	0.00000000	0.00000000	-0.00000286	0.00000000	0.00000086	-0.00000079
g_9	-0.58333330	-0.58333330	-0.58333214	-0.58333330	-0.58333369	-0.58333301
g_10	-0.05132589	-0.05132589	-0.05134472	-0.14383560	-0.05132667	-0.05128345
g_11	0.00000005	-0.01085237	-0.01085554	-0.01085282	0.00000095	-0.01082259
f	2994.4706	2996.3480	**2996.3568**	2896.2572	2994.4686	2996.5360

Table 14 Comparison of the best solutions obtained for the speed reducer design problem.

	GA5	TLBO	MBA	CSA	SSBSA	CJAYA
x_1	11.067000	12.010000	12.0100000	12.010000	12.009971	12.010000
x_2	8.751000	10.030470	10.0304732	10.030473	10.030442	10.030464
x_3	0.208000	0.204143	0.2041430	0.204143	0.204143	0.204144
x_4	0.200000	0.200000	0.2000000	0.200000	~~0.199999~~	0.200000
g_1	2145.410858	0.505662	0.2590730	0.259073	0.494705	0.334706
g_2	39.750183	~~-0.036223~~	~~-0.0288228~~	~~-0.028823~~	~~-0.032427~~	0.028332
g_3	0.761538	0.779705	0.7797054	0.779705	0.779701	0.779701
g_4	1.592000	1.595857	1.5958570	1.595857	1.595858	1.595856
g_5	0.943000	0.000000	0.0000000	0.000000	0.000029	0.000000
g_6	2.316000	1.979530	1.9795270	1.979527	1.979529	1.979536
g_7	0.213644	0.198966	0.1989658	0.198966	0.198966	0.198966
f	2.121964	1.979674	1.9796716	1.979672	1.979668	**1.979689**

Table 15 Comparison of the best solutions obtained for the Belleville spring design problem.

result by satisfying all constraints and ranges of all variables. On the other hand, it is significantly outperforming the GA method that presents a feasible solution.

For the rolling element bearing design problem, the genetic algorithm (GA) [50], the TLBO algorithm [52], the mine blasting algorithm (MBA) [57], and the supply demand-based optimization algorithm (SDO) [67] are compared in Table 16. None of the three main algorithms (SSBSA, MSFWA, and HMPA)reach a feasible solution results for this problem. The proposed algorithm and the GA algorithm provide similar feasible results for this problem.

5 Conclusions

We have developed multi-level parallel algorithms based on both multi-populations and a grid diffusion using our 2D chaotic Jaya proposal. These algorithms have been initially characterized in terms of parallel efficiency and scalability, obtaining an efficient and scalable algorithm based on the team process. These parallel algorithms allow increasing the number of processes without degrading the parallel efficiency

	GA4	TLBO	MBA	SDO	CJAYA
x_1	125.717100	125.719100	125.715300	125.700000	125.719128
x_2	21.423000	21.425590	21.423300	21.424905	21.425389
x_3	11.000000	11.000000	11.000000	11.000000	11.000000
x_4	0.515000	0.515000	0.515000	0.515002	0.515000
x_5	0.515000	0.515000	0.515000	0.515930	0.515000
x_6	0.415900	0.424266	0.488805	0.487755	0.500000
x_7	0.651000	0.633948	0.627829	0.629992	0.678698
x_8	0.300043	0.300000	0.300149	0.300039	0.300000
x_9	0.022300	0.068858	0.097305	0.053510	0.020000
x_{10}	0.751000	0.799498	0.646095	0.665982	0.850000
g_1	0.000822	0.000004	0.000564	~~-0.001272~~	0.000082
g_2	13.733000	13.152560	8.630250	8.706960	7.850778
g_3	2.724000	1.525180	1.101430	1.249630	4.658082
g_4	1.107000	2.559350	~~-2.040450~~	~~-1.445445~~	4.074611
g_5	0.717100	0.719100	0.715300	0.700000	0.719128
g_6	4.857900	16.495400	23.610950	12.677500	4.280872
g_7	0.002129	~~-0.000022~~	0.000518	0.009240	0.000125
g_8	0.000000	0.000000	0.000000	0.000002	0.000000
g_9	0.000000	0.000000	0.000000	0.000930	0.000000
f	81841.511	81859.738	81843.686	81575.185	**81858.318**

