An Interval Space Reducing Method for Constrained Problems with Particle Swarm Optimization

T. M. Machado-Coelho, A. M. C. Machado, L. Jaulin, P. Ekel, W. Pedrycz, G. L. Soares

Abstract

In this paper, we propose a method for solving constrained optimization problems using Interval Analysis combined with Particle Swarm Optimization. A Set Inverter Via Interval Analysis algorithm is used to handle constraints in order to reduce constrained optimization to quasi unconstrained one. The algorithm is useful in the detection of empty search spaces, preventing useless executions of the optimization process. To improve computational efficiency, a Space Cleaning algorithm is used to remove solutions that are certainly not optimal. As a result, the search space becomes smaller at each step of the optimization procedure. After completing pre-processing, a modified Particle Swarm Optimization algorithm is applied to the reduced search space to find the global optimum. The efficiency of the proposed approach is demonstrated through comprehensive experimentation involving 100,000 runs on a set of well-known benchmark constrained engineering design problems. The computational efficiency of the new method is quantified by comparing its results with other PSO variants found in the literature.

Keywords: Interval Analysis, Evolutionary computation, Particle Swarm Optimization, Constrained Optimization

1. Introduction

There are many optimization problems for which it is very hard to find the best possible solution within a reasonable amount of time. One way of solving these problems is to use evolutionary algorithms, which are a class of computational intelligence methods guided by the use of stochastic heuristics [4].

Particle Swarm Optimization (PSO) is an evolutionary algorithm inspired by the movement of particles on a n-dimensional space. It works by generating many different possible solutions, searching for the best one and moving the values of the others in a manner similar to the movement of a flock of birds. It can be used to solve unconstrained as well as constrained optimization problems. However, the use of PSO for solving constrained problems may generate many invalid candidate solutions, negatively affecting the efficiency of the method.

In this paper, we present [**I**]PSO, a novel hybrid interval PSO method directed to the solution of constrained optimization problems. The method uses Interval Analysis [30] in order to eliminate the consideration of the invalid candidate solutions that can be generated by PSO. Instead of using conventional penalty functions [32], [**I**]PSO applies the Set Inverter Via Interval Analysis (SIVIA) algorithm [23, 34, 31], that allows for the removal of some invalid regions of the search space, so that PSO can be applied as if it were almost unconstrained. As SIVIA exhibits a high computational cost, a space cleaning algorithm was created to eliminate the intervals that do not contain the optimum, reducing the total number of function calls made by SIVIA.

[**I**]PSO was validated using a set of benchmark engineering design problems [29, 20, 1, 6, 7, 37, 22, 26, 5, 39] and comparing the results with other PSO variants found in the literature [1, 7, 5]. The experiments have shown that [**I**]PSO achieves better or similar results when compared to the ones obtained by other studies. The paper's main contribution and its originality lie in the formation of a new optimization environment bringing together interval computing and evolutionary techniques, in order to create an efficient constraint handling method for constrained optimization problems.

The paper is structured as follows. Section 2 presents background knowledge of Interval Analysis, Constrained Optimization and Particle Swarm Optimization. Section 3 presents the proposed method and Section 4 the metrics used to evaluate the experiments. Section 5 presents a simple illustrative example of the pre-processing step of the proposed method. Section 6 presents the experiments and results, followed by conclusion and future works. Finally, the Appendix presents the problems used in the experiments and the best results found for each one.

2. Background and Related Work

2.1. Constrained Optimization Problems

Let $\mathbf{x}$ be the design parameter and $\mathbf{X}$ the search space, $\mathbf{x} \in \mathbf{X} \subseteq \mathbb{R}^{n_x}$. Let also $\mathbf{g}(\mathbf{x}) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_g}$ be the vector of constraint functions. Then the feasible search space $\mathbf{X}'$ can be defined as

$$\mathbf{X}' = \{\mathbf{x} \in \mathbf{X} \mid \mathbf{g}(\mathbf{x}) \leq \mathbf{0}\}. \tag{1}$$

Alternatively, $\mathbf{X}' = g_1(\mathbf{x}) \cap g_2(\mathbf{x}) \cap \cdots \cap g_{n_g}(\mathbf{x})$. For instance, Fig.1 shows a search space $\mathbf{X}'$ considering the constraints $0 \leq x_1 \leq 40$, $0 \leq x_2 \leq 60$, $g_1(\mathbf{x}) = x_1 + x_2 - 70 \leq 0$, $g_2(\mathbf{x}) = -5x_1 - 2x_2 + 100 \leq 0$, $g_3(\mathbf{x}) = x_2 - x_1 - 40 \leq 0$ and $g_4(\mathbf{x}) = x_1 - x_2 \leq 0$. If $f(\mathbf{x}) : \mathbb{R}^{n_x} \to \mathbb{R}^n$ is the objective function, then an unconstrained minimization problem on $\mathbf{X}'$ leads to the solution in the form:

$$\mathbf{x}^* = \arg\min f(\mathbf{x}), \quad \mathbf{x}^* \in \mathbf{X}'. \tag{2}$$

The maximization problem can be described analogously. In this paper, only minimization problems are considered.

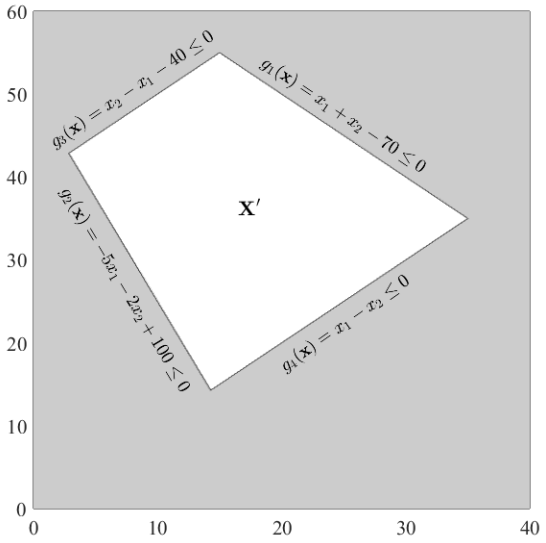


Figure 1: Example of search space $\mathbf{X} = [0, 40] \times [0, 60]$. The unconstrained space $\mathbf{X}'$ is shown in white.

2.2. Interval Analysis

Interval Analysis deals with interval processing. It uses branch-and-bound algorithms [23] to solve global optimization problems and was first used by Moore [30] to deal with rounding errors. The use of interval methods was further extended to solve problems with imprecise parameters and to handle different kinds of uncertainty [23]. In this work, interval analysis techniques were used to construct a wrapper over constraint functions, partitioning the search space into feasible, unfeasible, and indeterminate regions. This section addresses some important interval concepts needed to understand the handling of constraint functions considered in this paper.

Interval numbers (intervals), such as $[x]$, are sets consisting of all numbers delimited by a minimum $\underline{x}$ and a maximum $\overline{x}$ values. The domain of all interval numbers is denoted by $\mathbb{IR}$. Real intervals can be defined [34] as

$$[\underline{x}, \overline{x}] = \{x \in \mathbb{IR} | \underline{x} \leq x \leq \overline{x}\}\,. \tag{3}$$

An interval's width $w_{[x]}$ is calculated as:

$$w_{[x]} = (\overline{x} - \underline{x})\,. \tag{4}$$

If $f(x)$ is a $\mathbb{IR} \to \mathbb{IR}$ function, it's interval extension is defined as:

$$f([x]) \triangleq [\{f(x) | x \in [x]\}]\,. \tag{5}$$

Concatenated intervals or boxes $[\mathbf{x}] \in \mathbb{IR}^{n_x}$ are used to represent n_x-

dimensional environments, and can be defined as a Cartesian product:

$$[\underline{\mathbf{x}}, \overline{\mathbf{x}}] = [\underline{x}_1, \overline{x}_1] \times [\underline{x}_2, \overline{x}_2] \times \ldots \times \left[\underline{x}_{n_x}, \overline{x}_{n_x}\right]. \tag{6}$$

The interval function $f([\mathbf{x}])$ equivalent to its $f(\mathbf{x})$ counterpart can be obtained by replacing the $\mathbf{x}$ vector for $[\mathbf{x}]$, such that $\mathbf{x} \in [\mathbf{x}]$.

The box center $\mathbf{c}_{[\mathbf{x}]}$, width $w_{[\mathbf{x}]}$ and its hyper-volume $H_{[\mathbf{x}]}$ are defined as:

$$\mathbf{c}_{[\mathbf{x}]} = \left[\frac{(\underline{x}_1 + \overline{x}_1)}{2}, \ldots, \frac{(\underline{x}_i + \overline{x}_i)}{2}, \ldots, \frac{\left(\underline{x}_{n_x} + \overline{x}_{n_x}\right)}{2}\right], \tag{7}$$

$$w_{[\mathbf{x}]} = \max\left(w_{[x_i]}\right),\ i = 1 \ldots n_x, \tag{8}$$

$$H_{[\mathbf{x}]} = \prod_{i=1}^{n_x} w_{[x_i]}. \tag{9}$$

For instance, considering $[\mathbf{x}] = [0.5, 3] \times [2, 5]$, we have $\mathbf{c}_{[\mathbf{x}]} = (1.75, 3.5)$, $w_{[\mathbf{x}]} = 3$ and $H_{[\mathbf{x}]} = 2.5 \times 3 = 7.5$.

Considering the distance between two points $y \in \mathbb{R}$ and $x \in \mathbb{R}$, $d_{y,x} = |x - y|$, the distance between y and an interval $[x]$, $d_{y,[x]}$, can be defined as follows:

$$d_{y,[x]} = \begin{cases} 0, \text{ if } y \in [x], \\ \min\left(d_{y,\underline{x}}; d_{y,\overline{x}}\right). \end{cases} \tag{10}$$

Then, the distance between $\mathbf{y} \in \mathbb{R}^n$ and $[\mathbf{x}] \in \mathbb{IR}^n$, $d_{\mathbf{y},[\mathbf{x}]}$, is defined as

$$d_{\mathbf{y},[\mathbf{x}]} = \sqrt{\sum_{i=1}^{n} {d_{y_i,[\mathbf{x}_i]}}^2}. \tag{11}$$

For instance, Fig.2 shows an example of the distance between $y = (2.5, 2.5)$ and the boxes $[\mathbf{x}] = [0.5, 3] \times [2, 5]$, $[\mathbf{u}] = [5, 8] \times [3, 6]$, $[\mathbf{w}] = [2, 4] \times [4, 8.5]$, $[\mathbf{z}] = [1, 2] \times [0.5, 1]$. Then, $d_{\mathbf{y},[\mathbf{x}]} = 0$, $d_{\mathbf{y},[\mathbf{u}]} = 2.5495$, $d_{\mathbf{y},[\mathbf{w}]} = 1.5$ and $d_{\mathbf{y},[\mathbf{z}]} = 1.5811$.

