A Constrained Iterative Topographical Global Optimization Applied to Engineering Optimization

Matheus Pedroza Ferreira[a], Marcelo Lisboa Rocha[a,*], Antônio J. Silva Neto[b], Wagner F. Sacco[c]

[a]*Departamento de Ciência da Computação, Universidade Federal do Tocantins, Quadra 109 Norte, Avenida NS-15, ALCNO-14, Palmas, Tocantins, Brazil*
[b]*Departamento de Engenharia Mecânica e Energia, Instituto Politécnico, Universidade do Estado do Rio de Janeiro IPRJ/UERJ, RJ, Brazil*
[c]*Instituto de Engenharia e Geociências, Universidade Federal do Oeste do Pará, PA, Brazil*

Abstract

Nonlinear optimization is an active line of research, given the wide range of scientific fields that benefit from its development. In the last years, the meta-heuristics proved to be one of the most effective methods to tackle difficult optimization problems, providing an alternative in cases where exact methods would be infeasible. In this work, we developed a method based on the Iterative Topographical Global Optimization meta-heuristic, which we call C-ITGO, incorporating specific mechanisms to solve nonlinearly constrained optimization problems. We use the method developed in this work to optimize eight complex engineering design problems and compare the results obtained here against several methods found in the literature. In the tests performed, C-ITGO outperforms all competing methods, achieving state of the art results for the problems considered.

Keywords: Meta-heuristics, ITGO, Optimization, Engineering Problems.

1. Introduction

In recent years, there has been an increase in the interest of applying meta-heuristics to solve difficult numerical optimization problems, mainly due to the impos-

*Corresponding authors
Email addresses: matheuspedrozaferreira@uft.edu.br (Matheus Pedroza Ferreira), mlisboa@uft.edu.br (Marcelo Lisboa Rocha), ajsneto@iprj.uerj.br (Antônio J. Silva Neto), wagner.sacco@ufopa.edu.br (Wagner F. Sacco)

sibility of obtaining good quality solutions to some complex problems using standard local optimization methods. It is usual to find nonlinear constrained real-world problems which are not smooth or not even continuous, ruling out the possibility to apply any gradient-based optimization procedure.

Even when the problem at hand has continuous derivatives, it may be multimodal, having many local optima. Any deterministic local procedure is confined to finding sub-optimal solutions for any multimodal problem of reasonable size. Many modern meta-heuristics, on the other hand, are capable of finding an optimal or nearly optimal solution to these problems within a feasible amount of time.

Along with the meta-heuristics, there exist a class of deterministic global optimization methods that can tackle multimodal optimization problems, assuring convergence to the global optimum with predefined tolerance. These methods are in general more complex than meta-heuristics, having rigorous convergence proofs and requiring certain mathematical background to be correctly understood and implemented (Sergeyev et al., 2013; Paulavičius & Žilinskas, 2014; Sergeyev et al., 2018). Some examples include the well know DIRECT (DIviding RECTangles) method (Jones et al., 1993), its locally biased variant (Gablonsky & Kelley, 2001), the ADC (Adaptive Diagonal Curves) algorithm (Sergeyev & Kvasov, 2006), and some more recent methods Kvasov & Sergeyev (2012); Fok et al. (2017); Paulavičius & Žilinskas (2016). A thorough comparison between meta-heuristics and deterministic global optimization algorithms is made in Sergeyev et al. (2018), using some novel methods.

Among the most recent meta-heuristics used to solve nonlinear constrained optimization problems, we can cite some well known methods, such as Particle Swarm Optimization (PSO) (Machado-Coelho et al., 2017; Guedria, 2016; Liang et al., 2010), Artificial Bee Colony (ABC) (Brajevic, 2015; Long et al., 2017), Differential Evolution (DE) (Sarker et al., 2014; Takahama & Sakai, 2010; Melo & Carosio, 2013), Genetic Algorithms (GA) (Saha et al., 2010) and many new methods, some inspired by nature (Gandomi et al., 2013; Eskandar et al., 2012; Sadollah et al., 2013).

The objective is to find the best solution in a search space that satisfies some criteria, expressed in the form of constraints, whose value is rated according to a function. For minimization, a general nonlinearly constrained optimization problem can be defined as follows:

$$\begin{aligned} \underset{\boldsymbol{x}}{\arg\min} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & g_j(\boldsymbol{x}) \leq 0, \qquad j = 1, ..., p \\ & h_k(\boldsymbol{x}) = 0, \qquad k = 1, ..., q \\ & l_i \leq x_i \leq u_i, \qquad i = 1, ..., n \end{aligned} \tag{1}$$

where $f(\cdot)$ is the objective function or fitness function, $\boldsymbol{x} = (x_1, ..., x_n)$ is a solution vector containing n continuous or discrete values, $g_j(\cdot),\ j = 1, ..., p$, and $h_k(\cdot),\ k = 1, ..., q$, are the inequality and equality constraints respectively, where both can be nonlinear. The vectors $\boldsymbol{l}$ and $\boldsymbol{u}$ define the lower and upper bounds of the solution space. We also define the sum of constraint violation as:

$$v(\boldsymbol{x}) = \sum_{j=1}^{p} max\{g_j(\boldsymbol{x}), 0\} + \sum_{k=1}^{q} |h_k(\boldsymbol{x})| \tag{2}$$

so a solution $\boldsymbol{x}$ is feasible only when $v(\boldsymbol{x}) = 0$.

In this work, we develop some modifications over a successful meta-heuristic for continuous optimization, known as Iterative Topographical Global Optimization (ITGO) (Törn & Viitanen, 1996). The modifications consist of adapting ITGO to work with constrained optimization issues, calling this new algorithm as Constrained version of ITGO (C-ITGO). We compare the results obtained on eight complex constrained engineering design problems, used as a benchmark against several state-of-the-art methods. We show that C-ITGO outperforms the best competing techniques of literature, mainly relative to the number of function evaluations that characterize the execution effort of the technique.

The structure of the paper is as follows. Section 2 presents the C-ITGO method, the details of its implementation and the parameters used in the tests. We compare the developed algorithm against the competing methods in Section 3, and make the final conclusions in Section 4.

2. The Constrained Topographical Global Optimization Algorithm

The TGO algorithm, which stands for *Topographical Global Optimization*, is a metaheuristic developed by Törn & Viitanen (1992) for solving continuous optimization problems, possibly non-smooth and multimodal. The method has three steps: the random uniform generation of a population of solutions over the domain of the problem, the selection of some of the individuals of the population based on the topography of the function to be optimized, and the application of some sort of local search in the selected elements.

Initially, the TGO creates a population of size M denoted by $P = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_M\}$, uniformly distributed in the solution space. The TGO performance depends directly on the algorithm used in the generation of the population, and, consequently, on the pseudo-random number generator. In this work, we used the Sobol sequences (Sobol, 1967; Henderson et al., 2015), which has a significantly more uniform distribution than a state-of-the-art pseudorandom number generator. The Figure 1 shows the distribution of 300 points in the domain $[0, 1]^2$, with 150 of them (blue dots) generated by the pseudo-random generator Mersenne Twister proposed by Matsumoto & Nishimura (1998). The other points (red dots) are the 150 first elements of a Sobol sequence. From Figure 1 is possible to observe that the points generated by Sobol sequence cover the space more uniformly than that of Mersenne Twister.

The next step is the construction of a graph over possible solutions that incorporate information regarding the topography of the function $f(\cdot)$. Consider now a integer $K > 0$. We define the $KNN_K(\boldsymbol{x})$ neighborhood as the set of the K elements of the population P, different of $\boldsymbol{x}$, that have the smallest euclidean distance from $\boldsymbol{x}$.

A directed graph is then defined, having a vertex for every solution in the population. For each individual $\boldsymbol{x}$, a directed edge is created pointing to every element $\boldsymbol{y}$ of the population P such that $\boldsymbol{y} \in KNN_K(\boldsymbol{x})$ and $f(\boldsymbol{x}) \leq f(\boldsymbol{y})$. With this construction, every vertex has at most K outgoing edges. A topography matrix $A \in \mathbb{R}^{M \times M}$ is created based on this graph, defined as:

$$A_{i,j} = \begin{cases} 1, & \text{if } f(\boldsymbol{x}_i) \leq f(\boldsymbol{x}_j) \text{ and } \boldsymbol{x}_j \in KNN_K(\boldsymbol{x}_i) \\ 0, & \text{otherwise} \end{cases}$$

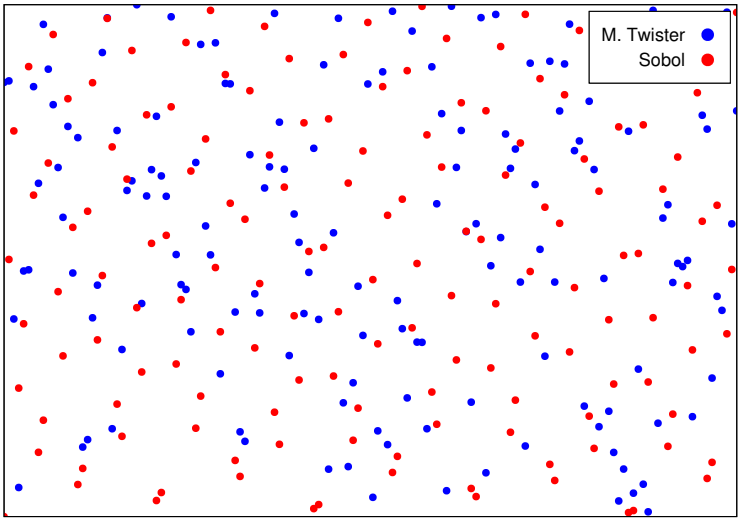


Figure 1: Example showing the distribution of points in the domain $[0, 1]^2$ by the Mersenne Twister generator (blue points) and by a Sobol sequence (red points).

The matrix A tells us if the element $\boldsymbol{x}_i$ has better fitness than the element $\boldsymbol{x}_j$, given that $\boldsymbol{x}_j$ is in the KNN set of $\boldsymbol{x}_i$. The graph defined previously has a direct link with the topography matrix: a directed edge between elements $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ only exists if $A_{i,j} = 1$. We have then created an ordering of the elements of the population based on the position in space, the objective function value, and the neighborhood defined by K.