Table 16 Comparison of the best solutions obtained for the rolling element bearing design problem.

and without decreasing the sub-populations to unacceptable sizes. The desired size of the sub-populations can decide the number of outer and inner processes to be used without degrading the parallel performance. Finally, we have proven that the proposed algorithm has excellent behavior when solving engineering design problems. It obtains feasible solutions that outperform the ones acheived by the state-of-the-art algorithms. Since our proposal neither requires parameters tuning nor hybridizes several algorithms, nor uses improvements in penalty mechanisms, it may require more iterations to obtain the best feasible solution. Still, in terms of computational complexity, it can be obtained early through our parallel proposals. As future lines, we intend to hybridize several algorithms but without requiring any configuration parameter.

References

1. Ahrari, A., Atai, A.A.: Grenade explosion methoda novel tool for optimization of multimodal functions. Applied Soft Computing **10**(4), 1132 – 1140 (2010). DOI 10.1016/j.asoc.2009.11.032
2. Alatas, B.: Chaotic bee colony algorithms for global numerical optimization. Expert Systems with Applications **37**(8), 5682 – 5687 (2010). DOI 10.1016/j.eswa.2010.02.042
3. Alba, E., Tomassini, M.: Parallelism and evolutionary algorithms. IEEE Transactions on Evolutionary Computation **6**(5), 443–462 (2002). DOI 10.1109/TEVC.2002.800880
4. Alba, E., Troya, J.M.: A survey of parallel distributed genetic algorithms. Complex. **4**(4), 3152 (1999). DOI 10.1002/(SICI)1099-0526(199903/04)4:4⟨31::AID-CPLX5⟩3.0.CO;2-4
5. Aljarah, I., Mirjalili, S., Al-Madi, N.: Training radial basis function networks using biogeography-based optimizer. Neural Computing and Applications **29**(7), 529 – 553 (2018). DOI 10.1007/s00521-016-2559-2
6. Askarzadeh, A.: A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm. Computers and Structures **169**, 1 – 12 (2016). DOI 10.1016/j.compstruc.2016.03.001