A bisection is an operation that divides a box into two boxes of the same size, by dividing the box's largest interval. Formally, *bisect* $([\mathbf{x}])$ is

$$[\mathbf{x}]^1, [\mathbf{x}]^2 \leftarrow \mathit{bisect}\left([\mathbf{x}]\right),\ \mathit{where} \tag{12}$$

$$[\mathbf{x}]^1 = [x_1] \times \ldots \times \left[\underline{x}_j, \frac{\underline{x}_j + \overline{x}_j}{2}\right] \times \ldots \times [x_{n_x}],$$

$$[\mathbf{x}]^2 = [x_1] \times \ldots \times \left[\frac{\underline{x}_j + \overline{x}_j}{2}, \overline{x}_j\right] \times \ldots \times [x_{n_x}].$$

Index j is associated with the first component that presents maximum width, i.e., $j = \min\{i \mid w_{[x_i]} = w_{[\mathbf{x}]})\}$. For instance, Fig.3 shows an example of a box $[\mathbf{x}] \in \mathbb{IR}^2$ that was bisected twice: a) after bisection 1, $[\mathbf{x}] = [\mathbf{x}]^1 \cup [\mathbf{x}]^2$; b) after

bisection 2, $[\mathbf{x}] = [\mathbf{x}]^2 \cup [\mathbf{x}]^{11} \cup [\mathbf{x}]^{12}$. The Bisection is an important interval operation for reducing uncertainty in interval based methods.

As defined by Jaulin et al. [23], an interval function $[f]([\mathbf{x}])$ is an *inclusion function* of $f([\mathbf{x}])$ if:

$$\forall [\mathbf{x}], f([\mathbf{x}]) \subset [f]([\mathbf{x}]). \tag{13}$$

Considering $[\mathbf{x}]_k \subseteq [\mathbf{x}]_{k-1} \subseteq \cdots \subseteq [\mathbf{x}]_m \subseteq \cdots \subseteq [\mathbf{x}]_0$, two properties of the inclusion functions are of special interest here:

a) monotonicity,

$$[\mathbf{x}]_k \subseteq [\mathbf{x}]_m \Rightarrow [f]([\mathbf{x}]_k) \subseteq [f]([\mathbf{x}]_m); \tag{14}$$

b) convergence,

$$\lim_{k\to\infty} w_{[\mathbf{x}]_k} = 0 \Rightarrow \lim_{k\to\infty} w_{[f]([\mathbf{x}]_k)} = 0. \tag{15}$$

Fig.4 and Fig.5 show these properties of inclusion functions. As a result, monotonic and convergent inclusion functions guarantee reduced uncertainties when used together with bisections. For example, if $[\mathbf{x}] = [\mathbf{x}]^1 \cup [\mathbf{x}]^2$, then $[f]([\mathbf{x}]^1) \cup [f]([\mathbf{x}]^2) \subseteq [f]([\mathbf{x}])$.

The inclusion test $[t]([\mathbf{x}])$ is defined as:

$$[t]([\mathbf{x}]) = \begin{cases} 1, \ t(\mathbf{x}) = 1, \ \forall \mathbf{x} \in [\mathbf{x}], \\ 0, \ t(\mathbf{x}) = 0, \ \forall \mathbf{x} \in [\mathbf{x}], \\ [0,1], \ otherwise. \end{cases} \tag{16}$$

where $[0,1]$ stands for an *indeterminate conclusion* for an inclusion test. The comparison between two boxes $[\mathbf{x}]$ and $[\mathbf{y}]$ can be described using inclusion tests.

$$[t]([\mathbf{x}] \leq [\mathbf{y}]) = \begin{cases} 1, \ \overline{\mathbf{x}} \leq \underline{\mathbf{y}} \\ 0, \ \underline{\mathbf{x}} > \overline{\mathbf{y}} \\ [0,1], \ otherwise. \end{cases} \tag{17}$$

The comparison operators $\geq, <$ and $>$ are described analogously.

Finally, the comparison between a point $\mathbf{x} \in \mathbb{R}^{n_x}$ and a box $[\mathbf{y}] \in \mathbb{IR}^{n_x}$ is given by:

$$[t](\mathbf{x} \leq [\mathbf{y}]) = \begin{cases} 1, \ \mathbf{x} \leq \underline{\mathbf{y}} \\ 0, \ \mathbf{x} > \overline{\mathbf{y}} \\ [0,1], \ otherwise. \end{cases} \tag{18}$$

As an example, using the boxes and point shown in Fig.2, for Eq.16 and Eq.17, $[t]([\mathbf{x}] \leq [\mathbf{u}]) = [0,1]$, $[t]([\mathbf{z}] > [\mathbf{w}]) = 0$, $[t]([\mathbf{w}] \geq [\mathbf{x}]) = [0,1]$, $[t]([\mathbf{z}] \leq [\mathbf{u}]) = 1$. For Eq.18, $[t](y \leq [\mathbf{u}]) = 1$, $[t](y \leq [\mathbf{x}]) = [0,1]$, $[t](y \leq [\mathbf{w}]) = [0,1]$ and $[t](y \leq [\mathbf{z}]) = 0$.

Notice that if $[t]([\mathbf{z}] < [\mathbf{u}]) = 1$, then $\forall \mathbf{z} \in [\mathbf{z}]$, $\forall \mathbf{u} \in [\mathbf{u}]$, $\mathbf{z} < \mathbf{u}$. As a result, the method proposed in this work is able to remove $[\mathbf{u}]$ from the search space. Also, considering $[\mathbf{u}]$ and $[\mathbf{z}]$ as in Fig.2, if $[f]([\mathbf{x}]) = [\mathbf{z}]$ and $[f]([\mathbf{y}]) = [\mathbf{u}]$, then

Algorithm 1 Adapted SIVIA **algorithm**

Input: $V_{[\mathbf{X}]}$, $I_{[\mathbf{X}]}$, ε
Output: $V_{[\mathbf{X}]}$, $I_{[\mathbf{X}]}$

```
[x] ← I_[X].dequeue()
while w_[x] < ε  and  I_[X] ≠ ∅ do
    [t]([x]) ← [g]([x]) ⊆ [−∞, 0]
    if [t]([x]) = 1 then
        V_[X].enqueue([x])
    else if [t]([x]) = [0, 1] then
        I_[X].enqueue(bisect([x]))
    end if
    [x] ← I_[X].dequeue()
end while
I_[X].enqueue([x])
return V_[X], I_[X]
```

$\forall \mathbf{x} \in [\mathbf{x}]$, $\forall \mathbf{y} \in [\mathbf{y}]$, $f(\mathbf{x}) < f(\mathbf{y})$. In this case the method is able to remove $[\mathbf{y}]$ from the search space.

2.3. SIVIA

Set Inverter Via Interval Analysis (SIVIA) [23] is an algorithm that uses inclusion tests and inclusion functions to validate the set relationship between two boxes. In this work, SIVIA was applied to test which parts of the search space are compatible with the constraint functions. SIVIA's inputs are an initial box $[\mathbf{X}] \subseteq \mathbb{R}^n$ that contains the search space $\mathbf{X}$ and an accuracy parameter ε that defines a stopping criterion. SIVIA's outputs form a queue with valid boxes $V_{[\mathbf{X}]}$ and a queue with indeterminate boxes $I_{[\mathbf{X}]}$. Boxes that are guaranteed to be invalid are deleted.

SIVIA works by initially obtaining $[\mathbf{X}]$ and adding it to $I_{[\mathbf{X}]}$. An inclusion test is made to verify if the constraint functions satisfy the condition $[\mathbf{g}]([\mathbf{x}]) \subseteq [-\infty, \mathbf{0}]$. This is equivalent to testing whether $\mathbf{g}([\mathbf{x}]) \leq \mathbf{0}$, $\forall\ \mathbf{x} \in [\mathbf{x}]$. If the condition is satisfied, $[\mathbf{x}]$ is added to $V_{[\mathbf{X}]}$, otherwise $[\mathbf{x}]$ is deleted. Finally, if the output is [0,1], $[\mathbf{x}]$ is bisected yielding $[\mathbf{x}]^1$ and $[\mathbf{x}]^2$, that are added to $I_{[\mathbf{X}]}$. This is repeated until $I_{[\mathbf{X}]}$ is empty or $w_{[\mathbf{x}]}$ is lower than ε, returning $V_{[\mathbf{X}]}$ and $I_{[\mathbf{X}]}$. Although SIVIA originally uses $[\mathbf{X}]$ as its input, in this work the inputs are $V_{[\mathbf{X}]}$ and $I_{[\mathbf{X}]}$, as shown in Alg. 1.

SIVIA can be used as a deterministic branch-and-bound algorithm for global optimization [17], with some minor modifications. It is able to always find the global optimum, if it exists, for any problem in which $[f]([\mathbf{x}])$ and $[\mathbf{g}]([\mathbf{x}])$ can be written as inclusion functions. It is also able to determine if a problem has no feasible solution. However, it has a very high computational cost when used alone. To overcome this problem, a hybrid method will be presented in Section 3.

2.4. Particle Swarm Optimization

PSO was proposed by Kennedy and Eberhart [24], with the purpose of simulating the movement of a flock of birds. After some improvements of the original idea, the authors concluded that PSO was very efficient for optimizing unconstrained non-linear functions [25]. In PSO, a set of discrete [38] or continuous variables is defined as a particle and a group of particles forms a swarm. In the first step, the PSO algorithm generates a random position ($\mathbf{s}$) and velocity ($\mathbf{v}$), for each particle, and uses an objective function to evaluate a fitness value for every particle. The values of $\mathbf{s}$ and $\mathbf{v}$ evolve during each iteration. PSO saves the particles with the best individual fitness in a **pbest** (particle best) swarm, and saves the best particle as a **gbest** (global best) particle.

In order to determine the updated values of a particle, PSO first determines a weight h_i for each iteration i which is used to calculate the velocities of the particles and their new positions in the space. After this, the fitness of each particle is computed and the value of **pbest** re-evaluated. When all particles have been inspected, the particle with the best fitness is chosen as the new **gbest**. The process is repeated until a pre-defined number of iterations has been reached.

The computation of the velocity and position of a particle j at iteration i is ruled by the following set of assignments:

$$h_i \leftarrow h_{max} - i\left(h_{max} - h_{min}\right)/it_{max}, \tag{19}$$

$$\begin{aligned}\mathbf{v}_i^j \leftarrow h_i\Big[\mathbf{v}_{i-1}^j + c_1\mathbf{rand}() \odot \Big(\mathbf{pbest}^j - \mathbf{s}_{i-1}^j\Big) + \\ + c_2\mathbf{rand}() \odot \Big(\mathbf{gbest} - \mathbf{s}_{i-1}^j\Big)\Big],\end{aligned} \tag{20}$$

$$\mathbf{s}_i^j \leftarrow \mathbf{s}_{i-1}^j + \mathbf{v}_i^j, \tag{21}$$

where h_{min}, h_{max}, it_{max}, c_1, c_2 and s_{size} are user-defined parameters. In particular, h_{min} and h_{max} are the minimum and maximum of the possible weights, it_{max} is the maximum number of iterations, and $\mathbf{s}_i^j$ is the j-th particle in a swarm of size s_{size}, at iteration i. The parameters c_1 and c_2 are constants that make the particles move preferentially in the direction of either **gbest** or **pbest**. Function **rand**() returns a n-dimensional vector of random numbers from 0 to 1 and the $\odot$ operator denotes the Hadamard element-wise product between two vectors. The maximum number of iterations for each problem is it_{max}.

2.5. Constrained Optimization using PSO

Although PSO was designed to deal with unconstrained optimization, it can be modified in order to solve constrained problems as well. For example, it is possible to adapt PSO by using penalty functions [32, 14, 8]. Penalty functions can be classified as being *stationary* or *non-stationary* [33]. Stationary functions take the fitness value of an unfeasible solution found in an iteration and multiply

this value by a constant penalty factor, depending on the degree of violation of the constraints. Non-stationary penalties multiply the fitness by a factor that changes depending on the iteration number. The new fitness values are then used in the original PSO algorithm.