The topographic heuristic is based on selecting every individual $\boldsymbol{x}_i$ such that $\sum_j^M A_{i,j} = K$, that is, every individual that has better fitness than every element of its KNN set. Likewise, every vertex that has K outgoing edges. These elements are considered local optimal points based on the estimated topography of the function $f(\cdot)$.

As there is no guarantee that the individuals selected are really local optimal points (as we only have an estimate of the function's surface given by the topography created), the application of some algorithm for fine-tuning is necessary.

The last step of the TGO method consists in applying some sort of local search in every local optimal point based on the topographic heuristic. Regarding the local search, many procedures used in the TGO are found in the literature, such as pattern search algorithms (Sacco et al., 2014), derivative-based (Henderson et al., 2015) and

derivative-free (Jardim et al., 2015) optimization methods. The local search algorithm depends directly on the properties of the function to be optimized and is of significative impact in the overall performance of the method.

We consider now a simple example. Let the function $f(\cdot)$ be defined by $f(x, y) = sin(x^2) + sin(y^2)$, the population P composed by the 10 elements:

$$\begin{aligned} \boldsymbol{x}_1 &= (-0.2, 0.16), \quad \boldsymbol{x}_2 = (1.2, -0.3), \quad \boldsymbol{x}_3 = (-0.6, 1.2) \\ \boldsymbol{x}_4 &= (-0.9, 2.4), \quad \boldsymbol{x}_5 = (2.0, 2.0), \quad \boldsymbol{x}_6 = (2.7, 0.3) \\ \boldsymbol{x}_7 &= (0.3, 2.2), \quad \boldsymbol{x}_8 = (2.0, -0.2), \quad \boldsymbol{x}_9 = (1.3, 2.8), \quad \boldsymbol{x}_{10} = (1.3, 1.2) \end{aligned}$$

and the KNN set determined by the $K = 3$ closest neighbors. Figure 2 shows the contour plot of the function and the directed graph created by the population. The topography matrix and the sum of each row are the following:

$$A = \left(\begin{array}{cccccccccc|c} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \mathbf{3} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \mathbf{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & \mathbf{3} \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \mathbf{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & \mathbf{2} \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & \mathbf{3} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} \end{array}\right)$$

According with the graph and the topography matrix created, it is easy to observe that only the elements $\boldsymbol{x}_1$, $\boldsymbol{x}_5$ and $\boldsymbol{x}_8$ are local optimum, that is, has K outgoing edges, satisfying $\sum_j^M A_{1,j} = \sum_j^M A_{5,j} = \sum_j^M A_{8,j} = K = 3$.

Even after the application of local search, there is no guarantee that the global optimum was found. It is common to successively apply the TGO method many times, with new elements at each iteration. This procedure is called *ITGO* (*Iterative Topographical Global Optimization*), and consists simply in the iterative application of the

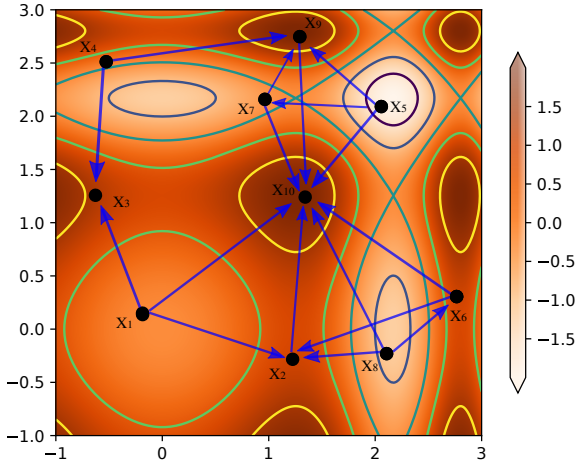


Figure 2: Contour plot of the function $f(x, y) = sin(x^2) + sin(y^2)$, and the graph based on the population.

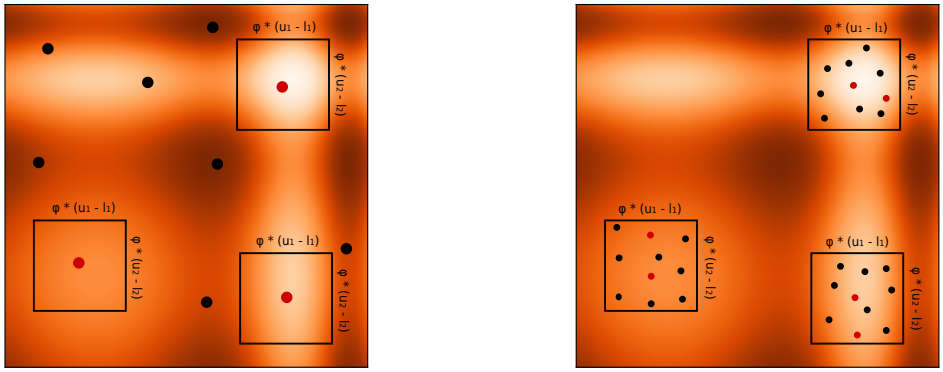
TGO method, returning the best element found during all the process.

2.1. Space Reduction

In Jardim et al. (2015) a heuristic for space reduction is proposed for the ITGO. After the individual selection step, a sub-space is created around every element, with space reduced in each dimension by $\phi \in (0, 1)$. A new population is generated for every space created, and new elements are selected, for each population. The process is repeated for a defined number of iterations until the execution of local search on selected elements of the last space reduction.

An example involving the application of this heuristic is presented in Figure 3, again for the function $f(x, y) = sin(x^2) + sin(y^2)$, in the domain $[-1, 3]^2$, with the first population composed by the same 10 points described previously. The Figure 3a shows the selection of the points $\boldsymbol{x}_1$, $\boldsymbol{x}_5$ and $\boldsymbol{x}_8$, with a space reduced by $\phi = 0.25$. In the Figure 3b we can observe the generation of populations for each new subspace. The red points are the individuals considered local optimum based on the topographic heuristic. It is worth noting that the local optimal points of the previous population are kept in the next population.

At each iteration, a new population size is set, along with a new value for K, which is usually smaller than in the previous iteration.



(a) (b)

Figure 3: Example showing the application of the space reduction heuristic. In (a) we have the space reduction around the three individuals of Figure 2 ($\boldsymbol{x}_1$, $\boldsymbol{x}_5$ and $\boldsymbol{x}_8$) in red. In (b) are the respective populations generated in each restricted space to apply ITGO, with the local optimal points in red.

2.2. Constrained Optimization

Until the moment, the ITGO algorithm was presented only for unconstrained optimization problems. ITGO was applied previously to constrained optimization problems (Sacco et al., 2014; Henderson et al., 2015), handling constraints by using a specific local search procedure or by penalizing infeasible individuals.

In this work, we propose a different approach, adopting a mechanism for comparing solutions proposed in Deb (2000), and used in some very competitive metaheuristics, specially on differential evolution algorithms (Sarker et al., 2014; Takahama & Sakai, 2010; Montes et al., 2006). As in this work we propose a constrained version of ITGO, in the rest of the paper it is named as C-ITGO, standing for Constrained ITGO. The method for comparison comprises the three steps criteria:

- Between two feasible solutions, the fittest one is better.
- A feasible solution is always preferred over an infeasible solution, irrespectively of its fitness.
- Between two infeasible solutions, the one with the smallest sum of constraint violation (equation 2) is better.

In the topographical heuristic step, we compare two solutions using this three steps criteria with probability α, and otherwise, we compare only the fitness value. What we try to achieve with this heuristic is keeping promising solutions with small violation of constraints for further exploration in the local search step.

We generate a symmetric matrix of random numbers R, with $R_{i,j} = R_{j,i} \in [0, 1]$. If $R_{i,j} < \alpha$, then we compare elements $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ using the three steps criteria. Otherwise, only the fitness value is compared. In general, with this configuration, the number of local optimal points selected by the topographical heuristic is smaller, specially on problems with a small feasible area.

An important thing to notice is that some distributions of the individuals, with populations of small size and a high value of K, may not generate any local optimum in the topographical heuristic. In the case of none of the individuals are selected as local optimum given the current population, we take the best individual according to the three steps criteria as the only local optimum of that population. In practice, however, it is rather unusual to happen.

The best individual in the whole search is always selected using the three steps criteria described above, since feasible solutions are generally preferred. In addition to this new topographical selection method, we also use local search algorithms for constrained optimization.

2.3. Implementation

We now present an overview of the implementation of the method, along with pseudocodes describing all necessary steps. Algorithm 1 shows the general procedure for the C-ITGO algorithm.

The parameters are the functions $f(\cdot)$ and $v(\cdot)$, which return the function value and sum of constraints violation respectively, as defined in equations 1 and 2, the lower and upper bound vectors $\boldsymbol{l}$ and $\boldsymbol{u}$, the vector of population sizes $\boldsymbol{PS}$, the vector of the K values $\boldsymbol{KS}$, the maximum number Max_LS of elements to execute the local search, the maximum number of function calls allowed in the local search, LS_1 and LS_2, the space reduction factor ϕ and the probability for the three steps comparison α.