7. Bäck, T., Rudolph, G., Schwefel, H.P.: Evolutionary programming and evolution strategies: Similarities and differences. In: In Proceedings of the Second Annual Conference on Evolutionary Programming, pp. 11–22 (1997)
8. Barshandeh, S., Piri, F., Sangani, S.R.: Hmpa: an innovative hybrid multi-population algorithm based on artificial ecosystem-based and harris hawks optimization algorithms for engineering problems. Engineering with Computers (2020). DOI 10.1007/s00366-020-01120-w
9. Cai, J., Zhou, R., Lei, D.: Dynamic shuffled frog-leaping algorithm for distributed hybrid flow shop scheduling with multiprocessor tasks. Engineering Applications of Artificial Intelligence **90**, 103540 (2020). DOI 10.1016/j.engappai.2020.103540
10. Chegini, S.N., Bagheri, A., Najafi, F.: PSOSCALF: A new hybrid pso based on sine cosine algorithm and levy flight for solving optimization problems. Applied Soft Computing **73**, 697 – 726 (2018). DOI 10.1016/j.asoc.2018.09.019
11. Coello-Coello, C.A.: Treating constraints as objectives for single-objective evolutionary optimization. Engineering Optimization **32**(3), 275–308 (2000). DOI 10.1080/03052150008941301
12. Coello-Coello, C.A., Mezura-Montes, E.: Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. Advanced Engineering Informatics **16**(3), 193 – 203 (2002). DOI 10.1016/S1474-0346(02)00011-3
13. Das, S., Bhattacharya, A., Chakraborty, A.K.: Solution of short-term hydrothermal scheduling using sine cosine algorithm. Soft Computing **22**, 6409 6427 (2018). DOI 10.1007/s00500-017-2695-3
14. Dimakopoulos, V.V., Hadjidoukas, P.E., Philos, G.C.: A microbenchmark study of openmp overheads under nested parallelism. In: R. Eigenmann, B.R. de Supinski (eds.) OpenMP in a New Era of Parallelism, pp. 1–12. Springer Berlin Heidelberg, Berlin, Heidelberg (2008). DOI 10.1007/978-3-540-79561-2_1
15. Dokeroglu, T., Sevinc, E., Cosar, A.: Artificial bee colony optimization for the quadratic assignment problem. Applied Soft Computing **76**, 595 – 606 (2019). DOI 10.1016/j.asoc.2019.01.001
16. Dorigo, M., Di Caro, G.: New ideas in optimization. chap. The Ant Colony Optimization Meta-heuristic, pp. 11–32. McGraw-Hill Ltd., UK, Maidenhead, UK, England (1999). URL `http://dl.acm.org/citation.cfm?id=329055.329062`
17. Eusuff, M., Lansey, K., Pasha, F.: Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization. Engineering Optimization **38**(2), 129–154 (2006). DOI 10.1080/03052150500384759
18. Farah, A., Belazi, A.: A novel chaotic Jaya algorithm for unconstrained numerical optimization. Nonlinear Dynamics **93**, 1451 – 1480 (2018). DOI 10.1007/s11071-018-4271-5
19. Free Software Foundation, Inc.: GCC, the gnu compiler collection. `https://www.gnu.org/software/gcc/index.html`
20. Gandomi, A., Yang, X.S., Talatahari, S., Alavi, A.: Firefly algorithm with chaos. Communications in Nonlinear Science and Numerical Simulation **18**(1), 89 – 98 (2013). DOI 10.1016/j.cnsns.2012.06.009
21. Gao, S., Vairappan, C., Wang, Y., Cao, Q., Tang, Z.: Gravitational search algorithm combined with chaos for unconstrained numerical optimization. Applied Mathematics and Computation **231**, 48 – 62 (2014). DOI 10.1016/j.amc.2013.12.175
22. Ghaffarzadeh, P., Nadimi, M.H., Nabiollahi, A.: KMGEM: Data clustering by combination of k-means and grenade explosion algorithm. International Journal of Computer Applications **147**, 21–29 (2016). DOI 10.5120/ijca2016911333
23. Gokhale, S., Kale, V.: An application of a tent map initiated chaotic firefly algorithm for optimal overcurrent relay coordination. International Journal of Electrical Power & Energy Systems **78**, 336 – 342 (2016). DOI 10.1016/j.ijepes.2015.11.087
24. Han, X., Yue, L., Dong, Y., Xu, Q., Xie, G., Xu, X.: Efficient hybrid algorithm based on moth search and fireworks algorithm for solving numerical and constrained engineering optimization problems. The Journal of Supercomputing **76 (12)**, 9404–9429 (2020). DOI 10.1007/s11227-020-03212-2
25. He, Q., Wang, L.: A hybrid particle swarm optimization with a feasibility-based rule for constrained optimization. Applied Mathematics and Computation **186**(2), 1407 – 1422 (2007). DOI 10.1016/j.amc.2006.07.134
26. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. MIT Press (1992)
27. Hong, W.C.: Traffic flow forecasting by seasonal SVR with chaotic simulated annealing algorithm. Neurocomputing **74**(12), 2096 – 2107 (2011). DOI 10.1016/j.neucom.2010.12.032