Constraints can also be handled without penalty functions. One way to do it is by removing invalid particles and replacing them by new ones, when generating particles in the first iteration. The process is repeated until a valid particle has been generated for every member of the population. For subsequent iterations, only **pbest** and **gbest** are updated when feasible solutions are found [20]. The main disadvantage of this method is that, depending on the problem, it may take a long time to generate new particles[25]. Some other strategies to handle constraints include resetting the particles to their former **pbest** values whenever they represent an unfeasible solution, as used by El-Gallad et al. in [12], or moving them back to their previous positions, as used by He et al. in [18]. These strategies, however, may force PSO to converge towards a local optimum [25]. A way to lower the likelihood of reaching local optima is by increasing the diversity of the swarm, changing the standard behavior of the particle [40, 36, 16, 27]. As an example, a mutation operator similar to the one used in genetic algorithms may be included at each iteration [19, 3, 21, 27]. Mazhoud et al.[28] present a constraint handling mechanism that consists of a closeness evaluation of the solutions to the feasible region, and uses Interval Arithmetic to normalize the total evaluations.

Another way to handle constraints, as proposed in this work, is to exclude the constrained space and force the optimizing tool to search for answers that lie exclusively within the feasible portion of the search space. This can be achieved using Interval Analysis, as shown by Soares in [34]. Solau et al. [35] use a method similar to the one discussed in this paper, although presenting a high computational cost when used with complex expressions, such as the ones found in some engineering problems.

Based on this rationale, we propose some changes to the methods in order to improve constraint handling in optimization by PSO.

3. Proposed Method - [I]PSO

[**I**]PSO is a hybrid method that combines Interval Analysis and PSO. Fig.6 shows the outline of the proposed method with the pre-processing block (see Alg. 2) composed by SIVIA and the Space Cleaning algorithm.

The SIVIA algorithm (see Alg. 1) takes $V_{[\mathbf{X}]}$ and $I_{[\mathbf{X}]}$ and bisects their elements in many smaller boxes $[\mathbf{x}]$ until a required precision ε_{min} is reached:

$$\varepsilon_{min} = \frac{1}{2^{\ell}} w_{[\mathbf{X}]}, \tag{22}$$

where ℓ is an integer effort parameter. The size of the resulting queues are then reduced, using Alg. 3, by finding a valid point $\mathbf{x}$ for each box in $V_{[\mathbf{X}]}$ and $I_{[\mathbf{X}]}$, using Alg. 4, and by removing the boxes for which $f([\mathbf{x}]) > f(\mathbf{x})$. Note that, due

to the property shown by Eq.18, Interval Analysis guarantees that the optimal region will never be removed from the search space. Once the pre-processing is completed, the boxes in $I_{[\mathbf{X}]}$ are joined with $V_{[\mathbf{X}]}$, generating a "clean" space $\mathbf{X}''$:

$$\mathbf{X}'' \triangleq V_{[\mathbf{X}]} \cup I_{[\mathbf{X}]}. \tag{23}$$

Therefore, from Eq.22 and Eq.23 it can be concluded that

$$\ell \to \infty \Rightarrow \mathbf{X}'' \to \mathbf{x}^*. \tag{24}$$

A high ℓ value, however, implies a high computational cost. The value of ℓ is thus limited and the adapted PSO is used in order to find $\mathbf{x}^*$, as shown in Alg. 5 and Alg. 6.

A common question when optimization algorithms are proposed is the computational complexity (also called computational cost or effort, CE) [9]. Usually, the CE of swarm methods like PSO is measured using the number of objective function calls. However, [I]PSO is a hybrid method and the overall CE is composed of the sum of objective function calls made by the pre-processing and PSO parts.

Eq.25 and 26 show the CE of the pre-processing in the worst case scenario, that is, when no boxes are removed from the search space. CE_f is measured in terms of objective function calls, with n_x being the dimensionality of the search space. CE_g measures the computational effort in terms of constraint function calls, with n_g being the number of constraint functions. Usually, the high computational cost is a drawback of Interval Analysis, and the Space Cleaning algorithm aims to rectify it.

$$CE_f = 2 \times \sum_{i=0}^{\ell} 2^{n_x \times i}. \tag{25}$$

$$CE_g = 2n_g \times \sum_{i=0}^{\ell} 2^{n_x \times i}. \tag{26}$$

After pre-processing, PSO (Alg. 6) is applied to the clean search space $\mathbf{X}''$. Fitness function calls are made once for each particle in the swarm per iteration, so the computational cost of PSO can be considered O($it_{max} * s_{size}$).

4. Metrics for optimization methods

Different performance metrics have been proposed for comparing evolutionary algorithms. In this paper, the following metrics were used: (a) the *remaining percentual hyper-volume* $H_{\%}$ of $\mathbf{X}$; (b) the obtained *best fitness* (f^*); and (c) the obtained *average of the best fitnesses* ($\overline{f^*}$). Note that metric (a) was developed for describing the method's performance. Metrics (b) and (c) are usually used to compare the performance of Evolutionary Algorithms [1, 2, 6, 7, 11, 20, 29, 26].

Algorithm 2 Pre-processing

Input: $\mathbf{X}$
Parameters: ε_{min}
Output: $\mathbf{X}^{''}$

1: $\varepsilon \leftarrow \frac{1}{2} w_{\mathbf{X}}$
2: Add $\mathbf{X}$ to $I_{[\mathbf{X}]}$
3: **while** $\varepsilon > \varepsilon_{min}$ **do**
4: $\quad V_{[\mathbf{X}]}, I_{[\mathbf{X}]} \leftarrow SIVIA\left(V_{[\mathbf{X}]}, I_{[\mathbf{X}]}, \varepsilon\right)$
5: $\quad V_{[\mathbf{X}]}, I_{[\mathbf{X}]} \leftarrow spaceCleaning\left(V_{[\mathbf{X}]}, I_{[\mathbf{X}]}\right)$
6: $\quad \varepsilon \leftarrow \frac{1}{2}\varepsilon$
7: **end while**
8: $\mathbf{X}^{''} \leftarrow V_{[\mathbf{X}]} \cup I_{[\mathbf{X}]}$
9: **return** $\mathbf{X}^{''}$

Algorithm 3 Space Cleaning

Input: $V_{[\mathbf{X}]}, I_{[\mathbf{X}]}$
Output: $V_{[\mathbf{X}]}, I_{[\mathbf{X}]}$

1: **for all** $[\mathbf{x}] \in V_{[\mathbf{X}]} \cup I_{[\mathbf{X}]}$ **calculate** $f([\mathbf{x}])$
2: **for all** $[\mathbf{x}] \in V_{[\mathbf{X}]} \cup I_{[\mathbf{X}]}$ **do**
3: $\quad$ Search for a $\mathbf{x} \in [\mathbf{x}], \mathbf{x} \in \mathbf{X}'$ (Alg. 4)
4: $\quad$ **if** $\mathbf{x}$ is found **then**
5: $\quad\quad$ Calculate $f(\mathbf{x})$
6: $\quad\quad$ Remove all $[\mathbf{x}]$ from $V_{[\mathbf{X}]} \cup I_{[\mathbf{X}]}$, $f(\mathbf{x}) < f([\mathbf{x}])$
7: $\quad$ **end if**
8: **end for**
9: **return** $V_{[\mathbf{X}]}, I_{[\mathbf{X}]}$

$H_{\%}$ shows the relationship between $\mathbf{X}^{''}$ and $\mathbf{X}$'s hyper-volume. It can be computed as

$$H_{\%} = \frac{H_{\mathbf{X}''}}{H_{\mathbf{X}}} \times 100, \tag{27}$$

where $H_{\mathbf{X}''}$ is the sum of the hyper-volumes of all its boxes

$$H_{\mathbf{X}''} = \sum H_{[\mathbf{x}]},\ \forall [\mathbf{x}] \in \mathbf{X}^{''}, \tag{28}$$

and $H_{\mathbf{X}}$ is the hyper-volume of $[\mathbf{X}]$ (See Eq.9).

The term f^* measures **gbest** with respect to its closeness to the objective function's fitness $f(\mathbf{x}^*)$ using the benchmark global optimum $\mathbf{x}^*$. Since this work only uses minimization, f^* is achieved when

$$f^* \leq (1+\epsilon) f(\mathbf{x}^*), \tag{29}$$

where ϵ is the required precision. Similarly, a sub optimum fitness f_s^* is reached

Algorithm 4 Interval local search algorithm

Input: $[\mathbf{x}]$
Parameters: w_{min}
Output: $\mathbf{x}$

1: Compute $\mathbf{c}_{[\mathbf{x}]}$ and $w_{[\mathbf{x}]}$
2: **if** $\mathbf{g}(\mathbf{c}_{[\mathbf{x}]}) \leq \mathbf{0}$ **or** $w_{[\mathbf{x}]} < w_{min}$ **then**
3: $\quad \mathbf{x} \leftarrow \mathbf{c}_{[\mathbf{x}]}$
4: $\quad$ **return** $\mathbf{x}$
5: **end if**
6: **if** $\sum_{j=1}^{n_g} g_j(\underline{\mathbf{x}}) \leq \sum_{j=1}^{n_g} g_j(\overline{\mathbf{x}})$ **then**
7: $\quad [\mathbf{x}] \leftarrow [\underline{\mathbf{x}}, \mathbf{c}_{[\mathbf{x}]}]$
8: **else if** $\sum_{j=1}^{n_g} g_j(\underline{\mathbf{x}}) > \sum_{j=1}^{n_g} g_j(\overline{\mathbf{x}})$ **then**
9: $\quad [\mathbf{x}] \leftarrow [\mathbf{c}_{[\mathbf{x}]}, \overline{\mathbf{x}}]$
10: **end if**
11: **go to step 1**

Algorithm 5 Generate New Particle

Input: $\mathbf{X}''$
Output: $\mathbf{x}$

1: $[\mathbf{x}] \leftarrow random\ box\ from\ \mathbf{X}''$
2: $\mathbf{x} \leftarrow random\ particle\ from\ [\mathbf{x}]$
3: **while** $[t]([\mathbf{x}]) = [0, 1]$ and $\mathbf{g}(\underline{\mathbf{x}}) > 0$ **do**
4: $\quad [\mathbf{x}] \leftarrow random\ box\ from\ \mathbf{X}''$
5: $\quad \mathbf{x} \leftarrow random\ particle\ from\ [\mathbf{x}]$
6: **end while**
7: **return** $\mathbf{x}$

when

$$f^* < f_s^* \leq (1 + k\epsilon) f(\mathbf{x}^*), \tag{30}$$

where k is a constant. The values of both ϵ and k are defined by the user.
Remark: $f(\mathbf{x}^*)$ is the minimum possible value. However, considering "build limitations" and possible rounding errors, our metrics consider f^* and f_s^* valid solutions.