Algorithm 1 C-ITGO($f(\cdot)$, $v(\cdot)$, $\boldsymbol{l}$, $\boldsymbol{u}$, $\boldsymbol{PS}$, $\boldsymbol{KS}$, Max_LS, LS_1, LS_2, ϕ, α)

1: $\boldsymbol{best} \leftarrow random(\boldsymbol{l}, \boldsymbol{u})$;
2: **while** $Converged(\boldsymbol{best}) =$ **False do**
3: $\quad Populations \leftarrow \{\boldsymbol{x} = \boldsymbol{x}^1, ..., \boldsymbol{x}^{\boldsymbol{PS}(1)} : \boldsymbol{x} \in random(\boldsymbol{l}, \boldsymbol{u})\}$;

4: $TopoBest \leftarrow \{\}$;
5: **for** $p \in \{1, ..., |\boldsymbol{PS}|\}$ **do**
6: $NewPops \leftarrow \{\}$;
7: **for** $pop \in Populations$ **do**
8: $Topo_p \leftarrow TopographicalHeuristic(f(\cdot), v(\cdot), pop, \boldsymbol{KS}(p), \alpha)$;
9: **if** $p < |\boldsymbol{PS}|$ **then**
10: **for** $\boldsymbol{x} \in Topo_p$ **do**
11: $NewPops \leftarrow NewPops \cup CreatePop(\boldsymbol{x}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{PS}(p+1), \phi^p)$;
12: **else**
13: $TopoBest \leftarrow TopoBest \cup Topo_p$;
14: **if** $p < |\boldsymbol{PS}|$ **then**
15: $Populations \leftarrow NewPops$;
16: $TopoBest \leftarrow \{\boldsymbol{x} = \boldsymbol{x}^1, ..., \boldsymbol{x}^{min(|TopoBest|, Max\_LS)} : \boldsymbol{x} \in sort(TopoBest)\}$;
17: **for** $\boldsymbol{x} \in TopoBest$ **do**
18: $\boldsymbol{x} \leftarrow LocalSearch(f(\cdot), v(\cdot), \boldsymbol{x}, LS_1)$;
19: **if** $Compare(f(\cdot), v(\cdot), \boldsymbol{x}, \boldsymbol{best})$ **or** $f(\boldsymbol{x}) < f(\boldsymbol{best})$ **then**
20: $\boldsymbol{x} \leftarrow LocalSearch(f(\cdot), v(\cdot), \boldsymbol{x}, LS_2)$;
21: **if** $Compare(f(\cdot), v(\cdot), \boldsymbol{x}, \boldsymbol{best})$ **then**
22: $\boldsymbol{best} \leftarrow \boldsymbol{x}$;
23: **return** $best$;

The first line initializes the vector $\boldsymbol{best}$ randomly within the range $[\boldsymbol{l}, \boldsymbol{u}]$, which is the variable that saves the best element found in the search. The main loop starts at line 2, where we check for convergence. The function $Converged$ is problem dependent and may take into account the number of iterations, the number of function calls, the convergence of the best individual or any other stopping criteria.

Line 3 initializes the vector of current populations, with all the $\boldsymbol{PS}(1)$ elements inside the bounds of the problem ($[\boldsymbol{l}, \boldsymbol{u}]$). The set $TopoBest$ at line 4 comprises the best local optimum elements found in all populations and is initialized empty. We use $random$ here to denote the generation of a random scalar or vector for simplicity, although in the implementation we used Sobol sequences for the generation of the individuals.

For each population size (line 5), which is the number of space reductions plus one, we generate new populations, starting from the empty set at line 6. Here, we use $|\cdot|$ to denote the number of elements inside a set or a vector. Line 7 loops through all the populations and execute the topographical heuristic ($TopographicalHeuristic$ function), with $K = \boldsymbol{KS}(p)$. The variable $Topo_p$, at line 8, saves the local optimal points selected from the current population.

If the current space reduction is not the last ($p < |\boldsymbol{PS}|$, line 9), we create a new

population around every point in the $Topo_p$ set, with size $\boldsymbol{PS}(p+1)$ and space reduced at every dimension by ϕ^p (lines 10 and 11). Otherwise, if it is the last space reduction, we have to save the local optimal points in the set $TopoBest$ to apply local search (lines 12 and 13).

Algorithm 2 CreatePop($\boldsymbol{x}_b$, $\boldsymbol{l}$, $\boldsymbol{u}$, $popSize$, ϕ)

1: $\boldsymbol{l}' \leftarrow max(\boldsymbol{l}, \boldsymbol{x}_b - (0.5 * \phi) * (\boldsymbol{u} - \boldsymbol{l}))$;
2: $\boldsymbol{u}' \leftarrow min(\boldsymbol{u}, \boldsymbol{x}_b + (0.5 * \phi) * (\boldsymbol{u} - \boldsymbol{l}))$;
3: $Population \leftarrow \{\boldsymbol{x}_b\} \cup \{\boldsymbol{x} = \boldsymbol{x}^1, ..., \boldsymbol{x}^{popSize-1} : \boldsymbol{x} \in random(\boldsymbol{l}', \boldsymbol{u}')\}$;
4: **return** $Population$;

The $CreatePop$ procedure is shown in algorithm 2. Given an individual $\boldsymbol{x}_b$ (a selected local optimum), the function returns a new randomly generated population with $popSize - 1$ individuals around this solution, in the original space reduced by the fraction ϕ. If the point is closer than $0.5 * (u_i - l_i),\ i = 1, ..., n$, from the lower or upper bounds at dimension i, the limits of that new population are taken to be the original bounds. The min and max operations are executed element by element. The solution $\boldsymbol{x}_b$ is also added to the new population.

At line 14, we check again if the current space reduction is not the last, and update the current populations at line 15, if the condition holds. Line 16 selects the Max_LS best elements of the set $TopoBest$, using the three steps criteria comparison as sorting criteria. If $|TopoBest| < Max\_LS$, we keep all the elements.

We loop through all the elements of the now sorted $TopoBest$ set at line 17, and apply local search, with at maximum LS_1 function evaluations, at line 18. The function $Compare$ (line 19) is the three steps comparison, returning true if the first solution (third argument) is better than the second solution (fourth argument), and returns false otherwise. The element returned by the local search procedure is compared against the best element found in the whole search at line 19. If it is better than the best element found (according to the three steps criteria), or if its fitness is better, a new local search procedure is applied, now with LS_2 iterations, at line 20.

The number of function calls is usually the determining performance factor. So, we wish to do as few local search iterations as possible, since a call to the local search algorithm typically makes hundreds or thousands of function evaluations. Here, we set $LS_2 > LS_1$, so every element undergoes LS_1 function evaluations in the local search, but we only search finely for promising solutions, setting higher values for LS_2. Usually, the second local search is only necessary for problems where very finely

tuned solutions are required.

At line 21 we compare again the solution generated by the second local search ($\boldsymbol{x}$) against the best element found in the whole search ($\boldsymbol{best}$) using the three steps criteria. If the new solution is better, we set $\boldsymbol{best}$ to that solution at line 22 and continue the loop for applying local search in the other elements. At the end of the execution, when $Converged$ in the outer loop returns true, we return the best solution found in the whole search at line 23. In practice, however, we may stop the algorithm as soon as the optimal solution is found or when the maximum number of function evaluations is reached.

Finally, let us explain the $TopographicalHeuristic$ procedure, shown in algorithm 3. The function takes as parameters: the objective function $f(\cdot)$ and the sum of constraints function $v(\cdot)$, the current population to be evaluated, the value K for the KNN set, and the probability α, for the three steps comparison.

Algorithm 3 TopographicalHeuristic($f(\cdot)$, $v(\cdot)$, $Population$, K, α)

1: $M \leftarrow |Population|$;
2: $\boldsymbol{best} \leftarrow Population(0)$;
3: $KNN_K \leftarrow Build\_KNN(Population, K)$;
4: $\boldsymbol{R} \leftarrow random([0,1]^{M \times M})$;
5: $TopoBest \leftarrow \{\}$;
6: **for** $i \in \{1, ..., |Population|\}$ **do**
7: $\quad insert \leftarrow \boldsymbol{True}$;
8: $\quad$ **for** $j \in \{1, ..., |Population|\} \cap \{\boldsymbol{x}_j \in KNN_K(\boldsymbol{x}_i)\}$ **do**
9: $\quad\quad$ **if** $\boldsymbol{R}_{i,j} < \alpha$ **then**
10: $\quad\quad\quad insert \leftarrow insert \,\&\, Compare(f(\cdot), v(\cdot), \boldsymbol{x}_i, \boldsymbol{x}_j)$;
11: $\quad\quad$ **else**
12: $\quad\quad\quad insert \leftarrow insert \,\&\, (f(\boldsymbol{x}) < f(\boldsymbol{y}))$;
13: $\quad$ **if** $insert = \boldsymbol{True}$ **then**
14: $\quad\quad TopoBest \leftarrow TopoBest \cup \{\boldsymbol{x}_i\}$;
15: $\quad$ **if** $Compare(f(\cdot), v(\cdot), \boldsymbol{x}_i, \boldsymbol{best})$ **then**
16: $\quad\quad \boldsymbol{best} \leftarrow \boldsymbol{x}_i$;
17: **if** $|TopoBest| = 0$ **then**
18: $\quad TopoBest \leftarrow \{\boldsymbol{best}\}$;
19: **return** $TopoBest$;

Lines 1-5 simply initialize the necessary structures. Namely, the number of elements M in $Population$, the $\boldsymbol{best}$ vector, which is the best element of the entire population based on three steps comparison (initially set to any individual of the population), the KNN_K structure based on the elements of the population, which is a mapping

from a solution vector $\boldsymbol{x}$ to a set of the vectors that belong to the KNN_K of $\boldsymbol{x}$, the random symmetric matrix R, with every element in the range $[0, 1]$, and the $TopoBest$ set, containing the local optimal points, initially empty.

At line 6 we loop through all indices of $Population$, and set the $insert$ flag to $\boldsymbol{True}$ at line 7, which indicates if the individual $\boldsymbol{x}_i$ is a local optimal point. For every index j, such that $\boldsymbol{x}_j$ is in the set $KNN_K(\boldsymbol{x}_i)$ (line 8), we compare $\boldsymbol{x}_j$ with $\boldsymbol{x}_i$. If $R_{i,j} < \alpha$, then we set the flag $insert$ to the boolean result of applying the *and* operator ($\&$) to $insert$ and the result of $Compare$ (lines 9-10). That is, if $Compare$ returns false at least one time for any j, $insert$ will also be false for the index i. If $R_{i,j} >= \alpha$, then we execute the same procedure, but now using fitness only comparison, at lines 11-12.

At line 13 we check if $insert = \boldsymbol{True}$, and, if so, $\boldsymbol{x}_i$ is a local optimum, and we insert it in the $TopoBest$ set, at line 14. Lines 15-16 select the best element found in the whole population based on the three steps comparison, and stores that solution in the variable $\boldsymbol{best}$. In case of no solution is selected as a local optimum, the set $TopoBest$ is composed only of the $\boldsymbol{best}$ element (lines 17-18). At line 19 we return the $TopoBest$ set, containing all the local optimal points (or the $\boldsymbol{best}$ element).

2.4. Local Search

In this section, we discuss the three different methods of local search used in C-ITGO and their implications on the final performance.

