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 unfeasible. 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 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 im-

[*]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)

possibility of obtaining good quality solutions to some complex problems using deterministic 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 optimum points. Any deterministic 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.

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 in 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 linear or 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:

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

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, know as Iterative Topographical Global Optimization (ITGO) (Törn & Viitanen, 1996). The modifications consist of adding skills to ITGO to work with constrained optimization issues, calling this new algorithm as Contrained version of ITGO (C-ITGO). We compare the results obtained on eight complex constrained engineering design problems used as benchmark against several state of the art methods, as the C-ITGO outperforming the best competing techniques of literature, mainly relative to the number of function evaluations (NFEs) 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 compared 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 meta-heuristic 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 use the Sobol sequences Sobol (1967); Henderson et al. (2015), which has a significantly more uniform distribution

than an excellent pseudo-random number generator. The Fig. 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 Mersene Twister proposed by Matsumoto & Nishimura (1998). The other points (red dots) are the 150 first elements of a Sobol sequence. From Fig 1 is possible to observe that the points generated by Sobol sequence cover the space more uniformly than that of Mersene Twister.

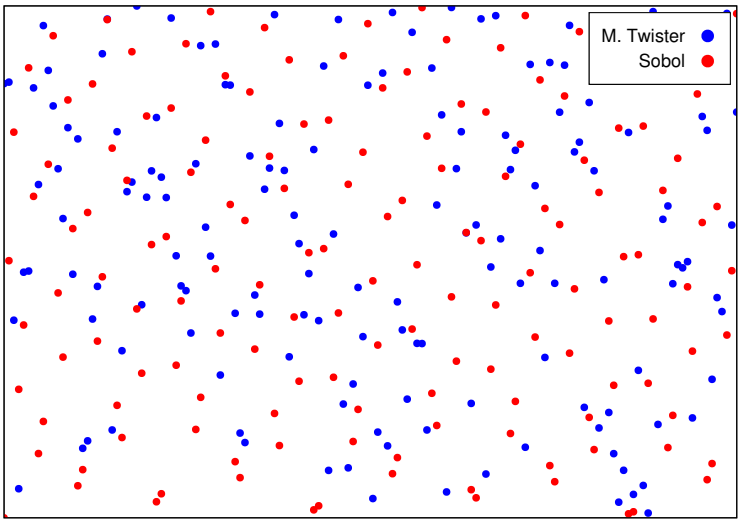


Figure 1: Example showing the distribution of points in the domain $[0, 1]^2$ by the Mersenne Twister generator (blue points) and by a Sobol sequence (red points).

The 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 edges, being some of them incoming and some outgoing. A topography matrix $A \in \mathbb{R}^{M \times M}$ is created based on this graph, defined as:

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

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 on 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 optimum points based on the estimated topography of the function $f(\cdot)$.

As there is no guarantee that the individuals selected are really local optimum points (the optimality relationship is based only on the topography created by the individuals of the population), 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 optimum 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 $f(\cdot)$ defined by $f(x, y) = sen(x^2) + cos(y^2)$, the population P composed by the 10 elements:

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

and the KNN set determined by the $K = 3$ closest neighbors. Fig. 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:

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

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 and no incoming edge, 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 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 Fig. 3, again for the function $f(x, y) = sen(x^2) + cos(y^2)$, in the domain $[-1, 3]^2$, with the first population composed by the same 10 points described previously. The Fig. 3a shows the

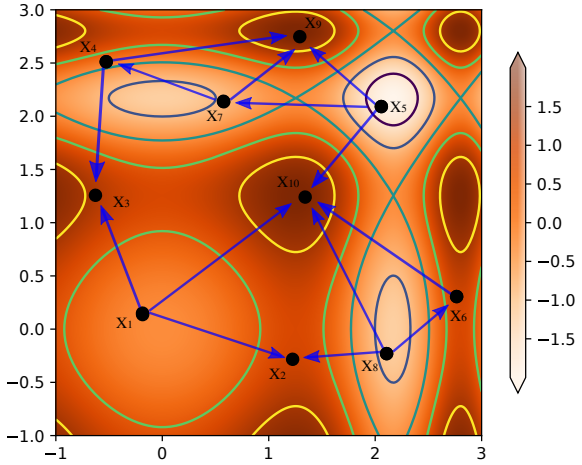


Figure 2: Contour plot of the function $f(x, y) = sen(x^2) + cos(y^2)$, and the graph based on the population.