Considering $f_i(\mathbf{gbest})$ being the best result found in simulation i, the mean result found for all simulations is $\overline{f^*}$ and can be written as

$$\overline{f^*} = \frac{1}{n_{run}} \sum_{i=1}^{n_{run}} f_i(\mathbf{gbest}). \tag{31}$$

Algorithm 6 Particle Swarm Optimization

Input: $\mathbf{X}''$
Parameters: c_1, c_2, s_{size}, h_{max}, h_{min}, p_m, it_{max}
Output: $\mathbf{x}^*$

for j from 1 to s_{size} **do**
  $\mathbf{s}_0^j \leftarrow$ generateNewParticle($\mathbf{X}''$) (Alg. 5)
  update **pbest** and **gbest**
end for
for i from 1 to it_{max} **do**
  for j from 1 to s_{size} **do**
    if $rand < p_m$ **then**
      $\mathbf{s}_i^j \leftarrow$ generateNewParticle($\mathbf{X}''$) (Alg. 5)
    else
      $\mathbf{s}_i^j \leftarrow$ new position (Eq. 19, 20 and 21)
    end if
    if $\mathbf{s}_i^j \notin \mathbf{X}''$ **then**
      $[\mathbf{x}] \leftarrow \arg\min_{[\mathbf{x}] \in \mathbf{X}''} \left(d_{\mathbf{s}_i^j,\, [\mathbf{x}]} \right)$
      $\mathbf{s}_i^j \leftarrow \arg\min_{\mathbf{x} \in [\mathbf{x}]} \left(d_{\mathbf{x},\, \mathbf{s}_i^j} \right)$
    end if
    update **pbest** and **gbest**
  end for
end for
return $\mathbf{x}^*$

5. An illustrative example

This section presents a simple example to explain how the pre-processing step of [**I**]PSO works, using the following function:

$$
\begin{aligned}
&Minimize: \\
&\quad f(\mathbf{x}) = 3(1-x_1)^2 \exp\left(-x_1^2 - (x_2+1)^2\right) \ \ldots \\
&\quad -10\left(\tfrac{x_1}{5} - x_1^3 - x_2^5\right) \exp\left(-x_1^2 - x_2^2\right) \ \ldots \\
&\quad -\tfrac{1}{3} \exp\left(-(x_1+1)^2 - x_2^2\right); \\
&subject\ to: \\
&\quad g_1(\mathbf{x}) = x_1^2 + x_2^2 \geq 1; \\
&\quad g_2(\mathbf{x}) = x_1^2 + x_2^2 \leq 4; \\
&where: \\
&\quad -\mathbf{3} \leq \mathbf{x} \leq \mathbf{3};
\end{aligned}
$$

According to subsection 2.1, $\mathbf{X} = \{[-3\ 3], [-3\ 3]\}$, and $\mathbf{X'} = g_1(\mathbf{X}) \cap g_2(\mathbf{X})$. Fig.7 shows various instances of the example using contour lines, with $f(\mathbf{x}) \times \mathbf{X}$

and $f(\mathbf{x}) \times \mathbf{X}'$ shown in Fig.7b and Fig.7a, respectively. The results presented in Fig.7b can be obtained using only SIVIA, with ε_{min}=0.1875 (Eq.22, with $\ell = 5$). In this case, SIVIA generates 264 blank boxes representing the valid search space, 416 black boxes with the indeterminate valid/invalid search space, with $w_{[x]} < \varepsilon_{min}$, and 284 light gray boxes of invalid search space. Fig. 7c to 7h show each step of SIVIA with the space cleaning algorithm. In this case, it generates 18 blank boxes (valid) that are stored in $V_{[\mathbf{X}]}$, 18 black boxes (indeterminate) stored in $I_{[\mathbf{X}]}$, and 29 light gray boxes (invalid). A total of 134 dark gray boxes was excluded because the space cleaning algorithm found a point (shown as a red spot) that presents a value better than any other possible values inside the box. Since it is impossible to know if the optimum is on $V_{[\mathbf{X}]}$ or $I_{[\mathbf{X}]}$, they were joined in the clean space $\mathbf{X}''$ (see Eq.23). [I]PSO then jumps through the boxes stored in $\mathbf{X}''$, shown in Fig.7h, searching for $\mathbf{x}^*$.

6. Experiments

In order to test [**I**]PSO, an interval arithmetic library was implemented in Java 8. The experiments consisted of: (a) solving benchmark optimization problems and comparing the results with the ones achieved by CiP-PSO [7], COPSO [1], CVI-PSO[28] and IAPSO [5]; (b) verifying the number of runs that were able to achieve global optimum or a value close to it; and (c) verifying the amount of unfeasible space that was removed by [**I**]PSO for each problem. [**I**]PSO was not compared to Solau et al.'s PSO [35] because the later did not present a comparable number of function evaluations. In this case, execution time would not be a proper metric for comparison due to differences in the implementation and the hardware used in the experiments. It should also be noted that there are many ways to improve PSO's performance that can be combined with [**I**]PSO. However, we did not demonstrate them in order to highlight the improvements made by [**I**]PSO.

Six benchmark problems were implemented in the experiments: the Pressure Vessel (PV), Welded Beam (WB), Speed Reducer (SR), Compression String (CS), Speed Reducer 2 (SR2) and Clutch Brake (CB) problems [2, 5, 7]. The objective functions and constraints for these problems can be found in the appendices. In order to show an accurate representation of [**I**]PSO's results, each experiment was executed 100,000 times using random seeds, with a total of 20,000 objective function calls per run. The value of 20,000 calls was chosen to allow for comparison with the previously published results. The parameters for [**I**]PSO were chosen based on the characteristics of each problem and are shown in Tab.1. Note that, due to the hybrid nature of the method, the maximum number of iterations was calculated based on the desired number of function calls, the number of function calls made in the pre-processing (shown in Tab.2) and the swarm size, as shown by Eq.32, so that the experiments could have a fair number of objective function calls when compared with other algorithms. Also note that $n_{f(\mathbf{x})}$ is the computational effort for the pre-processing of each

Table 1: [**I**]PSO parameters used in the experiments for the Pressure Vessel (PV), Welded Beam (WB), Speed Reducer (SR), Compression String (CS), Speed Reducer 2 (SR2) and Clutch Brake (CB) problems.

Parameters	SR	PV	WB	CS	SR2	CB
h_{max}	1	1	1	1	1	1
h_{min}	0.3	0.3	0.3	0.3	0.3	0.3
s_{size}	20	20	20	20	20	20
c_1;c_2	3;1	3;1	3;1	3;1	3;1	3;1
p_m	0.015	0.015	0.015	0.015	0.015	0.015
ℓ	3	6	6	4	3	3
k	10	10	10	10	10	10
ϵ	0.001	0.001	0.001	0.001	0.001	0.001
w_{min}	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-6}
it_{max}	896	744	791	992	952	530

problem, as shown in Tab.2.

$$it_{max} = (20000 - n_{f(\mathbf{x})})/s_{size}, \tag{32}$$

6.1. Space Cleaning Results

Tab.2 shows the results of the pre-processing, according to the metrics shown in Section 4, and the total number of objective function calls $n_{f(\mathbf{x})}$, which represent the computational cost of the method for each problem. The ε_{min} precision values used for each problem, which were chosen based on the best results found after several experiments, are shown in Tab.1. It can be seen that, for the WB, PV, SR and SR2 problems, $H_{\%}$ is lower than 1%, while for the CS problem it is close to 2% and for the CB problem, close to 3%. This shows that the original domain of the problems can be greatly reduced. The reduction of the search space can be seen in Fig.8

6.2. PSO Results

Tables 3-8 show a comparison of the statistical results obtained for each problem, with 30 runs for each. It can be seen that the results obtained by [**I**]PSO are similar to the ones recently found in the literature, with the main significant difference being the mean result for the PV problem. It should also be noted that 30 runs are not enough to properly show the results of the algorithm, and Tab.10 shows the statistical results obtained for 100,000 runs.

Table 2: Results of the pre-processing, with the original hyper-volume of the search space $H_{\mathbf{X}}$, resulting hyper-volume after using SIVIA $H_{\mathbf{X}''}$, the percentage of the remaining hyper-volume ($H_{\%}$), and the total number of number of objective function calls made in the pre-processing $n_{f(\mathbf{x})}$) for each problem.

	$H_{\mathbf{X}}$	$H_{\mathbf{X}''}$	$H_{\%}$	$n_{f(\mathbf{x})}$
SR	0.275	6.14E-04	0.223302841	2069
PV	1354314.063	32.28936344	0.002384186	5300
WB	353.8161	0.026530066	0.007498264	4410
CS	26.6175	0.591355591	2.221679687	148
SR2	0.55	0.00318	0.579833984	954
CB	2560000	82343.75	3.216552734	9388

Table 3: Statistical results for the WB problem, showing best, mean and worst experimental results for 30 runs, together with the standard deviation.

Algorithm	Best	Mean	Worst	Std.Dev
CiP-PSO[7]	1.724852	2.0574	NA	0.2154
COPSO[1]	1.724852	1.7248	NA	1.2E-05
CVI-PSO[28]	1.724852	1.725124	1.727665	6.12E-04
IAPSO[5]	1.724852	1.7248528	1.7248624	2.02E-06
[**I**]PSO	1.724852	1.7248787	1.7250002	3.28E-05

Tab.9 shows the number of global optima, sub-optima and failures achieved by [**I**]PSO, for each optimization problem. The optimality of the results was characterized based on Eq.29 and Eq.30. Therefore, in order to be considered optimal, the achieved solution should not differ more than 0.1% from the best found result. Sub-optimum solutions should differ less than 1%. All other solutions were considered failures. The average value found by [**I**]PSO was optimal according to these criteria, for all problems. The number of times the global optimum or sub-optimum values were not achieved is negligible.

Reducing the search space by an arbitrary amount does not guarantee a better mean value. As an example, Fig.9 shows the $\overline{f^*}$ value found by [**I**]PSO for the PV problem, while Fig.10 shows $H_{\%}$, for different $n_{f(\mathbf{x})}$ and ℓ values, with $\ell = 0$ meaning the pre-processing step is skipped. Note that for low ℓ values, [**I**]PSO did not get acceptable results. High ℓ values, however, imply in a high computational cost, which is also unacceptable. Because of this, it is necessary to find the ℓ value with the best possible cost-benefit, so that both the computational cost and mean result are acceptable. It should be noted that choosing the ℓ value depends on the precision degree required, and changes for each problem. In our experiments, the ℓ value used for each was decided after multiple tests so that it could offer the best possible results with the lowest

Table 4: Statistical results for the PV problem, showing best, mean and worst experimental results for 30 runs, together with the standard deviation.

Algorithm	Best	Mean	Worst	Std.Dev
CiP-PSO[7]	6059.7143	6092.0498	NA	12.1725
COPSO[1]	6059.7143	6071.0133	NA	15.1011
IAPSO[5]	6059.7143	6068.7539	6090.5314	14.0057
[**I**]PSO	6059.7143	6059.7155	6059.7257	0.00232

Table 5: Statistical results for the SR problem, showing best, mean and worst experimental results for 30 runs, together with the standard deviation.

Algorithm	Best	Mean	Worst	Std.Dev
CiP-PSO[7]	2996.348165	2996.3482	NA	0.0000
COPSO[1]	2996.372448	2996.4085	NA	0.0286
IAPSO[5]	2996.34816497	2996.3481	2996.34816497	6.88E-13
[**I**]PSO	2996.34816497	2996.3481	2996.34816509	2.43E-08

possible number of function evaluation calls.