As we used Matlab to program C-ITGO, a natural choice for local search is the optimization toolbox. In the case of real non-linearly constrained problems, we used the *SQP* (Sequential Quadratic Programming), present in the *fmincon* package (MathWorks, 2017). The basic SQP algorithm is described in Chapter 18 of Nocedal & Wright (2006), although the actual implementation used in fmincon uses some additional heuristics.

A very successful method, also implemented in Matlab, that uses that same package (SQP of fmincon) is the *MVMO* (Mean-Variance Mapping Optimization) (Rueda & Erlich, 2016), winner of two different categories of the IEEE Congress on Evolutionary Computation competition on real optimization in 2016.

For mixed integer problems with nonlinear constraints, we used the *OPTI* toolbox (Currie, 2014), which has many algorithms specialized for mixed integer programming. Specifically, we used the *BONMIN* (Basic Open-source Nonlinear Mixed INteger programming) (Biegler et al., 2015) and the *NOMAD* (Nonlinear optimization with

the MADS algorithm) (Le Digabel, 2011) solvers.

We emphasize here that any kind of local search can be used in conjunction with C-ITGO. We could use for example a specialized local search for a given problem. In this work, both problems on continuous and integer domains are treated in the same way, changing only the local search procedure.

It is true that some simpler problems can be solved solely by using an exact method such as those cited above, for example by calling the procedure at different random points. However, in multimodal objective functions with nonlinear constraints, it is usually not possible to find the global optimum. And even if a problem can be solved by an exact method, the number of function evaluations might be very large.

The objective is not to rely heavily on the local search procedure. Rather, what we want to achieve with the topographical heuristic is provide solutions close to a local or global optimum, so that any reasonably good local search can converge with relative ease. In this work, the maximum number of function evaluations allowed in the first call to the local search procedure is kept as small as possible, and, in most cases, it is enough to find optimal or nearly optimal solutions.

2.5. Parameters

Given the differences in the number of variables, constraints, size of the space, number of function calls to converge reported by competing methods, and general complexity of the problems considered in this work, we selected experimentally specific parameters for each problem aiming to obtain the best performance of C-ITGO. This is a general approach taken for most of the optimization methods compared here. The parameters were selected so as to find optimal or near-optimal solutions while minimizing the number of function evaluations (NFEs).

Table 1 shows the choice of the parameters for the eight engineering design problems we consider: Welded Beam (WB), Tension/Compression Spring (TC), Three-Bar truss (TB), Speed Reducer (SRI and SRII), Pressure Vessel (PV), Gear Train (GT) and Multiple Disk clutch brake (MD). We will comment each problem and the results obtained by C-ITGO and other competing methods in Section 3.

For all problems, we only use a single space reduction, which generates two populations and two values for K, namely PS_1 (first population size, before space reduction) and PS_2 (second population size, after space reduction), K_1 and K_2 (also before and

$Prob/$ $Param$	PS_1	PS_2	K_1	K_2	α	ϕ	$LS1$	$LS2$	$MaxLS$	$LSType$
WB	100	10	10	3	0.5	0.2	100	200	5	**SQP**
TC	50	10	8	3	0.5	0.2	100	200	5	**SQP**
TB	30	5	5	2	0.5	0.2	20	70	5	**SQP**
SRI	150	10	10	3	0.5	0.2	100	200	5	**BONMIN**
SRII	100	10	10	3	0.5	0.2	50	100	5	**SQP**
PV	50	10	8	3	0.5	0.5	30	100	5	**BONMIN**
GT	20	5	5	2	0.5	0.7	30	100	5	**NOMAD**
MD	20	5	7	2	0.5	0.7	100	200	5	**NOMAD**

Table 1: Parameters of C-ITGO for each engineering design problem.

after space reduction, respectively). For problems with a considerably large number of variables, it may be beneficial to use more than one space reduction.

The first population size is generally chosen to be proportional to the problem's domain size. For problems with a larger number of variables, we set PS_1 to 100 or above. The second population size, PS_2, was defined based on the first population size. For $PS_1 \geq 50$, we set $PS_2 = 10$, and for $PS_1 < 50$, PS_2 is set to 5.

The values of K_1 and K_2 also were chosen experimentally, but were based on the values of PS_1 and PS_2, respectively. K_1 was set to a maximum of 10 for larger problems, and for a minimum of 5 for small problems. For $PS_2 = 10$, K_2 was set to 3, and for $PS_2 = 5$, K_2 is 2.

The value for the probability α was kept constant at 0.5 for all problems. The α parameter controls the probability of executing the three steps comparison in the topographical heuristic step. For problems which the feasible area is small compared to the whole domain, increasing the value of α have the effect of reducing the number of local optimum points. However, the C-ITGO algorithm is not too sensitive to specific values of α, and the value of 0.5 seems to be a generally good choice.

The maximum number of elements to execute the local search, $MaxLs$, was also set to a default value for all problems, namely 5. It acts as a filter, selecting only the best solutions to undergo the local search procedure. For problems with a large value of local optima, setting $MaxLs$ to higher values may improve C-ITGO performance. However, if $MaxLs$ is too large, many unnecessary runs of the local search will be executed.

The value of ϕ was set to smaller values for problems defined on entirely continuous domains and assumed higher values for mixed integer problems. If we reduce the space too much for integer problems, we may end up with some dimensions having a very small domain, possibly excluding the global optimum.

The LS_1 and LS_2 control the maximum number of allowed function evaluations in

the local search in the first and second calls, respectively. We tried to set LS_1 as small as possible while attaining convergence to the required precision (which is specific for each problem). LS_2 assumed higher values for more difficult problems, where convergence is achieved only for solutions very close to the global optimum.

Finally, $LSType$ represents the algorithm used in the local search. The type of local search was determined based on the properties of each problem. For problems with only continuous variables, we used the SQP method (WB, TC, and TB). For problems with only discrete variables, we used the NOMAD solver (GT, MD), and for mixed integer problems, we used the BONMIN solver (SRI and PV). The only exception was SRII, in which we used the SQP algorithm rounding the variables that are required to be an integer. In tests with this specific problem, it resulted in better solutions than using the mixed integer solvers.

3. Computational Results

The C-ITGO has been implemented using Matlab, and the experiments were accomplished on a machine with the Intel i3-3110M CPU @2.40GHz processor with 4GB of RAM, running Windows 7. Also, we provide a free library for using C-ITGO for optimizing any user-defined function, being it constrained or not. The library uses by default the Matlab *fmincon* solver for continuous problems and the OPTI toolbox (currently only readily available for Windows, but can be compiled for Linux and Mac as well) for mixed integer problems. Also, the user has the option to easily incorporate other local search algorithms, if desired.

To evaluate the performance of the developed method to solve real-world problems, we use eight difficult constrained engineering optimization problems from the literature, whose objective functions and constraints are diverse (quadratic, cubic, polynomial and nonlinear) with many numbers and types of design variables (continuous, mixed and integer).

In all tests, given the stochastic characteristic of C-ITGO, we run it 25 times, saving the Best and Worst feasible solutions, as well as the mean value (Mean) and standard deviation (SD) of the fitness after all runs. Given the great variability between the results found in the literature for most problems, we stop the execution of C-ITGO as soon as the best solution found in a run is considerably close to the optimum (the optimum solution for each of the eight engineering design problems considered is

	WB	TC	TB	SRI	SRII	PV	GT	MD
GAP	1E-6	1E-6	1E-5	1E-8	1E-7	1E-4	1E-10	1E-5

Table 2: Minimum distance to optimum to determine C-ITGO convergence for each problem.

known). That is, the condition for stopping C-ITGO, when the fitness reaches a certain value, is problem dependent. At the end, the mean number of function evaluations (MNFEs) is reported for all runs.

Table 2 shows the values that determine the convergence of C-ITGO for each problem. The values in row *GAP* represent the minimum distance to the optimum that the fitness of a solution needs to be to stop the execution of C-ITGO. The Gear Train (GT) was the only problem where we specified a maximum number of function evaluations. For this problem, if the NFEs reaches 800, we stop immediately and return the best solution found. All other problems converged to at least *GAP* of the optimum without the requirement to specify a maximum NFEs.

In our tests, all solutions reported by C-ITGO are completely feasible for all problems. So, unless specified, we exclude from comparison methods that violate any constraint (infeasible).

We compare 20 different optimization methods against C-ITGO. The most common meta-heuristic for solving the problems considered in this work is the Particle Swarm Optimization (PSO), including PSO-DE (Lio et al., 2010), a hybrid PSO method combined with Differential Evolution (DE); the HPSO (He & Wang, 2007b), another hybridized particle swarm method in combination with simulated annealing; a co-evolutionary PSO denominated CPSO (He & Wang, 2007a); the hybridized PSO with Nelder-Mead simplex (NM-PSO) (Zahara & Kao, 2009); the Unified PSO (UPSO) (Parsopoulos & Vrahatis, 2005), a PSO variant that balances exploration and exploitation; the APSO, standing for Accelerated PSO (Yang, 2010); the Gaussian Quantum-Behaved Particle Swarm Optimization methods, named QPSO and G-QPSO (Coelho, 2010); and the IPSO (Machado-Coelho et al., 2017), which incorporates an interval reducing procedure. More recently, and among the best PSO methods for constrained global optimization, we can cite the Improved Accelerated PSO algorithm (IAPSO) (Guedria, 2016), which proposes some modifications and improves the APSO method.

Many other different optimization methods were also used for comparison in this work: the Mine Blast Algorithm (MBA) (Sadollah et al., 2013), a population-based

optimization method based on the mine bomb explosion concept; the League Championship Algorithm (LCA) (Kashan, 2011), which models a league championship environment with artificial teams; the Crossover-Based Artificial Bee Colony (CB-ABC) (Brajevic, 2015), applying modified search operators over the regular ABC algorithm; the Differential Evolution with Level Comparison (DELC) (Wang & Li, 2009), which converts a constrained problem into an unconstrained one by means of a level comparison mechanism; a Multi-View Differential Evolution (MVDE) (Melo & Carosio, 2013), which uses several different mutation strategies at every iteration; the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) method (Melo & Iacca, 2014), which builds a distribution model of the population; the Water Cycle Algorithm (WCA) (Eskandar et al., 2012), a nature-inspired method based on the water cycle process; and the Cuckoo Search algorithm (CS) (Gandomi et al., 2013), another nature-inspired method based on the cuckoo bird species behavior. Recently, reporting state-of-the-art results for some problems, we can cite the Improved Artificial Bee Colony with Modified Augmented Lagrangian (IABC-MAL) (Long et al., 2017), which integrates the Modified Augmented Lagrangian method to handle constraints with the Improved ABC algorithm (Liang et al., 2017), and the Self-Adaptive Multi-Population based Jaya (SAMP-Jaya) algorithm (Rao & Saroj, 2017), a multi-population scheme of the Jaya method (Rao, 2016).