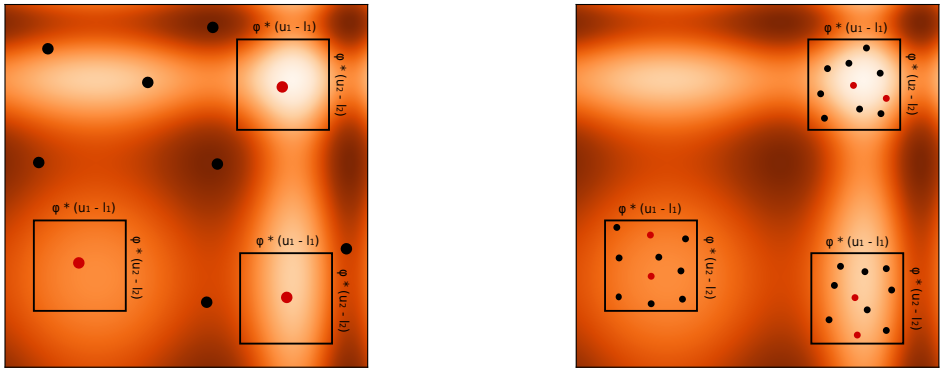
selection of the points $\boldsymbol{x}_1$, $\boldsymbol{x}_5$ and $\boldsymbol{x}_8$, with a space reduced by $\phi = 0.25$. In the Fig. 3b we can observe the generation of populations for each new subspace. The red points are the individuals considered local optimum based on the topographic heuristic. It is worth noting that the local optimum 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.

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 is proposed a constrained version of ITGO, in the rest of this work, it is named as C-ITGO, standing for Constrained ITGO. The method for comparison comprises the three steps criteria:



(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 Fig. 2 ($\boldsymbol{x}_1$, $\boldsymbol{x}_5$ and $\boldsymbol{x}_8$) in red (based on the topographical heuristic). In (b) are the respective populations generated in each restricted space to apply TGO.

- 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 (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 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 optimum 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

point in the topographical heuristic. In the case of none of the individuals are selected as local optimum given a population, we take the best individual according to the three step 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 step 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 local search number of iterations LS_1 and LS_2, the space reduction factor ϕ and the probability for the three step 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)

$\boldsymbol{best} \leftarrow random(\boldsymbol{l}, \boldsymbol{u})$;
while $Converged(\boldsymbol{best}) ==$ **False do**
  $Populations \leftarrow \{\boldsymbol{x} = \boldsymbol{x}^1, ..., \boldsymbol{x}^{\boldsymbol{PS}(1)} : \boldsymbol{x} \in random(\boldsymbol{l}, \boldsymbol{u})\}$;
  $TopoBest \leftarrow \{\}$;
  for $p \in \{1, ..., |\boldsymbol{PS}|\}$ **do**
    $NewPops \leftarrow \{\}$;
    for $pop \in Populations$ **do**
      $Topo_p \leftarrow TopographicalHeuristic(f(\cdot), v(\cdot), pop, \boldsymbol{KS}(p), \alpha)$;
      if $p < |\boldsymbol{PS}|$ **then**
        for $\boldsymbol{x} \in Topo_p$ **do**
          $NewPops \leftarrow NewPops \cup CreatePop(\boldsymbol{x}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{PS}(p+1), \phi^p)$;
      else
        $TopoBest \leftarrow TopoBest \cup Topo_p$;
    if $p < |\boldsymbol{PS}|$ **then**
      $Populations \leftarrow NewPops$;
  $TopoBest \leftarrow \{\boldsymbol{x} = \boldsymbol{x}^1, ..., \boldsymbol{x}^{min(|TopoBest|, Max\_LS)} : \boldsymbol{x} \in sort(TopoBest)\}$;
  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 $\boldsymbol{PS}(1)$ elements, with all 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 $rand$ here to denote the generation of a random scalar or vector for simplicity, although in the implementation we use 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 optimum 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 optimum points in the set $TopoBest$ to apply local search (lines 12 and 13).

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

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

The $CreatePop$ procedure is shown in algorithm 2. Given an individual $\boldsymbol{x}$ (a selected local optimum point), the function randomly generates a new population with $popSize$

individuals around that point, 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.