Remark: It is important to note that the total number of function evaluation calls used in the experiments was limited to 20,000, regardless of the ℓ value, so the algorithm has roughly the same computational cost for all ℓ values.

These results validate [**I**]PSO, showing that it is able to find the global optimum in most of the runs and that the reduction of the search space helps the global optimization process.

7. Conclusions

A novel Constrained Optimization method, [**I**]PSO, was presented and validated on a set of benchmark engineering problems. [**I**]PSO has some advantages that must be highlighted. 1. [**I**]PSO uses Interval Analysis to substantially reduce the search space and thereby the efforts needed to search for the optimum

Table 6: Statistical results for the CS problem, showing best, mean and worst experimental results for 30 runs, together with the standard deviation.

Algorithm	Best	Mean	Worst	Std.Dev
CiP-PSO[7]	0.012665	0.0131	NA	4.1E-04
COPSO[1]	0.012665	0.0126	NA	1.2E-06
CVI-PSO[28]	0.0126655	0.0127310	0.0128426	5.58E-05
IAPSO[5]	0.01266523	0.013676527	0.01782864	1.573E-03
[**I**]PSO	0.01266567	0.012671486	0.01268312	3.89E-06

Table 7: Statistical results for the SR2 problem, showing best, mean and worst experimental results for 30 runs, together with the standard deviation.

Algorithm	Best	Mean	Worst	Std.Dev
IAPSO[5]	2994.471066	2994.47106	2994.4710	2.65E-09
[**I**]PSO	2994.471067	2994.47108	2994.4711	9.27E-06

Table 8: Statistical results for the CB problem, showing best, mean and worst experimental results for 30 runs, together with the standard deviation.

Algorithm	Best	Mean	Worst	Std.Dev
IAPSO[5]	0.313656	0.313656	0.313656	1.13E-16
[**I**]PSO	0.313656	0.313656	0.313656	0

are reduced, increasing the probability of finding the global optimum or a suboptimal solution. This can be seen in Fig.10; 2. [**I**]PSO stops pursuing a solution when the valid search space is empty; usually, when this happens, stochastic algorithms that use standard constraint handling strategies may return invalid solutions or enter infinite loops. 3. Although the pre-processing step is used together with PSO, it is flexible enough to be used in combination with other optimization techniques; 4. Finally, since the pre-processing is deterministic, it only needs to be executed once for each problem.

[**I**]PSO was validated using four benchmark engineering problems whose objective and constraint functions were formulated as interval inclusion functions. Each problem was run 100,000 times to ensure more precise evaluation. The number of objective function calls made by [**I**]PSO was also limited in order to allow for comparisons with other methods found in the literature. Tab.9 highlights the effectiveness of [**I**]PSO, showing that it can be considered a good option in the solution of problems with similar characteristics. If the number of function calls is not a limiting issue, [**I**]PSO is always able to find the global optimum. It should also be noted that, although we limited the number of func-

Table 9: Number of experiments in which [**I**]PSO achieved global optimum, sub-optimum or failed to converge.

Probl.	f^*	f_s^*	failed
SR	99,772	200	28
PV	98,693	1,307	0
WB	99,453	546	1
CS	81,978	18,022	0
SR2	96,161	1,145	2,694
CB	100,000	0	0

Table 10: Statistical results for 100,000 runs, showing best, mean and worst experimental results, together with the standard deviation.

Algorithm	Best	Mean	Worst	Std.Dev
WB	1.7248523273365091	1.7249294530346946	1.7391705694087012	0.00030214148342926588
PV	6059.7143350503729	6059.966809804203	6091.1301522511558	1.8472132251313591
CS	0.012665232841936448	0.012672868998827698	0.012723627431697459	5.5266747835980769E-06
SR	2996.3481649685305	2996.3737892120294	3035.625578659377	0.69778131505116459
SR2	2994.4710663190704	2995.6624564145163	3046.7136848521786	6.4699844474780095
CB	0.313656	0.313656	0.313656	1.13E-16

tion calls in our experiments to 20000, most design problems are not real-time applications, so the number of function calls does not need to be limited, and the ℓ precision parameter can be increased as much as the user needs.

[**I**]PSO has some drawbacks. First, the pre-processing has a high computational cost due of the number of bisections it performs for each variable and due to its precision level, given by ε_{min}. In other words, the computational cost increases as the number of dimensions increases, or ε_{min} decreases. Second, when it is used with problems with a small restricted space, it acts as an ordinary PSO, but may waste time in the pre-processing. Finally, since [**I**]PSO uses Interval Analysis, it can only be applied with problems that can be modeled with interval inclusion functions.

8. Future work

[**I**]PSO requires analytically expressed interval functions in order to be used. There are many problems that have no analytical description. Besides, it may take some effort to reduce a regular mathematical model to its interval counterpart. Therefore, we plan to create a Genetic Programming algorithm to model mathematical functions, simulations and measurements in the interval arithmetic format that can be used together with [**I**]PSO. The method should also be tested with high-dimensional problems.

[**I**]PSO can be further improved by changing the PSO topologies and mechanisms in order to improve global search. The use of the pre-processing step allows each box in $\mathbf{X}^{''}$ to be handled as a different search space. The consideration of other properties of Interval Arithmetic as well as the analysis of other classes of problems may also reveal new aspects related to the proposed method and are a field for future investigation. Another possibility is to use a local optimization tool in each box found in the pre-processing, rather than estimating a valid point by the boxes' center point. This might solve the issue of the result not necessarily improving as the ℓ parameter is increased.

9. Appendix

The optimization problems used as a benchmarks in this work are shown below. Tables 11-16 show the best results found by [**I**]PSO for each benchmark.

9.1. Welded Beam Design Optimization Problem (WB)

The Welded Beam Design Optimization Problem can be stated as

$$
\begin{aligned}
&Minimize:\\
&\quad f(\mathbf{x}) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14 + x_2);\\
&subject\ to:\\
&\quad g_1(\mathbf{x}) = \tau(\mathbf{x}) - 13\ 600 \leq 0;\\
&\quad g_2(\mathbf{x}) = \sigma(\mathbf{x}) - 30\ 000 \leq 0;\\
&\quad g_3(\mathbf{x}) = x_1 - x_4 \leq 0;\\
&\quad g_4(\mathbf{x}) = 0.10471(x_1^2) + 0.04811x_3x_4(14 + x_2) - 5 \leq 0;\\
&\quad g_5(\mathbf{x}) = 0.125 - x_1 \leq 0;\\
&\quad g_6(\mathbf{x}) = \delta(\mathbf{x}) - 0.25 \leq 0;\\
&\quad g_7(\mathbf{x}) = 6000 - Pc(\mathbf{x}) \leq 0;\\
&with:\\
&\quad \tau(\mathbf{x}) = \sqrt{(\tau')^2 + (2\tau'\tau'')x_2/2R + (\tau'')^2};\\
&\quad \tau' = 6000/(\sqrt{2}x_1x_2);\\
&\quad \tau'' = MR/J;\\
&\quad M = 6000(14 + x_2/2);\\
&\quad R = \sqrt{x_2^2/4 + (x_1 + x_3/2)^2};\\
&\quad J = 2x_1x_2\sqrt{2}(x_2^2/12 + (x_1 + x_3/2)^2);\\
&\quad \sigma(\mathbf{x}) = 504000/(x_4x_3^2);\\
&\quad \delta(\mathbf{x}) = 65856000/(30 \times 10^6 x_4x_3^3);\\
&\quad Pc(\mathbf{x}) = \tfrac{4.013(30\times10^6)}{196}\sqrt{x_3^2x_4^6/36}\left(1 - \tfrac{x_3}{28}\sqrt{\tfrac{30\times10^6}{4(12\times10^6)}}\right);\\
&\quad 0.1 \leq x_1,x_4 \leq 2.0;\\
&\quad 0.1 \leq x_2,x_3 \leq 10.0;
\end{aligned}
$$

Remark: The value found in [13] was better than the previously found optimal result, but is unfeasible.

9.2. Tension/Compression Spring Design Optimization Problem (CS)

The Tension/Compression Spring Design Optimization Problem can be stated as

$$
\begin{aligned}
&Minimize: \\
&\quad f(\mathbf{x}) = (x_3+2)x_2x_1^2; \\
&subject\ to: \\
&\quad g_1(\mathbf{x}) = 1 - x_2^3x_3/(7.178x_1^4) \leq 0; \\
&\quad g_2(\mathbf{x}) = \tfrac{4x_2^2 - x_1x_2}{12566(x_2x_1^3 - x_1^4)} + \tfrac{1}{5108x_1^2} - 1 \leq 0; \\
&\quad g_3(\mathbf{x}) = 1 - 140.45x_1/(x_2^2x_3) \leq 0; \\
&\quad g_4(\mathbf{x}) = (x_2+x_1)/1.5 \leq 0; \\
&with: \ 0.05 \leq x_1 \leq 2.0, \ 0.25 \leq x_2 \leq 1.3 \ and \\
&2.0 \leq x_3 \leq 15.0
\end{aligned}
$$

9.3. Speed Reducer Design Optimization Problem (SR)

The Speed Reducer Design Optimization Problem can be stated as

$$
\begin{aligned}
&Minimize: \\
f(\mathbf{x}) = \ & 0.7854x_1x_2^2(3.3333x_3^2 + 14.9334x_3 - 430934) - \\
& -1.508x_1(x_6^2 + x_7^2) + 7.4777(x_6^3 + x_7^3) + \\
& +0.7854(x_4x_6^2 + x_5x_7^2);
\end{aligned}
$$

$$
\begin{aligned}
&subject\ to: \\
&\quad g_1(\mathbf{x}) = 27/(x_1x_2^2x_3) - 1 \leq 0; \\
&\quad g_2(\mathbf{x}) = 397.5/(x_1x_2^2x_3^2) - 1 \leq 0; \\
&\quad g_3(\mathbf{x}) = 1.93x_4^3/(x_2x_3x_6^4) - 1 \leq 0; \\
&\quad g_4(\mathbf{x}) = 1.93x_5^3/(x_2x_3x_7^4) - 1 \leq 0; \\
&\quad g_5(\mathbf{x}) = \tfrac{1.0}{110x_6^3}\sqrt{\left(\tfrac{745.0x_4}{x_2x_3}\right)^2 + 16.9 \times 10^6} - 1 \leq 0; \\
&\quad g_6(\mathbf{x}) = \tfrac{1.0}{85x_7^3}\sqrt{\left(\tfrac{745.0x_5}{x_2x_3}\right)^2 + 157.5 \times 10^6} - 1 \leq 0; \\
&\quad g_7(\mathbf{x}) = x_2x_3/40 - 1 \leq 0; \\
&\quad g_8(\mathbf{x}) = 5x_2/x_1 - 1 \leq 0; \\
&\quad g_9(\mathbf{x}) = x_1/(12x_2) - 1 \leq 0; \\
&\quad g_{10}(\mathbf{x}) = (1.5x_6 + 1.9)/x_4 - 1 \leq 0; \\
&\quad g_{11}(\mathbf{x}) = (1.1x_7 + 1.9)/x_5 - 1 \leq 0; \\
&with: \ 2.6 \leq x_1 \leq 3.6, \ 0.7 \leq x_2 \leq 0.8, \\
&17 \leq x_3 \leq 283, \ 7.3 \leq x_4 \leq 8.3, \ 7.8 \leq x_5 \leq 8.3, \\
&2.9 \leq x_6 \leq 3.9 \ and \ 5.0 \leq x_7 \leq 5.5
\end{aligned}
$$

9.4. Speed Reducer Design Optimization Problem 2 (SR2)

SR2 is another version of the SR problem. SR2 differs from SR by changing the bound of the design variable x_5, so that $7.3 \leq x_5 \leq 8.3$.