Following will be briefly explained the engineering optimization problems that will be tackled in this work as well as the solutions obtained by C-ITGO against the solutions of the best previously cited competing techniques of the literature. For the sake of space, the mathematical formulations, the project variables, and their specific values will not be presented here. Further information on the Welded Beam (WB), Tension/Compression Spring (TC), Speed Reducer (SRI and SRII), Pressure Vessel (PV), Gear Train (GT) and Multiple Disk clutch brake (MD) problems can be found in the work of Guedria (2016). Three-Bar truss (TB) and additional information on Welded Beam (WB), Tension/Compression Spring (TC), Speed Reducer (SRI), Pressure Vessel (PV) and Gear Train (GT) can be seen in the work of Sadollah et al. (2013).

3.1. Welded beam design problem

The welded beam design (Ragsdell & Phillips, 1976) is the problem of minimizing the fabrication cost of a welded beam, subject to seven inequality constraints, being two linear and five nonlinear. The constraints include shear stress, bending stress in

the beam, buckling load on the bar and end deflection on the beam. The four design variables are continuous. Figure 4 presents the schematic of the welded beam design problem. We included in the appendix the full statement of the problem, for sake of illustration.

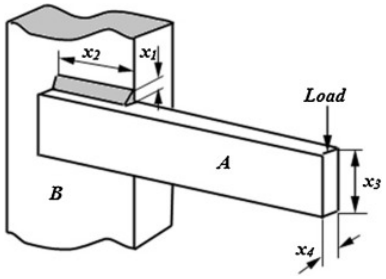


Figure 4: Schematic view of the welded beam design problem.

We compare the results obtained by C-ITGO against a number of state-of-the-art methods used to solve this problem, including PSO-DE, HPSO, UPSO, MBA, CMA-ES, MVDE, CPSO, LCA, IPSO, CB-ABC, DELC, IABC-MAL, NM-PSO, APSO, WCA, IAPSO and SAMP-Jaya. Table 3 shows the comparison of the statistical results obtained by all cited methods to solve the welded beam design problem. All the methods, with exception of CMA-ES, SAMP-Jaya and C-ITGO, took more than 10,000 iterations to achieve good quality solutions. C-ITGO achieved optimal solutions with a very small standard deviation of 3.65E-12, using ten times fewer function evaluations on average than most of the methods.

We write the statistics for the C-ITGO method in bold simply to highlight the reported results, not because it always presents the best statistics for all problems.

The results for C-ITGO and SAMP-Jaya are very similar, with the former having a slightly worse standard deviation, but converging in less than one-third of the number of function evaluations. We note here that the best solution achieved by NM-PSO is slightly infeasible, so it should not be directly compared to the other methods.

3.2. Tension/compression spring design problem

This problem was introduced by Belegundu (1982), and the objective is the minimization of the weight of a tension/compression spring (Figure 5). The problem has three continuous design variables, being them the diameter of spring wire, the diame-

Method	Best	Mean	Worst	SD	MNFEs
PSO-DE	1.724852	1.724852	1.724852	6.70E-16	66,600
HPSO	1.724852	1.814295	1.749040	4.01E-02	81,000
UPSO	1.921990	2.837210	N/A	6.83E-01	100,000
MBA	1.724853	1.724853	1.724853	6.94E-19	47,340
CMA-ES	1.7248523	1.7248523	1.7248523	1.66E-09	4,658
MVDE	1.7248527	1.7248621	1.7249215	7.88E-06	15,000
CPSO	1.728024	1.748831	1.782143	1.29E-02	240,000
LCA	1.7248523	1.7248523	1.7248523	7.11E-15	15,000
IPSO	1.724852	1.7248787	1.7250002	3.28E-05	20,000
CB-ABC	1.724852	1.724852	N/A	2.38E-11	15,000
DELC	1.724852	1.724852	1.724852	4.10E-13	20,000
IABC-MAL	1.724852	1.724852	1.724852	2.31E-12	15,000
NM-PSO	1.724717	1.726373	1.733393	3.50E-03	80,000
APSO	1.736193	1.877851	1.993999	0.076118	50,000
WCA	1.724856	1.726427	1.744697	4.29E-03	46,450
IAPSO	1.7248523	1.7248528	1.7248624	2.02E-06	12,500
SAMP-Jaya	1.724852	1.724852	1.724852	6.7E-16	3,618.25
C-ITGO	**1.7248523**	**1.7248523**	**1.7248523**	**3.65E-12**	**940.68**

Table 3: Statistical results of different methods for the Welded beam problem.

ter of spring mean coil and the number of active coils. It is subject to three nonlinear and one linear inequality constraints.

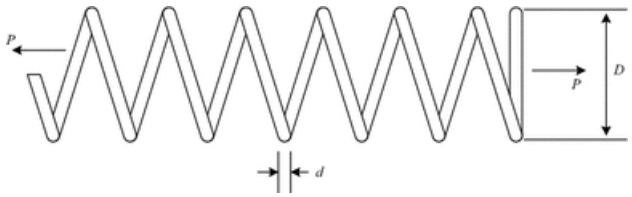


Figure 5: Schematic view of the tension/compression spring design problem.

A variety of methods were used in literature to solve the tension/compression spring design problem. Between them, we can cite CPSO, HPSO, NM-PSO, MBA, UPSO, PSO-DE, LCA, CB-ABC, QPSO, G-QPSO, APSO, CMA-ES, MVDE, DELC, WCA, IPSO, IAPSO, IABC-MAL and SAMP-Jaya. Table 4 shows the statistical results of all methods, along with the number of function evaluations taken to achieve such results.

The methods vary greatly in the number of function evaluations required to converge to good quality solutions. Only three of the methods (QPSO, G-QPSO and IAPSO) achieved convergence with 2,000 function evaluations, while some methods required more than 100,000. The best solution achieved by all methods have the same fitness value up to five decimal places, with the exception of CPSO, UPSO, APSO and NM-PSO.

The best solution reported by NM-PSO is again infeasible, so we do not compare it against other methods. The SAMP-Jaya method also seems to report a slightly infeasible best solution, differing from the optimal value reported by other methods.

Method	Best	Mean	Worst	SD	MNFEs
CPSO	0.0126747	0.012730	0.012924	5.20E-04	240,000
HPSO	0.012665	0.012719	0.012707	1.58E-05	81,000
NM-PSO	0.012630	0.012633	0.012631	8.47E-07	80,000
MBA	0.012665	0.012713	0.012900	6.30E-05	7,650
UPSO	0.013120	0.022940	N/A	7.20E-03	100,000
PSO-DE	0.012665	0.012665	0.012665	1.20E-08	24,950
LCA	0.01266523	0.01266541	0.01266667	3.88E-07	15,000
CB-ABC	0.012665	0.012671	N/A	1.42E-05	15,000
QPSO	0.012669	0.018127	0.013854	1.34E-03	2,000
G-QPSO	0.012665	0.017759	0.013524	1.27E-03	2,000
APSO	0.012700	0.013297	0.014937	6.85E-04	120,000
CMA-ES	0.01266524	0.01266861	0.01269335	6.30E-06	19,445
MVDE	0.01266527	0.01266732	0.01271906	2.45E-06	10,000
DELC	0.012665	0.012666	0.012665	1.30E-07	20,000
WCA	0.012665	0.012746	0.012952	8.06E-05	11,750
IPSO	0.01266567	0.012671486	0.01268312	3.89E-06	20000
IAPSO	0.01266523	0.013676527	0.01782864	1.573E-3	2,000
IABC-MAL	0.01266523	0.01266525	0.01266539	6.78E-08	15,000
SAMP-Jaya	0.012664	0.013193	0.012714	9.25E-05	6,861
C-ITGO	**0.01266523**	**0.01266523**	**0.01266525**	**2.81E-9**	**535.08**

Table 4: Statistical results of different methods for tension/compression spring design problem.

However, the result achieved by SAMP-Jaya differs only at the sixth decimal place, and the difference may be due to wrong rounding. We cannot state this for sure since we do not have access to the solution vector found by the method. Given the differences between the precision in the solutions reported by the methods, we consider $f(\boldsymbol{x}) = 0.012665$ to be the fitness at the global optimum.

The C-ITGO method performed remarkably well on this problem, converging to the best solution found in 535.08 function evaluations on average, almost four times less than the second best competing method. Also, the standard deviation is very small, in the order of 2.81E-9.

Given the relatively small domain of the problem, we believe that the C-ITGO method was able to cover the space efficiently, providing hot starting points for the local search method, which proved to be very effective in this case.

3.3. Three-bar truss design problem

The three-bar truss design problem (Ragsdell & Phillips, 1973) consists in minimizing the volume of a statically loaded three-bar truss. There are two continuous design variables and three nonlinear inequality constraints, based on the stress on each of the truss members. Figure 6 shows the schematic view of the problem.

Here we present only the five best-performing methods used to solve this problem since the solutions found in the literature have very small variance from each other. These methods are PSO-DE, CMA-ES, MVDE, DELC, MBA and IABC-MAL. We present the statistical results for this problem in Table 5.

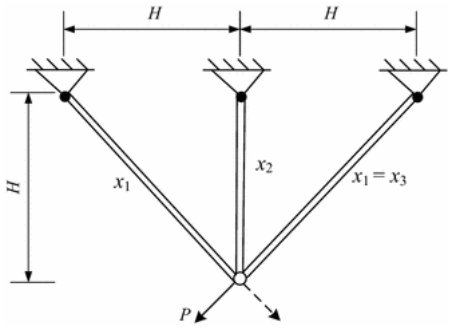


Figure 6: Schematic view of the three-bar truss design problem.