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 top 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 iterations, at line 18. The function $Compare$ (line 19) is the three step 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, 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 factor of performance. So, we wish to do, as few local search iterations as possible, since a call to the local search for each individual typically makes hundreds or thousands of function evaluations. Here, we set $LS_2 > LS_1$, so every element undergoes LS_1 iterations of local search, but we only search finely for promising solutions, setting larger 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 found element (***best***) using the three step criteria. If the new solution is better, we set ***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 optimum solution is found or when the maximum number of function evaluations is exceeded.

The only thing left is the $TopographicalHeuristic$ procedure, shown in algorithm 3. The function takes as parameter 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 step comparison.

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

1: $M \leftarrow |Population|$;
2: $\boldsymbol{best} \leftarrow Population(0)$;
3: $\boldsymbol{KNN}_K \leftarrow Build\_KNN(Population, K)$;
4: $\boldsymbol{R} \leftarrow random([0,1]^{M \times M})$;
5: $TopoBest \leftarrow \{\}$;
6: **for** $i \in \{1, ..., |Population|\}$ **do**
7: $\quad insert = \boldsymbol{True}$;
8: $\quad$ **for** $j \in \{1, ..., |Population|\} \cap \{\boldsymbol{x}_j \in \boldsymbol{KNN}_K(\boldsymbol{x}_i)\}$ **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 step comparison (initially set to any individual of the population), the $\boldsymbol{KNN}_K$ 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 $\boldsymbol{KNN}_K$ of $\boldsymbol{x}$, the random symmetric matrix R, with every element in the range $[0, 1]$, and the $TopoBest$ set, containing the local optimum 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 optimum point. For every index j, such that $\boldsymbol{x}_j$ is in the set $\boldsymbol{KNN}_K(\boldsymbol{x}_i)$ (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 make 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 selects the best element found in the whole population based on the three step comparison, and stores that solution in the

variable ***best***. In case of no solution is selected as a local optimum, the set $TopoBest$ is composed only of the ***best*** element (lines 17-18). At line 19 we return the $TopoBest$ set, containing all the local optimum points (or the ***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 have 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 infeasible. And even if a problem can be solved by an exact method, the number of function evaluations can 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

$Prob/$ $Param$	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..

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 taken by competing methods to converge 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), and K_1 and K_2 (also before and after space reduction, respectively). The value for the probability α was kept constant for all problems, as well as the maximum number of elements to execute the local search, $MaxLs$. The value of ϕ was set to smaller values for problems defined on entirely continuous domains and assumed higher values for mixed integer problems. 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. Finally, $LSType$ represents the type of local search procedure used.

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. The source code for the reported results in this work can be found and downloaded for free at **?**. 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 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 literature for most problems, we stop the execution of C-ITGO as soon as the best solution found in a run is considerable close to the optimum (the optimum solution for each of the eight engineering design problems considered is known). That is, the condition for stopping C-ITGO, when the fitness reaches a certain value, is problem dependent.

Also, a maximum number of function evaluations (NFEs) was set for each problem. If the value of NFEs exceeds the maximum, the algorithm immediately stops, and the

best solution found is returned. At the end, the mean number of function evaluations (MNFEs) is reported for all runs. This value is also problem dependent, and is based on the results reported by other methods.

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 19 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 state of the art 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 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 a 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) (Melo & Iacca, 2014) method, 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), another nature inspired method based on the cuckoo bird specie behavior. Recently, and reporting state of the art results on 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) (Rao & Saroj, 2017) algorithm, a multi-population scheme of the Jaya (Rao, 2016) method.

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 Welded Beam (WB), Tension / Compression Spring (TC), Speed Reducer (SRI and SRII), Pressure Vessel (PV), Gear Train (GT) and Multiple Disk clutch brake (MD) 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.

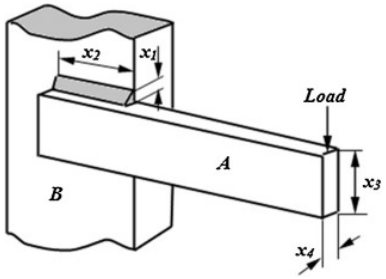


Figure 4: Schematic view of the welded beam design problem.