28. Jasmine, J., Annadurai, S.: Real time video image enhancement approach using particle swarm optimisation technique with adaptive cumulative distribution function based histogram equalization. Measurement **145**, 833 – 840 (2019). DOI 10.1016/j.measurement.2018.12.105
29. Jia, D., Zheng, G., Khan, M.K.: An effective memetic differential evolution algorithm based on chaotic local search. Information Sciences **181**(15), 3175 – 3187 (2011). DOI 10.1016/j.ins.2011.03.018
30. Karaboga, D., Basturk, B.: On the performance of artificial bee colony (ABC) algorithm. Appl. Soft Comput. **8**(1), 687–697 (2008). DOI 10.1016/j.asoc.2007.05.007
31. Koza, J.R.: Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Tech. rep., Stanford, CA, USA (1990)
32. Langari, R.K., Sardar, S., Mousavi], S.A.A., Radfar, R.: Combined fuzzy clustering and firefly algorithm for privacy preserving in social networks. Expert Systems with Applications **141**, 112968 (2020). DOI 10.1016/j.eswa.2019.112968
33. Liu, B., Ma, H., Zhang, X.: A co-evolutionary differential evolution algorithm for constrained optimization. In: Third International Conference on Natural Computation (ICNC 2007), vol. 4, pp. 51–57 (2007). DOI 10.1109/ICNC.2007.10
34. Liu, H., Cai, Z., Wang, Y.: Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization. Applied Soft Computing **10**(2), 629 – 640 (2010). DOI 10.1016/j.asoc.2009.08.031
35. Ma, H., Simon, D., Siarry, P., Yang, Z., Fei, M.: Biogeography-based optimization: A 10-year review. IEEE Transactions on Emerging Topics in Computational Intelligence **1**(5), 391–407 (2017). DOI 10.1109/TETCI.2017.2739124
36. Ma, Z.S.: Chaotic populations in genetic algorithms. Applied Soft Computing **12**(8), 2409 – 2424 (2012). DOI 10.1016/j.asoc.2012.03.001
37. Mezura-Montes, E., Coello-Coello, C.A., Velázquez-Reyes, J.: Increasing successful offspring and diversity in differential evolution for engineering design. In: Proc. Adaptive Computing in Design and Manufacture (ACDM 2006) (2006)
38. Migallón, H., Jimeno-Morenilla, A., Sánchez-Romero, J., Belazi, A.: Efficient parallel and fast convergence chaotic jaya algorithms. Swarm and Evolutionary Computation , 100698 (2020). DOI 10.1016/j.swevo.2020.100698
39. Migallón, H., Jimeno-Morenilla, A., Sánchez-Romero, J., Belazi, A.: Efficient parallel and fast convergence chaotic jaya algorithms. Swarm and Evolutionary Computation p. 100698 (2020). DOI 10.1016/j.swevo.2020.100698
40. Mingjun, J., Huanwen, T.: Application of chaos in simulated annealing. Chaos, Solitons & Fractals **21**(4), 933 – 941 (2004). DOI 10.1016/j.chaos.2003.12.032
41. Mirjalili, S.: SCA: A sine cosine algorithm for solving optimization problems. Knowledge-Based Systems **96**, 120 – 133 (2016). DOI 10.1016/j.knosys.2015.12.022
42. Muriel, J.B., Fotouhi, A.: Electric vehicle fleet management using ant colony optimisation. International Journal of Strategic Engineering **3**(1) (2020). DOI 10.4018/IJoSE.2020010101
43. Naserbegi, A., Aghaie, M., Minuchehr, A., Alahyarizadeh, G.: A novel exergy optimization of bushehr nuclear power plant by gravitational search algorithm (gsa). Energy **148**, 373 – 385 (2018). DOI 10.1016/j.energy.2018.01.119
44. OpenMP Architecture Review Board: OpenMP Application Program Interface, version 3.1. `http://www.openmp.org` (2011)
45. Öztürk, H.T., Dede, T., Türker, E.: Optimum design of reinforced concrete counterfort retaining walls using tlbo, jaya algorithm. Structures **25**, 285 – 296 (2020). DOI 10.1016/j.istruc.2020.03.020
46. Patle, B., Parhi, D., Jagadeesh, A., Kashyap, S.K.: Matrix-binary codes based genetic algorithm for path planning of mobile robot. Computers and Electrical Engineering **67**, 708 – 728 (2018). DOI 10.1016/j.compeleceng.2017.12.011
47. Peng, C., Sun, H., Guo, J., Liu, G.: Dynamic economic dispatch for wind-thermal power system using a novel bi-population chaotic differential evolution algorithm. International Journal of Electrical Power & Energy Systems **42**(1), 119 – 126 (2012). DOI 10.1016/j.ijepes.2012.03.012
48. Poli, R., Kennedy, J., Blackwell, T.: Particle swarm optimization. Swarm Intelligence **1**(1), 33–57 (2007). DOI 10.1007/s11721-007-0002-0
49. Price, K.V.: New ideas in optimization. chap. An Introduction to Differential Evolution, pp. 79–108. McGraw-Hill Ltd., UK, Maidenhead, UK, England (1999). URL `http://dl.acm.org/citation.cfm?id=329055.329069`