9.5. Pressure Vessel Design Optimization Problem (PV)

The Pressure Vessel Design Optimization Problem can be stated as

$$
\begin{aligned}
&Minimize: \\
&\quad f(\mathbf{x}) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 \\
&\quad +19.84x_1^2x_3 \\
&subject\ to: \\
&\quad g_1(\mathbf{x}) = -x_1 + 0.0193x_3 \leq 0; \\
&\quad g_2(\mathbf{x}) = -x_2 + 0.00954x_3 \leq 0; \\
&\quad g_3(\mathbf{x}) = -\pi x_3^2x_4^2 - \tfrac{4}{3}\pi x_3^3 + 1\ 296\ 000 \leq 0; \\
&\quad g_4(\mathbf{x}) = x_4 - 240 \leq 0; \\
&with:\ x_1 = 0.0625n_1,\ x_2 = 0.0625n_2,\ where \\
&1 \leq n_1 \leq 99,\ 1 \leq n_2 \leq 99,\ 10.0 \leq x_3 \leq 200.0 \\
&and\ 10.0 \leq x_4 \leq 200.0,\ n_1, n_2 \in \mathbb{N}, x_3, x_4 \in \mathbb{R}
\end{aligned}
$$

Remark: The papers [15] and [13] claimed to have found better than the previously found optimal. Such results, however, are unfeasible, since the values they found for x_1 and x_2 are not discrete.

9.6. Multiple disk clutch brake design optimization problem (CB)

The Multiple Disk Clutch Brake Design Optimization Problem can be stated as

$$
\begin{aligned}
&Minimize:\\
&\quad f(\mathbf{x}) = \pi(x_2^2 - x_1^2)x_3(x_5+1)\rho;\\
&subject\ to:\\
&\quad g_1(\mathbf{x}) = x_2 - x_1 - \Delta R \geq 0;\\
&\quad g_2(\mathbf{x}) = L_{max} - (x_5+1)(x_3+\delta) \geq 0;\\
&\quad g_3(\mathbf{x}) = p_{max} - p_{rz} \geq 0;\\
&\quad g_4(\mathbf{x}) = p_{max} \times Vsr, max - p_{rz} \times Vsr \geq 0;\\
&\quad g_5(\mathbf{x}) = Vsr, max - Vsr \geq 0;\\
&\quad g_6(\mathbf{x}) = T_{max} - T \geq 0;\\
&\quad g_7(\mathbf{x}) = M_h - s \times M_s \geq 0;\\
&\quad g_8(\mathbf{x}) = T \geq 0;\\
&with:\\
&\quad M_h = \tfrac{2}{3}\mu x_4 x_5 \tfrac{x_2^3 - x_1^3}{x_2^2 - x_1^2}\text{N}\cdot\text{mm};\\
&\quad \omega = \pi n/30 \text{ rad/s};\\
&\quad A = \pi(x_2^2 - x_1^2)\ \text{mm}^2;\\
&\quad p_{rz} = \tfrac{x_4}{A}\text{N/mm}^2;\\
&\quad Vsr = \tfrac{\pi R_{sr} n}{30}\text{mm/s};\\
&\quad R_{sr} = \tfrac{2}{3}\tfrac{x_2^3 - x_1^3}{x_2^2 - x_1^2}\text{mm};\\
&\quad \Delta R = 20\text{mm};\ L_{max} = 30\text{mm};\\
&\quad \mu = 0.5;\ p_{max} = 1\text{MPa};\\
&\quad \rho = 0.0000078\text{kg/mm}^3;\\
&\quad Vsr, max = 10\text{m/s};\ \delta = 0.5\text{mm};\\
&\quad s = 1.5;\ T_{max} = 15\text{s};\\
&\quad n = 250\text{rpm};\ I_z = 55\text{kg}\cdot\text{m}^2;\\
&\quad M_s = 40\text{Nm};\ M_f = 3\text{Nm};\\
&\quad 60\text{mm} \leq x_1 \leq 80\text{mm};\\
&\quad 90\text{mm} \leq x_2 \leq 110\text{mm};\\
&\quad 1.0\text{mm} \leq x_3 \leq 3\text{mm};\\
&\quad 0 \leq x_4 \leq 1,000\text{N};\\
&\quad 2 \leq x_5 \leq 9;\\
&\quad x_1, x_2, x_5 \in \mathbb{N};\\
&\quad x_3 = 0.5n_1; n_1 \in \mathbb{N};\\
&\quad x_4 = 10n_2; n_2 \in \mathbb{N};
\end{aligned}
$$

Remark: The original CB problem had $1.5 \leq x_3$ [10]. In order to compare fairly with the literature [5], the lower bound of the design variable x_3 was changed, so that $1.0 \leq x_3$.

References

[1] Aguirre, A.H., Zavala, A., Diharce, E.V., Rionda, S.B., 2007. COPSO: Constrained Optimization via PSO Algorithm. Technical Report I-07-04/22-02-2007. Center for Research in Mathematics (CIMAT).

Table 11: Best result found by [**I**]PSO for the Pressure Vessel Problem

$f(\mathbf{x}^*)$	6059.7143350503729
x_1	0.8125
x_2	0.4375
x_3	42.098445595839479
x_4	176.6365958426332
$g_1(x)$	-2.97983859809392E-13
$g_2(x)$	-0.035880829015691396
$g_3(x)$	-1.3562384992837906E-8
$g_4(x)$	-63.3634041573668

Table 12: Best result found by [**I**]PSO for the Compression String Problem

$f(\mathbf{x}^*)$	0.012665232841936448
x_1	0.051688394316786956
x_2	0.35670169894030945
x_3	11.289906277646015
$g_1(x)$	-2.0062300709611236E-9
$g_2(x)$	-9.813179158157936E-10
$g_3(x)$	-4.053753932941942
$g_4(x)$	-0.7277399378286025

[2] Akhtar, S., Tai, K., Ray, T., 2002. A socio-behavioural simulation model for engineering design optimization. ENGINEERING OPTIMIZATION 34, 341–354. doi:10.1080/0305215021000001677.

[3] Arumugam, M., Chandramohan, A., Rao, M., 2005. Competitive approaches to PSO algorithms via new acceleration co-efficient variant with mutation operators, in: Selvaraj, H and Verma, B and DeCarvalho, A (Ed.), ICCIMA 2005: Sixth International Conference on Computational Intelligence and Multimedia Applications, Proceedings, pp. 225–230.

[4] Back, T., Hammel, U., Schwefel, H.P., 1997. Evolutionary computation: comments on the history and current state. IEEE Transactions on Evolutionary Computation 1, 3–17. doi:10.1109/4235.585888.

[5] Ben Guedria, N., 2016. Improved accelerated PSO algorithm for mechanical engineering optimization problems. APPLIED SOFT COMPUTING 40, 455–467. doi:10.1016/j.asoc.2015.10.048.

[6] Bernardino, H.S., Barbosa, H.J.C., Lemonge, A.C.C., 2007. A hybrid genetic algorithm for constrained optimization problems in mechanical engineering, in: 2007 IEEE CONGRESS ON EVOLUTIONARY COMPUTATION, VOLS 1-10, PROCEEDINGS, IEEE. pp. 646–653. doi:10.1109/CEC.2007.4424532.

Table 13: Best result found by [**I**]PSO for the Speed Reducer Problem

$f(\mathbf{x}^*)$	2996.3481649685305
x_1	3.5
x_2	0.69999999999999996
x_3	17
x_4	7.2999999999999998
x_5	7.7999999999999998
x_6	3.3502146660964498
x_7	5.2866832297579167
$g_1(x)$	-0.07391528039787332
$g_2(x)$	-0.1979985271419492
$g_3(x)$	-0.4991722481024208
$g_4(x)$	-0.901471697615325
$g_5(x)$	-1.9984014443252818E-15
$g_6(x)$	-3.3306690738754696E-16
$g_7(x)$	-0.7025
$g_8(x)$	0.0
$g_9(x)$	-0.5833333333333333
$g_{10}(x)$	-0.05132575354182545
$g_{11}(x)$	-0.010852365034139666

[7] Cagnina, L.C., Esquivel, S.C., Coello, C.A.C., 2008. Solving engineering optimization problems with the simple constrained particle swarm optimizer. Informatica 32, 319–326.

[8] Datta, R., Deb, K., 2016. Uniform adaptive scaling of equality and inequality constraints within hybrid evolutionary-cum-classical optimization. SOFT COMPUTING 20, 2367–2382. doi:10.1007/s00500-015-1646-0.

[9] De Falco, I., Della Cioppa, A., Maisto, D., Scafuri, U., Tarantino, E., 2014. An adaptive invasion-based model for distributed Differential Evolution. INFORMATION SCIENCES 278, 653–672. doi:10.1016/j.ins.2014.03.083.

[10] Deb, K., Srinivasan, A., 2006. Innovization: Innovating design principles through optimization, in: Keijzer, M (Ed.), GECCO 2006: Genetic and Evolutionary Computation Conference, Vol 1 and 2, ACM SIGEVO. ASSOC COMPUTING MACHINERY, 1515 BROADWAY, NEW YORK, NY 10036-9998 USA. pp. 1629–1636. 8th Annual Genetic and Evolutionary Computation Conference, Seattle, WA, JUL 08-12, 2006.

[11] Eberhart, R., Shi, Y., 2001. Particle swarm optimization: Developments, applications and resources, in: PROCEEDINGS OF THE 2001 CONGRESS ON EVOLUTIONARY COMPUTATION, VOLS 1 AND 2, IEEE. pp. 81–86.

Table 14: Best result found by [**I**]PSO for the Welded Beam Problem

$f(\mathbf{x}^*)$	1.7248523273365091
x_1	0.20572963082113713
x_2	3.4704888677326489
x_3	9.0366238866879289
x_4	0.20572964102998761
$g_1(x)$	-1.4176112017594278E-7
$g_2(x)$	-2.4231234419858083E-5
$g_3(x)$	-1.0208850487192223E-8
$g_4(x)$	-3.432983762316975
$g_5(x)$	-0.08072963082113713
$g_6(x)$	-0.23554032255855922
$g_7(x)$	-9.850136666500475E-5

[12] El-Gallad, A., El-Hawary, M., Sallam, A., 2001. Swarming of intelligent particles for solving the nonlinear constrained optimization problem. ENGINEERING INTELLIGENT SYSTEMS FOR ELECTRICAL ENGINEERING AND COMMUNICATIONS 9, 155–163.

[13] Garg, H., 2016. A hybrid PSO-GA algorithm for constrained optimization problems. APPLIED MATHEMATICS AND COMPUTATION 274, 292–305. doi:10.1016/j.amc.2015.11.001.

[14] Ghovvati, M., Khayati, G., Attar, H., Vaziri, A., 2016. Kinetic parameters estimation of protease production using penalty function method with hybrid genetic algorithm and particle swarm optimization. BIOTECHNOLOGY & BIOTECHNOLOGICAL EQUIPMENT 30, 404–410. doi:10.1080/13102818.2015.1134279.