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
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
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 2: Statistical results of different methods for Welded beam 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, MBA, CMA-ES, MVDE, CPSO, LCA, NM-PSO, APSO, WCA, IABC-MAL, IAPSO and SAMP-Jaya. Table 2 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 $3.65E-12$ with ten times fewer function evaluations in average than most of the methods.

The results for C-ITGO and SAMP-Jaya are very similar, with same standard deviation, but with the former 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 diameter of spring mean coil and the number of active coils. It is subject to three nonlinear and one linear inequality constraints.

A variety of methods were used in literature to solve the tension / compression spring design problem. Between them, we can cite HPSO, NM-PSO, MBA, PSO-DE, LCA, CB-ABC, QPSO, G-QPSO, APSO, CMA-ES, MVDE, DELC, IAPSO, IABC-MAL and SAMP-Jaya. Table 3 shows the statistical results of all methods, along with the number of function evaluations taken to achieve such results.

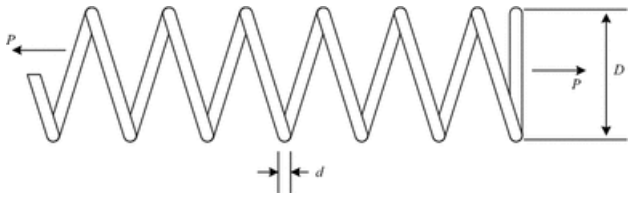


Figure 5: Schematic view of the tension/compression spring design problem

Method	Best	Mean	Worst	SD	MNFEs
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
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	2.45E06	10,000
DELC	0.012665	0.012666	0.012665	1.30E-07	20,000
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 3: Statistical results of different methods for tension/compression spring design problem.

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 3 decimal places.

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 found, differing from the optimal value reported by other methods. However, the result 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 best solution found.

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 competing method. Also, the standard deviation is very small, in the

Method	Best	Mean	Worst	SD	MNFEs
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 4: Statistical results of different methods for three-bar truss design problem.

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 (σ) on each of the truss members. Figure 6 shows the schematic view of the problem.

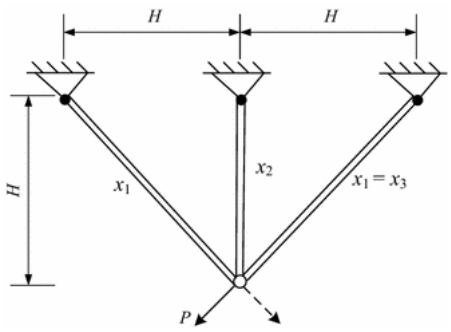


Figure 6: Schematic view of the three-bar truss design problem

Here we present only the five best-performing methods used to solve this problem, since the solutions found in literature have very small variance from each other. These methods are CMA-ES, MVDE, DELC, MBA, and IABC-MAL. We present the statistical results for this problem in Table 4.

From Table 4 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 has a smaller domain than any other engineering design problem considered in this work.

C-ITGO achieved convergence quickly, taking 136.48 function evaluations in 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 NFEs 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 (less than 0.01%) 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, where the only difference is in the bounds of the fifth variable. We will call this problems SRI and SRII, respectively.

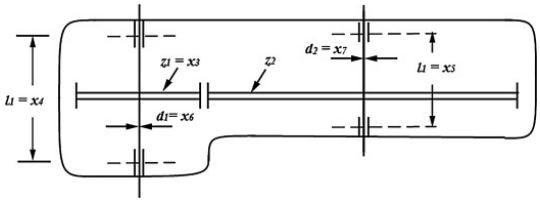


Figure 7: Schematic view of the speed reducer design problem

For SRI, we compare C-ITGO results against 6 optimization methods, namely CPSO, LCA, IPSO, APSO, IAPSO and SAMP-Jaya, shown in Table 5. The reported results for the SAMP-Jaya method seems to violate the integer constraint at variable three, but it is included so as to show that even in harder situations the C-ITGO algorithm can outperform any competing methods compared here.

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

Method	Best	Mean	Worst	SD	MNFEs
CPSO	2996.372448	2996.408525	NA	2.867E-02	30,000
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
SAMP-Jaya	2760.673988	2760.673988	2760.673988	2.54E-11	3744.66
C-ITGO	2996.34816497	2996.34816497	2996.34816497	7.5E-13	856.40

Table 5: Statistical results of different methods for the speed reducer design problem I.