Method	Best	Mean	Worst	SD	MNFEs
PSO-DE	263.895843	263.895843	263.895843	4.50E-12	17,600
CMA-ES	263.895843	263.895843	263.895843	2.7E-09	1,706
MVDE	263.895843	263.895843	263.895855	2.58E-07	7,000
DELC	263.895843	263.895843	263.895843	4.3E-14	10,000
MBA	263.895852	263.897996	263.915983	3.93E-03	13,280
IABC-MAL	263.895843	263.895843	263.895843	0.0	15,000
C-ITGO	**263.895843**	**263.895843**	**263.895843**	**2.0E-12**	**136.48**

Table 5: Statistical results of different methods for three-bar truss design problem.

From Table 5 it is possible to note that all methods have very similar results, differing mainly in the number of function calls. This is not surprising, given that the problem is simpler, has fewer variables and consequently has a smaller domain than any other engineering design problem considered in this work.

C-ITGO achieved convergence quickly, taking 136.48 function evaluations on average. In this problem, we have set a small population, as well as a small number of function calls allowed in the local search procedure. Thus, we obtained the lowest MNFEs than any of the competing techniques, at least ten times lower. It seems that the solutions found by the topographical heuristic were already close to the optimum, so the local search converged in very few iterations. However, the C-ITGO method has a slightly worse standard deviation than DELC and IABC-MAL.

3.4. Speed reducer design problem

In this problem, the total weight of the speed reducer (Figure 7) is to be minimized. The problem has seven design variables: face width, teeth module, number of teeth on the pinion, first and second length of shafts between bearings and first and second shafts diameter (Golinski, 1973). The problem has 11 nonlinear constraints, and the third variable is constrained to be an integer. We use two versions of this problem,

Method	Best	Mean	Worst	SD	MNFEs
LCA	2996.34816497	2996.34816497	2996.34816497	2.63E-12	24,000
IPSO	2996.34816497	2996.3481	2996.34816509	2.43E-18	20,000
APSO	3177.530771	3855.581557	4677.005187	473.767	30,000
IAPSO	2996.34816497	2996.34816497	2996.34816497	6,88E-13	6,000
C-ITGO	**2996.34816497**	**2996.34816497**	**2996.34816497**	**7.5E-13**	**856.40**

Table 6: Statistical results of different methods for the speed reducer design problem I.

where the only difference is in the bounds of the fifth variable. We will call these problems SRI and SRII, respectively.

For SRI, we compare C-ITGO results against four optimization methods, namely LCA, IPSO, APSO and IAPSO, shown in Table 6.

The LCA, IPSO, IAPSO and C-ITGO present similar results, all finding the constrained global optimum solution and having very low standard deviation.

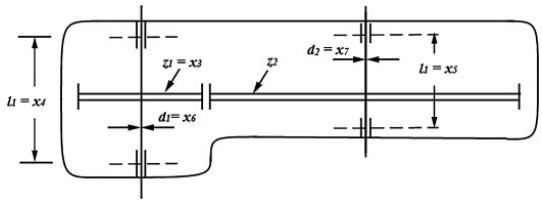


Figure 7: Schematic view of the speed reducer design problem.

The first three methods take at least 20,000 iterations to converge to good quality solutions, while IAPSO takes only 6,000 NFEs. C-ITGO outperforms the other methods by a large margin, converging in 858.40 function calls on average while maintaining negligible higher standard deviation than IAPSO and IPSO.

For this problem, we used considerably greater populations, of sizes 150 and 10, while maintaining the number of allowed function evaluations of the local search relatively small (100 and 200). For the worst case, C-ITGO took 3 full iterations to converge, while in most cases it converged in the first iteration.

For SRII, the methods used for comparison were PSO-DE, WCA, DELC, CB-ABC, CMA-ES, MVDE, IABC-MAL, LCA, APSO, MBA, IPSO, IAPSO and SAMP-Jaya. The reported results for the SAMP-Jaya method seems to violate the integer constraint at variable three, but it is included to show that even in harder situations the C-ITGO algorithm can still produce state-of-the-art results.

From Table 7, we can see that DELC, CMA-ES, MVDE, IABC-MAL, LCA, IAPSO, SAMP-Jaya and C-ITGO are the only methods that converged to the optimum in ev-

Method	Best	Mean	Worst	SD	MNFEs
PSO-DE	2996.348167	2996.348174	2996.348204	6.40E-06	54,350
WCA	2994.471066	2994.474392	2994.505578	7.4E-03	15,150
DELC	2994.471066	2994.471066	2994.471066	1.90E-12	30,000
CB-ABC	2994.471066	2994.471066	N/A	2.48E-07	15,000
CMA-ES	2994.471066	2994.471066	2994.471066	8.98E-10	12,998
MVDE	2994.471066	2994.471066	2994.471066	2.82E-07	30,000
IABC-MAL	2994.471066	2994.471066	2994.471066	8.51E-13	15,000
LCA	2994.471066	2994.471066	2994.471066	2.66E-12	24000
APSO	3187.630486	3822.640624	4443.017639	366.146	30,000
MBA	2994.482453	2996.769019	2999.652444	1.56	6300
IPSO	2994.471067	2994.47108	2994.4711	9.27E-06	20,000
IAPSO	2994.471066	2994.471066	2994.471066	2.65E-09	6,000
SAMP-Jaya	2760.673988	2760.673988	2760.673988	2.54E-11	3744.66
C-ITGO	**2994.471066**	**2994.471066**	**2994.471066**	**9.28E-13**	**491.24**

Table 7: Statistical results of different methods for the speed reducer design problem II.

ery run, having very small standard deviation. Although similar results are observed regarding the quality of the solutions, C-ITGO converges much faster than any competing method, using seven times fewer function evaluations than SAMP-Jaya. The standard deviation achieved by C-ITGO is also only worse than the standard deviation reported by IABC-MAL (the difference is in the order of 1E-13), which took 15,000 function evaluations to converge.

For this specific problem, we used the SQP algorithm as local search and rounded the value of the third variable. In this case, this approach proved to converge much faster than using a mixed integer local search. Also, the first population size is considerably smaller than the one used in SRI (100 in this problem).

3.5. Pressure vessel design problem

In the pressure vessel design problem, the objective is to minimize the total manufacturing cost, including the cost of the material, forming and welding costs (Sandgren, 1990). The problem is subject to three linear and one nonlinear inequality constraint, and its structure is shown in Figure 8. The four variables are the thickness of the shell, the thickness of the head, the inner radius, and the length of the cylindrical section of the vessel. This is an example of a mixed integer problem, where the first and second variables are constrained to be integers.

We compare C-ITGO against the UPSO, QPSO, G-QPSO, CMA-ES, MVDE, CB-ABC, PSO-DE, HPSO, WCA, LCA, APSO, CPSO, MBA, IPSO, DELC, IABC-MAL, SAMP-Jaya and IAPSO methods. The statistical results are shown in Table 8. Some of the methods used to solve this problem report unfeasible results (PSO-DE, WCA, MBA and SAMP-Jaya), probably either because the integer constraints are ignored or due to wrong rounding. Nevertheless, we still use the results of these methods to prove the superior

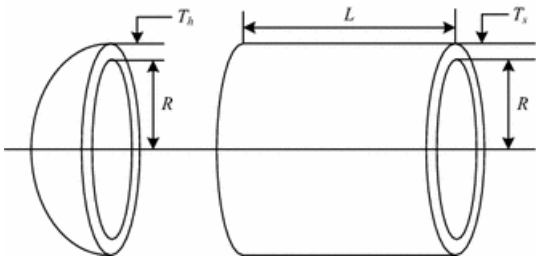


Figure 8: Schematic view of pressure vessel design problem.

Method	Best	Mean	Worst	SD	MNFEs
UPSO	6544.2700	9032.5500	N/A	9.95E+02	100,000
QPSO	6059.7209	8017.2816	6440.3786	4.79E+02	8,000
G-QPSO	6059.7208	7544.4925	6440.3786	4.48E+02	8,000
CMA-ES	6059.7143	6170.25055	6410.08676	140.4843	30,018
MVDE	6059.7144	6059.99724	6090.53353	2.9103	15,000
CB-ABC	6059.7143	6126.6237	N/A	1.14E+02	15,000
PSO-DE	6059.7140	6059.7140	N/A	N/A	42,100
HPSO	6059.7143	6099.9323	6288.6770	86.2000	81,000
WCA	5885.3327	6198.6172	6590.2129	213.0490	27,500
LCA	6059.8553	6070.5884	6090.6114	11.37534	24,000
APSO	6059.7242	6470.7156	7544.4927	326.9688	200,000
CPSO	6061.0777	6147.1332	6363.8041	86.4500	240,000
MBA	5889.3216	6200.64765	6392.5062	160.34	70,650
IPSO	6059.7143	6059.7155	6059.7257	0.00232	20,000
DELC	6059.7143	6059.7143	6059.7143	2.10E-11	30,000
IABC-MAL	6059.7143	6072.5972	6089.2720	1.88E-06	15,000
SAMP-Jaya	5872.2129	5872.2129	5872.2129	5.0E-12	6513.33
IAPSO	6059.7143	6068.7539	6090.5314	14.0057	7,500
C-ITGO	**6059.7143**	**6059.7143**	**6059.7143**	**9.8E-13**	**1101.64**

Table 8: Statistical results of different methods for the pressure vessel design problem.

convergence of C-ITGO, even in constrained mixed integer problems.

The algorithms CMA-ES, CB-ABC, HPSO, IABC-MAl, IPSO, DELC, IAPSO and C-ITGO are the only who found the feasible optimal solution. All other methods presented relatively poor performance, with much higher NFEs on average. C-ITGO takes less than five times the number of function evaluations to converge than the best competing method, SAMP-Jaya, which reported a result of 6513.33 MNFEs. Also, C-ITGO achieves the smallest standard deviation among all methods. All runs of C-ITGO converged quickly to the known global optimum, showing unarguably better results than any other method compared in this problem.