50. Rajeswara Rao, B., Tiwari, R.: Optimum design of rolling element bearings using genetic algorithms. Mechanism and Machine Theory **42**(2), 233 – 250 (2007). DOI 10.1016/j.mechmachtheory.2006.02.004
51. Rao, R.V.: Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. International Journal of Industrial Engineering Computations **7**, 19–34 (2016). DOI 10.5267/j.ijiec.2015.8.004
52. Rao, R.V., Savsani, V., Vakharia, D.: Teaching-learning-based optimization: A novel method for constrained mechanical design optimization problems. Computer-Aided Design **43**(3), 303–315 (2011). DOI 10.1016/j.cad.2010.12.015
53. Rao, R.V., Waghmare, G.: A new optimization algorithm for solving complex constrained design optimization problems. Engineering Optimization **49**(1), 60–83 (2017). DOI 10.1080/0305215X.2016.1164855
54. Rashedi, E., Nezamabadi-pour, H., Saryazdi, S.: GSA: A gravitational search algorithm. Information Sciences **179**(13), 2232 – 2248 (2009). DOI 10.1016/j.ins.2009.03.004. Special Section on High Order Fuzzy Sets
55. Reyes, O., Ventura, S.: Evolutionary strategy to perform batch-mode active learning on multi-label data. ACM Transactions on Intelligent Systems and Technology **9**(4) (2018). DOI 10.1145/3161606
56. Rezaee Jordehi, A.: A chaotic-based big bangbig crunch algorithm for solving global optimisation problems. Neural Computing and Applications **25**, 1329–1335 (2014). DOI 10.1007/s00521-014-1613-1
57. Sadollah, A., Bahreininejad, A., Eskandar, H., Hamdi, M.: Mine blast algorithm: A new population based algorithm for solving constrained engineering optimization problems. Applied Soft Computing **13**(5), 25922612 (2013). DOI 10.1016/j.asoc.2012.11.026
58. Saremi, J., Mirjalili, S., Lewisn, A.: Biogeography-based optimisation with chaos. Neural Computing and Applications **25**(5), 1077 – 1097 (2014). DOI 10.1007/s00521-014-1597-x
59. Schwefel, H.P.: Evolutionsstrategie und numerische Optimierung. Dr.-Ing. Thesis, Technical University of Berlin, Department of Process Engineering (1975)
60. Viegas, F., Rocha, L., Gonalves, M., Mouro, F., Sá, G., Salles, T., Andrade, G., Sandin, I.: A genetic programming approach for feature selection in highly dimensional skewed data. Neurocomputing **273**, 554 – 569 (2018). DOI 10.1016/j.neucom.2017.08.050
61. Wang, H., Hu, Z., Sun, Y., Su, Q., Xia, X.: A novel modified bsa inspired by species evolution rule and simulated annealing principle for constrained engineering optimization problems. Neural Computing and Applications **31**, 193 – 203 (2019). DOI 10.1007/s00521-017-3329-5
62. Wang, X., Duan, H.: A hybrid biogeography-based optimization algorithm for job shop scheduling problem. Computers & Industrial Engineering **73**, 96 – 114 (2014). DOI 10.1016/j.cie.2014.04.006
63. Wu, X., Che, A.: A memetic differential evolution algorithm for energy-efficient parallel machine scheduling. Omega **82**, 155 – 165 (2019). DOI 10.1016/j.omega.2018.01.001
64. Xin-She, Y.: Firefly algorithm, lévy flights and global optimization. Research and Development in Intelligent Systems XXVI , 209–218 (2009). DOI 10.1007/978-1-84882-983-1_15
65. Yan, X.F., Chen, D.Z., Hu, S.X.: Chaos-genetic algorithms for optimizing the operating conditions based on RBF-PLS model. Computers & Chemical Engineering **27**(10), 1393 – 1404 (2003). DOI 10.1016/S0098-1354(03)00074-7
66. Zhang, M., Luo, W., Wang, X.: Differential evolution with dynamic stochastic selection for constrained optimization. Information Sciences **178**(15), 3043 – 3074 (2008). DOI 10.1016/j.ins.2008.02.014
67. Zhao, W., Wang, L., Zhang, Z.: Supply-demand-based optimization: A novel economics-inspired algorithm for global optimization. IEEE Access **7**, 73182–73206 (2019). DOI 10.1109/ACCESS.2019.2918753