[15] Gu, H.j., Xu, L., 2011. Perceptive particle swarm optimization algorithm for constrained optimization problems. J. Comput. App 31, 85–88.

[16] Gu, J., Shi, X., 2014. An Adaptive PSO Based on Motivation Mechanism and Acceleration Restraint Operator, in: 2014 IEEE CONGRESS ON EVOLUTIONARY COMPUTATION (CEC), IEEE. pp. 1328–1336.

[17] Hansen, E., Walster, G.W., 2003. Global optimization using interval analysis: revised and expanded. volume 264. CRC Press.

[18] He, S., Prempain, E., Wu, Q., 2004. An improved particle swarm optimizer for mechanical design optimization problems. ENGINEERING OPTIMIZATION 36, 585–605. doi:10.1080/03052150410001704854.

[19] Higashi, N., Iba, H., 2003. Particle swarm optimization with Gaussian mutation, in: PROCEEDINGS OF THE 2003 IEEE SWARM INTELLIGENCE SYMPOSIUM (SIS 03), IEEE. pp. 72–79. doi:10.1109/SIS.2003.1202250.

Table 15: Best result found by [**I**]PSO for the Multiple disk clutch brake design optimization Problem

$f(\mathbf{x}^*)$	0.31365661053440497
x_1	70
x_2	90
x_3	1
x_4	1000
x_5	3
$g_1(x)$	0.0
$g_2(x)$	24.0
$g_3(x)$	0.9005281605675655
$g_4(x)$	9.790581597222223
$g_5(x)$	7.894696589781841
$g_6(x)$	2.7585833547687812
$g_7(x)$	60.624999999999986
$g_8(x)$	12.241416645231219

[20] Hu, X., Eberhart, R., Shi, Y., 2003. Engineering optimization with particle swarm, in: PROCEEDINGS OF THE 2003 IEEE SWARM INTELLIGENCE SYMPOSIUM (SIS 03), IEEE. pp. 53–57.

[21] Hu, Y., 2010. A new evolutionary algorithm based on simplex crossover and pso mutation for constrained optimization problems, in: Computational Intelligence and Security (CIS), 2010 International Conference on, pp. 142–146. doi:10.1109/CIS.2010.38.

[22] Huang, F.z., Wang, L., He, Q., 2008. A Hybrid Differential Evolution with Double Populations for Constrained Optimization, in: 2008 IEEE CONGRESS ON EVOLUTIONARY COMPUTATION, VOLS 1-8, IEEE. pp. 18–25. doi:10.1109/CEC.2008.4630770.

[23] Jaulin, L., Kieffer, M., Didrit, O., Walter, E., 2001. Applied Interval Analysis. Springer London.

[24] Kennedy, J., Eberhart, R., 1995. Particle swarm optimization, in: 1995 IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS PROCEEDINGS, VOLS 1-6, IEEE. pp. 1942–1948. doi:10.1109/ICNN.1995.488968.

[25] Levitin, G., Hu, X., Dai, Y.S., 2007. Particle swarm optimization in reliability engineering, in: Computational Intelligence in Reliability Engineering. Springer Berlin Heidelberg. volume 40 of *Studies in Computational Intelligence*. chapter 4, pp. 83–112.

[26] Li, L.D., Li, X., Yu, X., 2008. A Multi-Objective Constraint-Handling Method with PSO Algorithm for Constrained Engineer-

Table 16: Best result found by [**I**]PSO for the Speed Reducer 2 Problem

$f(\mathbf{x}^*)$	2994.4710663190704
x_1	3.5000000000424829
x_2	0.69999999999999996
x_3	17
x_4	7.2999999999999998
x_5	7.7153199169683742
x_6	3.3502146661947205
x_7	5.2866544649959746
$g_1(x)$	-0.07391528040911399
$g_2(x)$	-0.19799852715168376
$g_3(x)$	-0.49917224816118333
$g_4(x)$	-0.9046439043536426
$g_5(x)$	-8.800005169007363E-11
$g_6(x)$	-7.886691300029725E-12
$g_7(x)$	-0.7025
$g_8(x)$	-1.2138068328226836E-11
$g_9(x)$	-0.5833333333282757
$g_{10}(x)$	-0.05132575352163282
$g_{11}(x)$	-7.093419185366656E-10

ing Optimization Problems, in: 2008 IEEE CONGRESS ON EVOLUTIONARY COMPUTATION, VOLS 1-8, IEEE. pp. 1528–1535. doi:10.1109/CEC.2008.4630995.

[27] Lin, L., Gen, M., Liang, Y., 2014. A Hybrid EA for High-dimensional Subspace Clustering Problem, in: 2014 IEEE CONGRESS ON EVOLUTIONARY COMPUTATION (CEC), IEEE. pp. 2855–2860.

[28] Mazhoud, I., Hadj-Hamou, K., Bigeon, J., Joyeux, P., 2013. Particle swarm optimization for solving engineering problems: A new constraint-handling mechanism. ENGINEERING APPLICATIONS OF ARTIFICIAL INTELLIGENCE 26, 1263–1273. doi:10.1016/j.engappai.2013.02.002.

[29] Mezura-Montes, E., Coello, C., Landa-Becerra, R., 2003. Engineering optimization using a simple evolutionary algorithm, in: Werner, B (Ed.), 15TH IEEE INTERNATIONAL CONFERENCE ON TOOLS WITH ARTIFICIAL INTELLIGENCE, PROCEEDINGS, IEEE COMPUTER SOC. pp. 149–156. doi:10.1109/TAI.2003.1250183.

[30] Moore, R.E., 1966. Interval analysis. volume 4. Prentice-Hall, Englewood Cliffs.

[31] Neuland, R., Maffei, R., Jaulin, L., Prestes, E., Kolberg, M., 2014. Improving the precision of AUVs localization in a hybrid interval-probabilistic approach using a set-inversion strategy. Unmanned Systems 02, 361–375.

[32] Parsopoulos, K., Vrahatis, M., 2002. Particle Swarm Optimization method for Constrained Optimization problems, in: Sincak, P and Vascak, J and Kvasnicka, V and Pospichal, J (Ed.), INTELLIGENT TECHNOLOGIES - THEORY AND APPLICATIONS: NEW TRENDS IN INTELLIGENT TECHNOLOGIES, IOS PRESS. pp. 214–220.

[33] Rao, S., 2009. Engineering Optimization: Theory and Practice. Wiley, Hoboken.

[34] Soares, G.L., 2008. Algoritmos Determinísticos e Evolucionares Intervalares para Otimização Robusta Multi-Objetivo. Ph.D. thesis. Universidade Federal de Minas Gerais and École Nationale Supérieure d'Ingénieurs and Université de Bretagne Occidentale.

[35] Solau, C., Marhic, B., Delahoche, L., Clerentin, A., Jolly-Desodt, A.M., Menga, D., 2011. Combination of interval analysis and PSO for optimization, in: PROCEEDINGS OF THE 7TH CONFERENCE OF THE EUROPEAN SOCIETY FOR FUZZY LOGIC AND TECHNOLOGY (EUSFLAT-2011) AND LFA-2011, ATLANTIS PRESS. pp. 978–985.

[36] Vitorino, L.N., Ribeiro, S.F., Bastos-Filho, C.J.A., 2012. A Hybrid Swarm Intelligence Optimizer based on Particles and Artificial Bees for High-Dimensional Search Spaces, in: 2012 IEEE CONGRESS ON EVOLUTIONARY COMPUTATION (CEC), IEEE.

[37] Worasucheep, C., 2008. Solving Constrained Engineering Optimization Problems by the Constrained PSO-DD, in: ECTI-CON 2008: PROCEEDINGS OF THE 2008 5TH INTERNATIONAL CONFERENCE ON ELECTRICAL ENGINEERING/ELECTRONICS, COMPUTER, TELECOMMUNICATIONS AND INFORMATION TECHNOLOGY, VOLS 1 AND 2, IEEE. pp. 5–8.

[38] Wu, H., Nie, C., Kuo, F.C., Leung, H., Colbourn, C.J., 2015. A Discrete Particle Swarm Optimization for Covering Array Generation. IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION 19, 575–591. doi:10.1109/TEVC.2014.2362532.

[39] Yu, K., Wang, X., Wang, Z., 2016. Constrained optimization based on improved teaching-learning-based optimization algorithm. INFORMATION SCIENCES 352, 61–78. doi:10.1016/j.ins.2016.02.054.

[40] Zhu, H., Pu, C., Eguchi, K., Gu, J., 2009. Euclidean particle swarm optimization, in: Intelligent Networks and Intelligent Systems, 2009. ICINIS '09. Second International Conference on, pp. 669–672. doi:10.1109/ICINIS.2009.171.

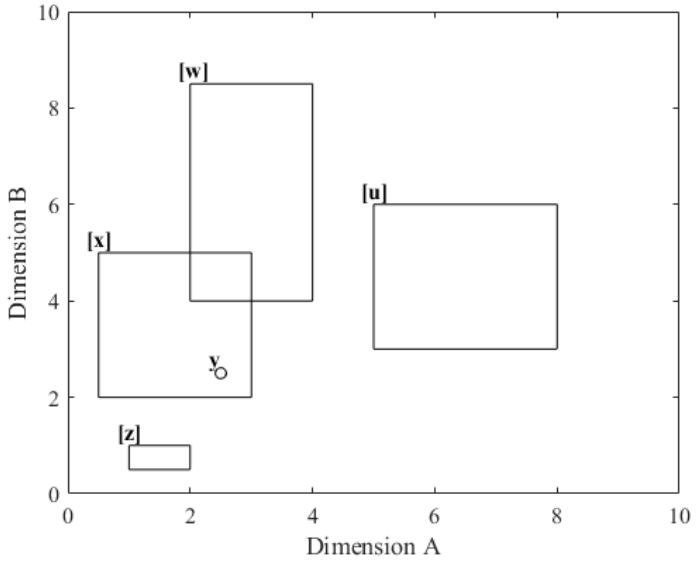


Figure 2: An example showing the point y and boxes [**x**], [**u**], [**w**] and [**z**]

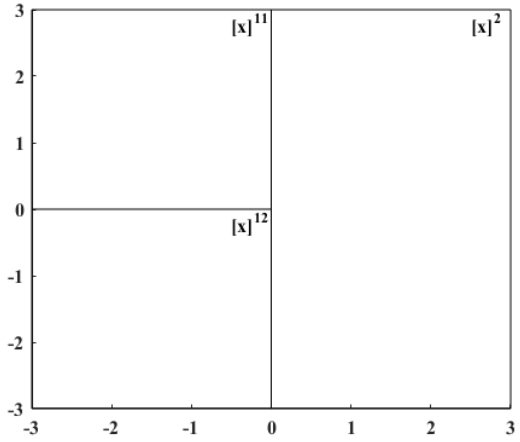


Figure 3: Bisecting the box [**x**] yields the boxes $[\mathbf{x}]^1$ and $[\mathbf{x}]^2$. Bisecting $[\mathbf{x}]^1$ creates the boxes $[\mathbf{x}]^{11}$ and $[\mathbf{x}]^{12}$.