3.6. Gear train design problem

In this problem, the objective is to minimize the cost of the gear ratio of the gear train (Sandgren, 1990). An example is shown in Figure 9. The problem has four variables, representing the number of teeth for the four gears. The only constraint of

Method	Best	Mean	Worst	SD	MNFEs
MBA	2.700857E-12	2.062904E-08	2.47E-09	3.94E-09	1,120
UPSO	2.700857E-12	3.80562E-08	N/A	1.09E-07	100,000
CS	2.7009E-12	1.9841E-9	2.3576E-9	3.55E-9	5,000
APSO	2.700857E-12	4.781676E-07	7.072678E-06	1.44E-06	8,000
IAPSO	2.700857E-12	5.492477E-09	1.827380E-08	6.36E-09	800
C-ITGO	**2.700857E-12**	**4.6504232E-09**	**2.7264505E-08**	**6.85E-09**	**773.0**

Table 9: Statistical results of different methods for the gear train design problem.

the problem is that all variables must be integers.

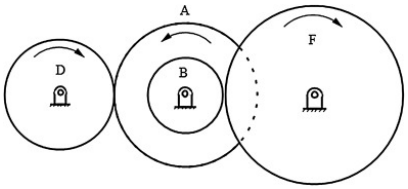


Figure 9: Gear train design problem structure.

We compare the C-ITGO results, shown in Table 9, against other five methods: MBA, UPSO, CS, APSO and IAPSO. In general, all methods compared achieved similar results, having found the global optimal solution for the problem. The main difference relies on the number of function calls, varying from 100,00 for UPSO to 800 for IAPSO.

Since this is an integer optimization problem, we used the mixed integer local search based on the NOMAD solver. In this problem, the solver had some difficulty in some runs, achieving 6.85E-09 of standard deviation, more than IAPSO, CS and MBA. The mean fitness value, however, is just worse than the CS method, which takes 5,000 function calls to converge. Besides this drawbacks, C-ITGO was the fastest method to achieve convergence, taking only 773.0 function calls on average.

3.7. Multiple disk clutch brake design problem

This is also an example of a problem where all variables are discrete. The Multiple disk clutch brake design problem aims at minimizing the mass of a multiple disk clutch brake Osyczka (2002). There are five integer variables: the inner radius, outer radius, thickness of the disk, actuating force and number of friction surfaces (Figure 10). The problem is also constrained by nine nonlinear inequalities.

To compare the results, we used only methods who achieved close performance to the obtained by C-ITGO, since some methods in literature have very diverging results. The methods compared are WCA, IPSO, APSO and IAPSO. The WCA, IPSO, and IAPSO

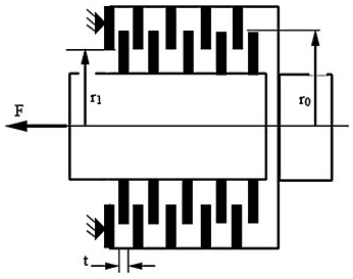


Figure 10: Schematic view of the multiple disk clutch brake design problem.

Method	Best	Mean	Worst	SD	MNFEs
WCA	0.313656	0.313656	0.313656	1.69E-16	500
IPSO	0.313656	0.313656	0.313656	0.0	20,000
APSO	0.337181	0.506829	0.716313	0.09767	2,000
IAPSO	0.313656	0.313656	0.313656	1.13E-16	400
C-ITGO	**0.313656**	**0.313656**	**0.313656**	**1.13E-16**	**286.48**

Table 10: Statistical results of different methods for the multiple disk clutch break design problem.

methods achieved good performance in this problem, finding the global optimal solution and having very small variance, as shown in Table 10. The WCA and IAPSO took very few function evaluations to converge, respectively 500 and 400. IPSO took long more, with 20,000 function evaluations. However, the reported standard deviation for IPSO was 0.0.

The C-ITGO clearly outperforms the other methods, having a standard deviation of 1.13E-16, while converging with only 286.48 function evaluations on average. For this problem, we used very small populations, of sizes 20 and 5. The number of function evaluations in the first step was 100, and most runs converged in much less, enjoying the quality of initial solutions provided by the topographical heuristic step. Some runs, however, took more than 1,000 function evaluations, bringing the mean NFEs up.

3.8. Best Solutions to the Engineering Problems

Table 11 shows the fitness value, the values of the variables and the values of the constraints for the best solution found by C-ITGO for all problems. The precision of the results of each problem was set to match the results reported by the competing methods cited in this work. However, to the best of our knowledge, the best solution found by C-ITGO for each problem is equal to, or negligibly worse (up to several decimal places) than the best solutions reported by state-of-the-art methods.

Prob.	WB	TC	TB	SRI	SRII	PV	GT	MD
$f(\cdot)$	1.7248523	0.01266523	263.895843	2996.34816497	2994.471066	6059.7143	2.700857E-12	0.313656
$\boldsymbol{x}_1$	0.2057296	0.05168906	0.788675	3.5	3.5	13.0	43	70
$\boldsymbol{x}_2$	3.4704886	0.35671774	0.408248	7.0	0.7	7.0	16	90
$\boldsymbol{x}_3$	9.0366239	11.28896574		17	17	41.5984	19	1
$\boldsymbol{x}_4$	0.2057296			7.3	7.3	176.6366	49	830
$\boldsymbol{x}_5$				7.8	7.715320			3
$\boldsymbol{x}_6$				3.35021467	3.350215			
$\boldsymbol{x}_7$				5.28668323	5.286654			
$\boldsymbol{g}_1$	0.0	0.0	0.0	-0.07391528	-0.073915	-2.6645E-15		0.0
$\boldsymbol{g}_2$	0.0	0.0	-1.464102	-0.19799853	-0.197999	-0.0359		-24.0
$\boldsymbol{g}_3$	0.0	-4.05378563	-0.535898	-0.49917225	-0.499172	-4.4238E-09		-0.917438
$\boldsymbol{g}_4$	-3.4329838	-1.09159320		-0.90147170	-0.904644	-63.3634		-9.826183
$\boldsymbol{g}_5$	-0.0807296			-8.846257E-13	-2.220446E-16			-7.894697
$\boldsymbol{g}_6$	-0.2355403			-4.885758E-12	0.0			-0.173855
$\boldsymbol{g}_7$	-1.818989E-12			-0.7025	-0.7025			-40.118750
$\boldsymbol{g}_8$				-1.887379E-15	0.0			-14.826145
$\boldsymbol{g}_9$				-0.58333333	-0.583333			
$\boldsymbol{g}_{10}$				-0.05132575	-0.051326			
$\boldsymbol{g}_{11}$				-0.01085237	0.0			

Table 11: Results for the best solution found by C-ITGO for each engineering design problem.

3.9. Statistical Tests To Analyse the Computational Results

We now prove the significant improvement of C-ITGO over the competing methods by means of a nonparametric statistical test, where no assumptions are made regarding the underlying distribution of the data. Since no other algorithm than C-ITGO solves all the eight engineering design problems presented in this paper, we cannot apply typical statistical methods, such as the Friedman test (Derrac et al., 2011).

Instead, we use the Skillings-Mack test (Skillings & Mack, 1981), which is a Friedman-type statistical test that can be used when there are missing data, and that reduces to the Friedman test when the data has no missing entries. Just as in the Friedman test, the Skillings-Mack test finds the rank of the competing algorithms for each problem and then calculates the mean or average rank. The lower the mean rank, the better is the performance of the algorithm. The Skillings-Mack statistical test is also useful when there are many ties or equal ranks, as well as for small samples.

The method reports a *p-value*, the probability that, when the null hypothesis is assumed to be true (in this case, no difference between the performance of the optimization methods), the results observed by the experiments at hand are at least of the same magnitude that the true values that would be observed in the limit of an infinite number of samples. Thus, a small p-value, say less than 0.05, represents a high chance that the null hypothesis is false.

In this work, the mean number of function evaluations (MNFEs) is used as the comparing metric. We used the *Skillings.Mack* package (Srisuradetchai, 2015) of the *R* language (R Dev. Core Team, 2008) to report the following results.

In Table 12 we show the results of applying the statistical test to all the methods that solved at least three of the eight engineering design problems considered in this work. The mean rank of C-ITGO is 1.0, given that it has the smallest MNFEs for all problems. That is much less than IAPSO and SAMP-Jaya, which have a mean rank of 2.75. The mean ranks are also shown in the form of a bar plot for all methods in Figure 11.

Algorithms	**Skillings-Mack ranks**
C-ITGO	1.0
IAPSO	2.75
SAMP-Jaya	2.75
IABC-MAL	5.125
PSO-DE	8.625
HPSO	7.9375
MBA	5.875
CPSO	8.9375
LCA	5.8125
CB-ABC	4.875
DELC	7.0
CMA-ES	5.0
MVDE	5.0625
IPSO	6.5
WCA	5.875
UPSO	8.5625
APSO	9.25
Skillings-Mack Statistic	63.66813949
p-value	1.2469E-07
χ^2	26.296

Table 12: Skillings-Mack test for methods solving at least three problems.

As shown in Table 12, the value of the Skillings-Mack statistic is 63.66813949 with an approximate p-value of 1.2469E-07 calculated from the chi-squared distribution with 16 degrees of freedom, which strongly implies that the null hypothesis does not hold true at the critical level of $\alpha = 0.05$ or $\alpha = 0.01$. That is, there is at least one method that is significantly better than the others.

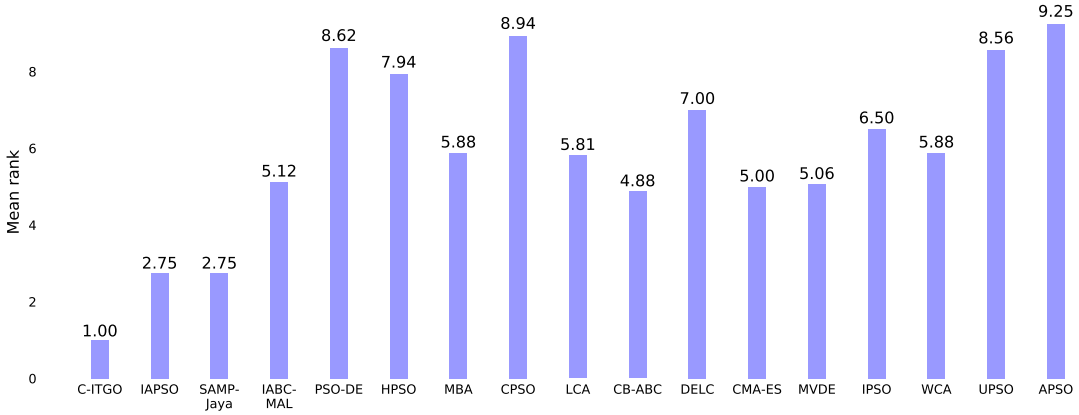


Figure 11: Mean rank plot for all methods that solve at least three problems.