Method	Best	Mean	Worst	SD	MNFEs
PSO-DE	2996.348167	2996.348174	2996.348204	6.40E-06	54,350
MBA	2994.482453	2996.769019	2999.652444	1.56E+00	6,300
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
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
C-ITGO	2994.471066	2994.471066	2994.471066	9.28E-13	491.24

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

The first three methods take more than 20,000 iterations to converge to good quality solutions, while IAPSO and SAMP-Jaya take only 6,000 and 3744.66 NFEs respectively. C-ITGO outperforms the other methods by a large margin, converging in 858.40 function calls in average, while maintaining negligible higher standard deviation than IAPSO and IPSO.

For this problem, we used considerable 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 takes 3 full iterations to converge, while converging in the first iteration in most runs.

For SRII, the methods used for comparison were PSO-DE, MBA, WCA, DELC, CB-ABC, CMA-ES, MVDE, LCA, IABC-MAL, APSO, MBA, IPSO and IAPSO. From Table 6, we can see that DELC, CMA-ES, MVDE, LCA, IAPSO and C-ITGO are the only methods that converged to the optimum in every 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 twelve times fewer function evaluations than IAPSO. The standard deviation achieved by C-ITGO is also the smallest among all compared methods.

Method	Best	Mean	Worst	SD	MNFEs
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	NA	1.14E+02	15,000
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
MBA	5889.3216	6200.64765	6392.5062	160.34	70,650
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 7: Statistical results of different methods for the pressure vessel design problem.

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 considerable 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.

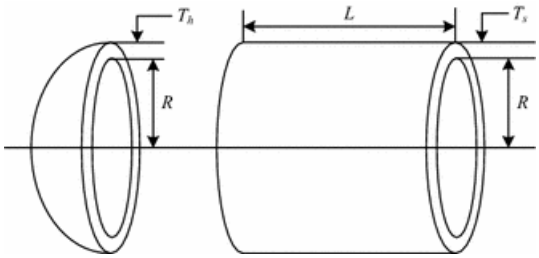


Figure 8: Schematic view of pressure vessel design problem.

We compare C-ITGO against the QPSO, G-QPSO, CMA-ES, MVDE, CB-ABC, HPSO, WCA, LCA, APSO, MBA, IABC-MAL, SAMP-Jaya and IAPSO methods. The statistical

results are shown in Table 7. Some of the methods used to solve this problem report unfeasible results (WCA, MBA and SAMP-Jaya), probably because the integer constraints are ignored. Nevertheless, we still use the results of these methods to prove the superior convergence of C-ITGO, even in constrained mixed integer problems.

The algorithms CMA-ES, CB-ABC, HPSO, IABC-MAl, IAPSO and C-ITGO are the only who found the feasible optimal solution. All other methods presented relatively poor performance, with much higher NFEs in 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 6,513.33 NFEs. Also, C-ITGO achieves the smallest standard deviation among all methods. All runs of C-ITGO converged quickly to the global optimum, showing unarguably better results than any other method used for comparison for 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 the problem is that all variables must be integers.

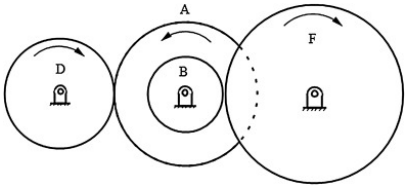


Figure 9: Gear train design problem structure.

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 optimal solution for the problem. UPSO failed at finding some solutions, while APSO has a considerable greater mean value than other the methods. 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

Method	Best	Mean	Worst	SD	MNFEs
MBA	2.700857E-12	2.062904E-08	2.47E-09	3.94E-09	1120
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 8: Statistical results of different methods for the gear train design problem.

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 converges faster than the other methods compared here, 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 disc clutch brake Osyczka (2002). There are five integer variables: the inner radius, outer radius, thickness of the disc, actuating force and number of friction surfaces (Figure 10). The problem is also constrained by nine nonlinear inequalities.

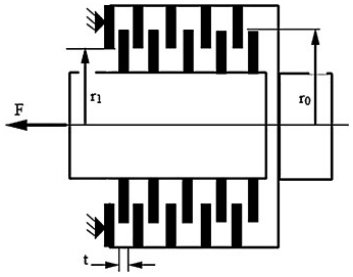


Figure 10: Schematic view of the multiple disk clutch brake design problem.