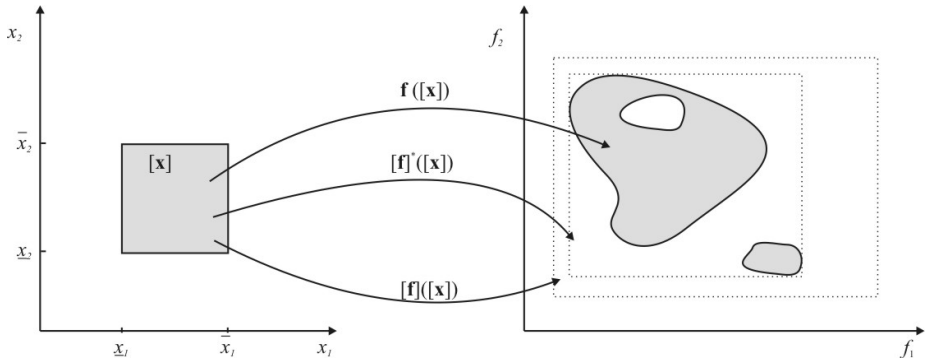


Figure 4: An example of an inclusion function [23]

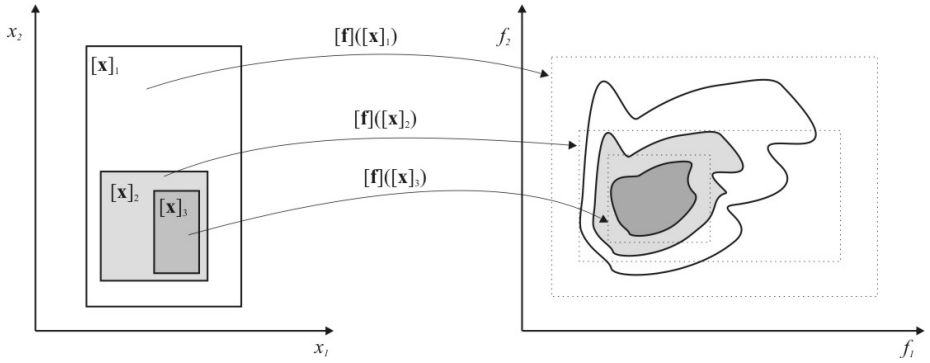


Figure 5: An example of a convergent monotonic inclusion function [23]

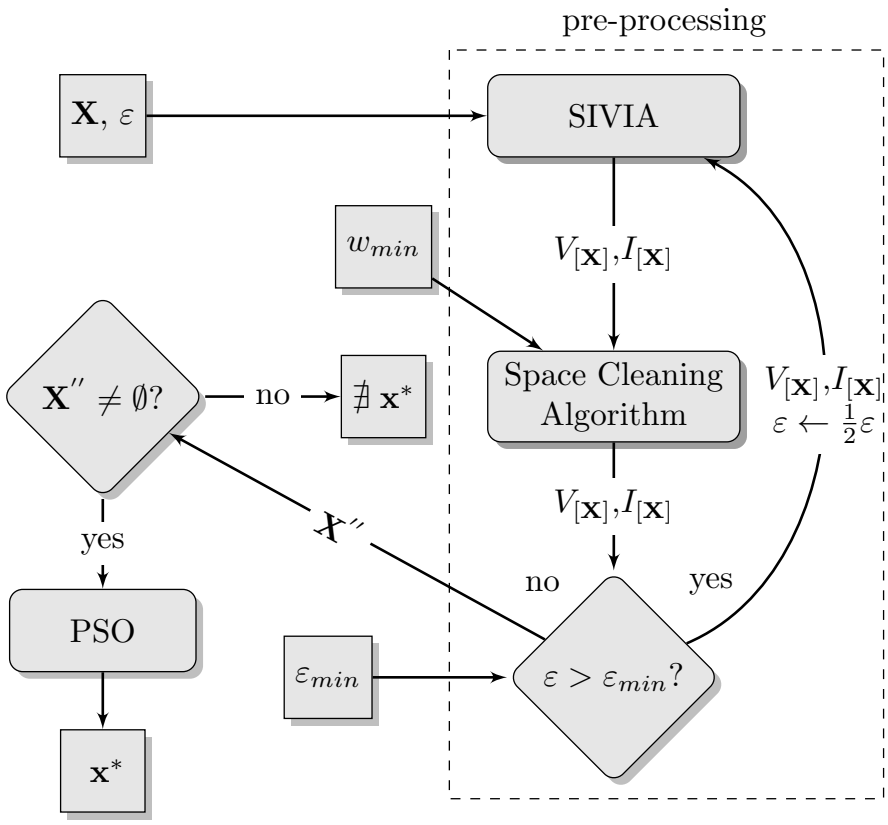
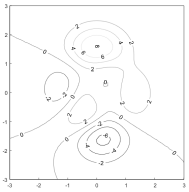
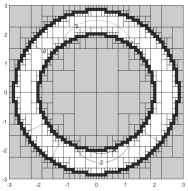


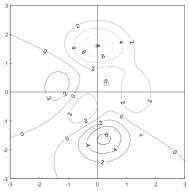
Figure 6: [**I**]PSO method. (Pre-processing: Alg. 1, Alg. 2, Alg. 3, Alg. 4. PSO: Alg. 5, Alg. 6)



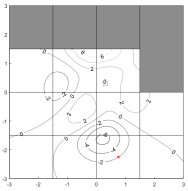
(a) $f(\mathbf{x}) \times \mathbf{X}$ at $\ell = 0$. $H_{\mathbf{X}''} = H_{\mathbf{X}}$, $H_{[\mathbf{x}]} = 36$ and $H_{\%} = 100.00$.



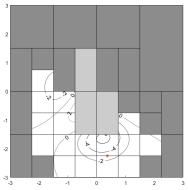
(b) $f(\mathbf{x}) \times \mathbf{X}'$, $\mathbf{X}'$ in blank boxes. $H_{\mathbf{X}''} = 14.45$, $H_{\%} = 40.14$.



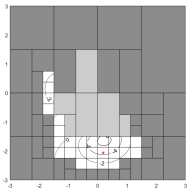
(c) $f(\mathbf{x}) \times \mathbf{X}''$ at $\ell = 1$. $H_{\%} = 100.00$.



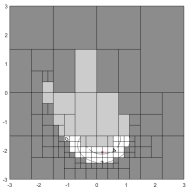
(d) $f(\mathbf{x}) \times \mathbf{X}''$ at $\ell = 2$, $H_{\%} = 68.75$



(e) $f(\mathbf{x}) \times \mathbf{X}''$ at $\ell = 3$, $H_{\%} = 25.00$.



(f) $f(\mathbf{x}) \times \mathbf{X}''$ at $\ell = 4$, $H_{\%} = 10.16$.



(g) $f(\mathbf{x}) \times \mathbf{X}''$ at $\ell = 5$, $H_{\%} = 3.42$.

(h) $f(\mathbf{x}) \times \mathbf{X}''$ at $\ell = 6$, $H_{\%} = 0.93$.

Figure 7: Pre-processing step of [**I**]PSO

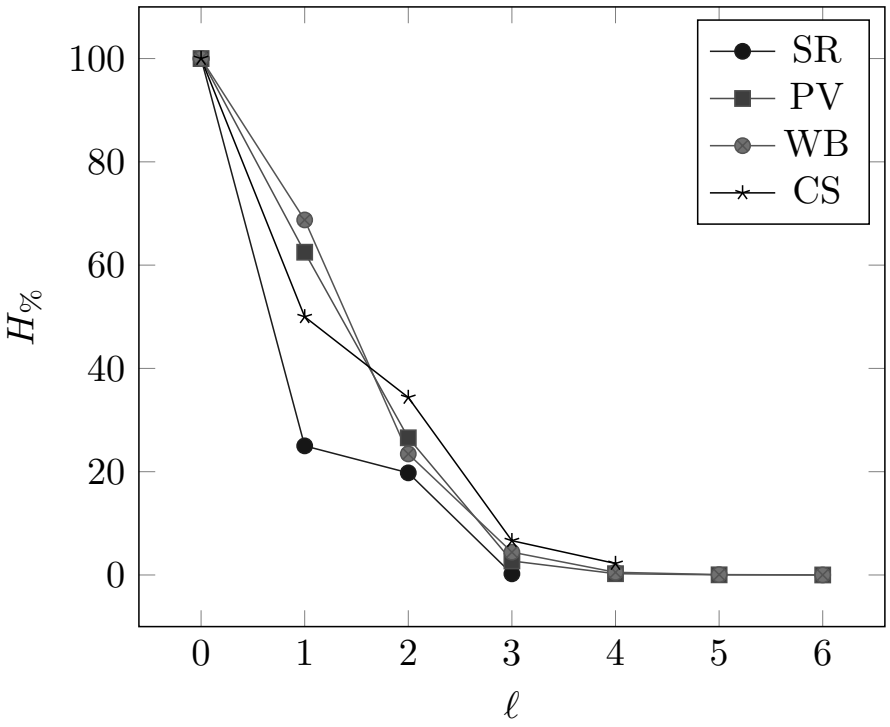


Figure 8: $H_{\%}$ remaining after each pre-processing iteration, for some problems. If the ℓ value was high enough, the decision maker could select the remaining search space as the global optimum

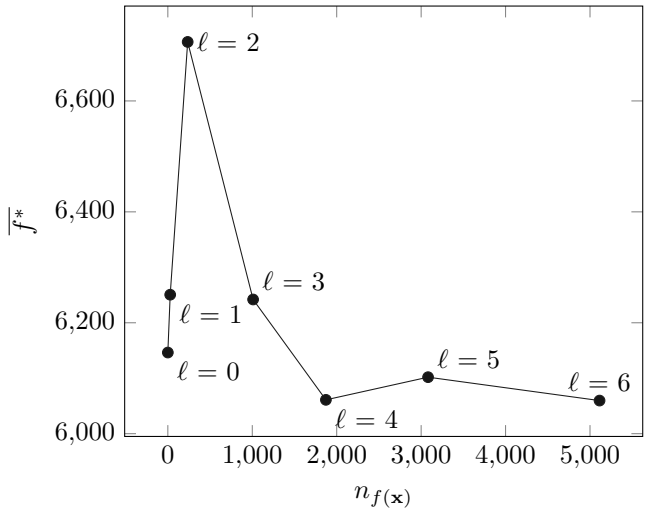


Figure 9: Number of function evaluations made during the pre-processing x $\overline{f^*}$, for the PV problem

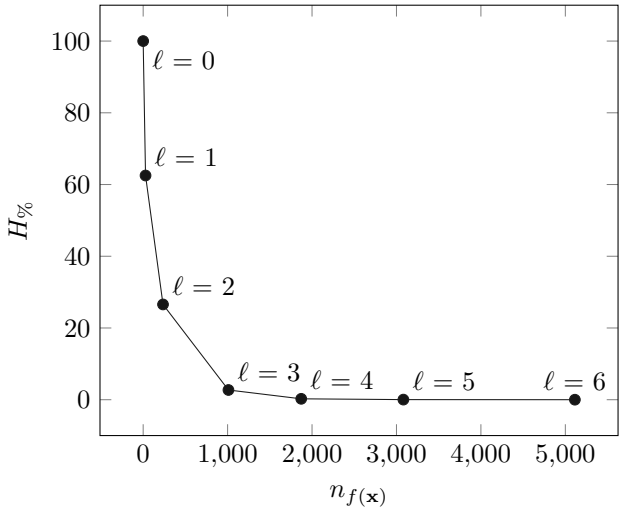


Figure 10: Number of function evaluations made during the pre-processing x $H_{\%}$, for the PV problem