We also show in Table 13 the results of applying the Skillings-Mack test to the methods that solved at least five problems. For this case, the value of the Skillings-Mack statistic is 44.23462786 with an approximate p-value of 7E-06 calculated from the chi-squared distribution with 11 degrees of freedom. The difference between the results reported in Table 12 is mainly due to the reduction of the number of methods, and the exclusion of some methods that performed well, but in four or fewer problems, such as SAMP-Jaya.

We see again that C-ITGO has a mean rank of 1.0, far less than IAPSO with 2.375, standing in second place. The plot for the mean rank is shown in Figure 12. The p-value (7E-06) for this case is also very small, rejecting the null hypothesis at critical level of $\alpha = 0.05$ or $\alpha = 0.01$.

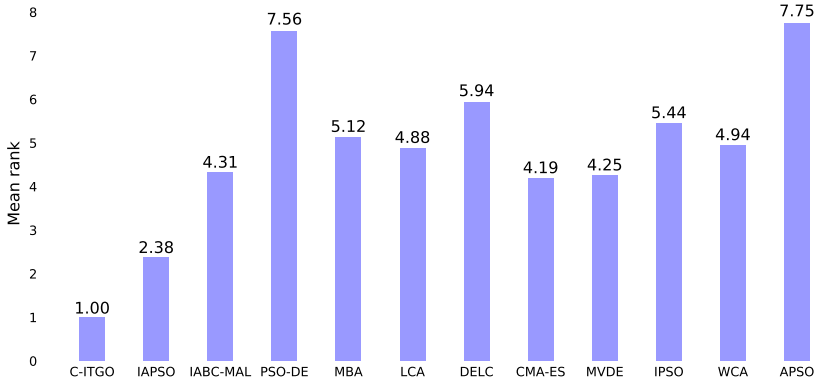


Figure 12: Mean rank plot for all methods that solve at least five problems.

Algorithms	Skillings-Mack ranks
C-ITGO	1.0
IAPSO	2.375
IABC-MAL	4.3125
PSO-DE	7.5625
MBA	5.125
LCA	4.875
DELC	5.9375
CMA-ES	4.1875
MVDE	4.25
IPSO	5.4375
WCA	4.9375
APSO	7.75
Skillings-Mack Statistic	44.23462786
p-value	7E-06
χ^2	19.67513757

Table 13: Skillings-Mack test for methods solving at least five problems.

We also present in Table 14 the results of applying the pairwise signed test, comparing C-ITGO against the same methods presented in Table 13 and considering the same performance metric. Here, a "+" represents that C-ITGO outperforms the corresponding method, a "=" represents a tie, and a "-" represents a loss (C-ITGO has worse performance than the corresponding method). If a method does not solve a certain problem, that entry is ignored (marked with "*"). The last row shows the final sum of the number of "+", "=" and "-" for all methods.

As we can see from Table 14, C-ITGO always has a better MNFEs than any method for all problems, so there are only "+" entries. Following the guidelines presented in Derrac et al. (2011), we can conclude that C-ITGO always achieves a p-value of at least 0.05 when compared to the methods of Table 14.

To prove the superior convergence of C-ITGO, we present one more nonparametric statistical test, the Wilcoxon signed rank test, also described in Derrac et al. (2011). This is a more robust approach to a pairwise statistical comparison than the previous test, that aims to detect a significant difference between the means of two compet-

Eng. Prob.	C-ITGO	IAPSO	IABC-MAL	PSO-DE	MBA	LCA	DELC	CMA-ES	MVDE	IPSO	WCA	APSO
WB	**940.68**	12,500 +	15,000 +	66,000 +	47,340 +	15,000 +	20,000 +	4,658 +	15,000 +	20,000 +	46,450 +	50,000 +
TC	**535.08**	2,000 +	15,000 +	24,950 +	7,650 +	15,000 +	20,000 +	19,445 +	10,000 +	20,000 +	11,750 +	120,000 +
TB	**136.48**	*	15,000 +	17,600 +	13,280 +	*	10,000 +	1,706 +	7,000 +	*	*	*
SRI	**856.40**	6,000 +	*	*	*	24,000 +	*	*	*	20,000 +	*	30,000 +
SRII	**491.24**	6,000 +	15,000 +	54,350 +	6,300 +	24,000 +	30,000 +	12,998 +	30,000 +	20,000 +	15,150 +	30,000 +
PV	**1101.64**	7,500 +	15,000 +	42,100 +	70,650 +	24,000 +	30,000 +	30,018 +	15,000 +	20,000 +	27,500 +	200,000 +
GT	**773.0**	800 +	*	*	1,120 +	*	*	*	*	*	*	8,000 +
MD	**286.48**	400 +	*	*	*	*	*	*	*	20,000 +	500 +	2,000 +
+/=/-		**7/0/0**	**5/0/0**	**5/0/0**	**6/0/0**	**5/0/0**	**5/0/0**	**5/0/0**	**5/0/0**	**6/0/0**	**5/0/0**	**7/0/0**

Table 14: The signed test comparing C-ITGO against the other methods that solved at least five problems. Empty entries are represented by *.

ing techniques. The results shown in Table 15 were generated using the R package *wilcoxon.test,* considering C-ITGO against the other algorithms. Empty entries (problems not solved by some method) are treated by default.

	IAPSO	IABC-MAL	MBA	LCA	CMA-ES	MVDE	APSO
p-value	0.01606	0.001661	0.001207	0.002025	0.002155	0.002129	0.0007229

Table 15: Wilcoxon signed rank test comparing C-ITGO against other methods that solved at least five problems.

Given that C-ITGO always had the best MNFEs when compared to all the other methods, as the Table 15 states, C-ITGO shows a improvement over IABC-MAL, MBA, LCA, CMA-ES, MVDE and APSO with a level of significance $\alpha = 0.01$, and over IAPSO with a level of significance $\alpha = 0.02$.

4. Conclusion

This work presents a modified iterative topographical global optimization algorithm, denominated C-ITGO, specialized in solving constrained optimization problems. The method developed in this paper uses a custom topographical heuristic, based on a stochastic application of the three steps criteria comparison, along with a space reduction procedure. In addition to being very simple conceptually, the C-ITGO algorithm is very generic in the sense that any local search procedure can be used to tackle a specific problem, be it continuous or discrete.

The C-ITGO method was compared against several algorithms found in literature, some presenting state-of-the-art results, in eight difficult constrained engineering optimization problems. In all tests performed, the C-ITGO outperforms any competing method regarding the mean number of function evaluations (MNFEs) to converge to the best-known solution found, with performance gain in some problems being of an order of magnitude over the best performing methods compared.

We also presented several statistical tests comparing C-ITGO against the other methods used in this work. We have shown that C-ITGO outperforms the other competing methods, presenting a better performance that is significantly different at 0.05 level with respect to the MNFEs required to converge to optimal or near-optimal solutions for all the engineering design problems considered.

As future work, we suggest the following possibilities:

- use new, possibly even more complex optimization problems to test the performance of C-ITGO.
- propose an adaptive method to auto adjust the parameters that can influence on C-ITGO performance.
- develop a parallel version aiming at faster execution time, even if this is not the main measure of performance.

Acknowledgments

This work is partially supported by the Brazilian Council for Scientific and Technological Development (CNPq): PIBIC/CNPq, process 163120/2017-8. The development of this research benefited from the UFT Institutional Productivity Research Program

(PROPESQ / UFT). AJSN acknowledges the financial support provided by FAPERJ, CNPq and CAPES, research supporting agencies from Brazil. WFS gratefully acknowledges the financial support provided by CNPq.

References

Appendix

Welded beam design problem

Minimize:

$$f(\boldsymbol{x}) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14 + x_2)$$

subject to:

$$g_1(\boldsymbol{x}) = \tau(\boldsymbol{x}) - 13,600 \leq 0$$

$$g_2(\boldsymbol{x}) = \sigma(\boldsymbol{x}) - 30,000 \leq 0$$

$$g_3(\boldsymbol{x}) = x_1 - x_4 \leq 0$$

$$g_4(\boldsymbol{x}) = 0.10471x_1^2 + 0.04811x_3x_4(14 + x_2) - 5 \leq 0$$

$$g_5(\boldsymbol{x}) = 0.125 - x_1 \leq 0$$

$$g_6(\boldsymbol{x}) = \delta(\boldsymbol{x}) - 0.25 \leq 0$$

$$g_7(\boldsymbol{x}) = 6,000 - Pc(\boldsymbol{x}) \leq 0$$

where:

$$\tau(\boldsymbol{x}) = \sqrt{\tau^2 + \frac{(2\tau'\tau''x_2)}{2R} + \tau''^2}$$

$$\tau' = \frac{6,000}{(sqrt2x_1x_2)}$$

$$\tau'' = \frac{(MR)}{J}$$

$$M = 6,000(14 + \frac{x_2}{2})$$

$$R = sqrt\frac{x_2^2}{4} + (x_1 + \frac{x_3}{2})^2$$

$$J = 2x_1x_2\sqrt{2}(\frac{x_2^2}{12} + (x_1 + \frac{x_3}{2})^2)$$

$$\sigma(\boldsymbol{x}) = 504,000/(x_4x_3^2)$$

$$\delta(\boldsymbol{x}) = \frac{65,856,000}{(30 \times 10^6x_4x_3^3)}$$

$$Pc(\boldsymbol{x}) = \frac{4.013(30 \times 10^6)}{196}\sqrt{x_3^2\frac{x_4^6}{36}}\left(1 - \frac{x_3}{28}\sqrt{\frac{30 \times 10^6}{4(12 \times 10^6)}}\right)$$

with bounds:

$$0.1 \leq x_1, x_4 \leq 2$$

$$0.1 \leq x_2, x_3 \leq 10$$