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 WCA, IPSO, APSO and IAPSO methods achieved good performance in this problem, finding the optimal solution and having very small variance, as shown in Table 9. 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.

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

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

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 also have used a very small populations of size 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 1,500 function evaluations, bringing the mean NFEs up.

3.8. Best Solutions

Table 10 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 for 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.

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 step 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, being 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 number of function evaluations 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.

As future work, we sugest 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 a faster execution time, even this is not being the main measure of performance.

Acknowledgments

This work is partially supported by the Brazilian Council for Scientific and Technological Development (CNPq): PIBIC/CNPq, process 135109/2016-7. 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.

References

Belegundu, A. D. (1982). *A study of mathematical programming methods for structural optimization*. Ph.D. thesis Department of Civil and Environmental Engineering.

Biegler, L. T., Conn, A. R., Cornuejols, G., Grossmann, I. E., Lodi, A., & Sawaya, N. (2015). BONMIN (Basic Open-source Nonlinear Mixed INteger programming). `https://projects.coin-or.org/Bonmin`. Accessed: 2017-09-7.

Brajevic, I. (2015). Crossover-based artificial bee colony algorithm for constrained optimization problems. *Neural Computing and Applications*, *26*, 1587–1601.

Coelho, L. S. (2010). Gaussian quantum-behaved particle swarm optimization approaches for constrained engineering design problems. *Expert Systems with Applications*, *37*, 1676–1683.

Currie, J. (2014). OPTI toolbox - a free matlab toolbox for optimization. `https://www.inverseproblem.co.nz/OPTI/`. Accessed: 2017-09-14.

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, *186*, 311–338.

Eskandar, H., Sadollah, A., Bahreininejad, A., & Hamdi, M. (2012). Water cycle algorithm – a novel metaheuristic optimization method for solving constrained engineering optimization problems. *Computers and Structures*, *110-111*, 151–166.

Gandomi, A. H., Yang, X. S., & Alavi, A. H. (2013). Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. *Engineering with Computers*, *29*, 17–35.

Golinski, J. (1973). An adaptive optimization system applied to machine synthesis. *Mechanism and Machine Theory*, *8*, 419–436.

Guedria, N. B. (2016). Improved accelerated PSO algorithm for mechanical engineering optimization problems. *Applied Soft Computing*, *40*, 455–467.

He, Q., & Wang, L. (2007a). An effective co-evolutionary particle swarm optimization for constrained engineering design problems. *Engineering Applications of Artificial Intelligence*, *20*, 89–99.

He, Q., & Wang, L. (2007b). A hybrid particle swarm optimization with a feasibility-based rule for constrained optimization. *Applied Mathematics and Computation*, *186*, 1407–1422.

Henderson, N., Rêgo, S. M., Sacco, W. F., & Rodrigues, R. A. (2015). A new look at the

topographical global optimization method and its application to the phase stability analysis of mixtures. *Chemical Engineering Science*, *127*, 151–174.

Jardim, L. C. S., Knupp, D. C., Sacco, W. F., & Silva Neto, A. J. (2015). Iterative topographical global optimization with space reduction. In *SENAI CIMATEC, Salvador, BA*.

Kashan, A. H. (2011). An efficient algorithm for constrained global optimization and application to mechanical engineering design: League Championship Algorithm (LCA). *Computer-Aided Design*, *43*, 1769–1792.

Le Digabel, S. (2011). NOMAD: Nonlinear optimization with the MADS algorithm. *ACM Transactions on Mathematical Software*, *37*, 44:1–44:15.

Liang, J. J., Shang, Z., & Li, Z. (2010). Coevolutionary comprehensive learning particle swarm optimizer. In *IEEE Congress on Evolutionary Computation* (pp. 1505–1512).

Liang, Y., Wan, Z., & Fang, D. (2017). An improved artificial bee colony algorithm for solving constrained optimization problems. *International Journal of Machine Learning and Cybernetics*, *8*, 739–754.

Lio, H., Cai, Z., & Wang, Y. (2010). Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization. *Applied Soft Computing*, *10*, 629–640.

Long, W., Liang, X., Cai, S., Jiao, J., & Zhang, W. (2017). An improved artificial bee colony with modified augmented lagrangian for constrained optimization. *Soft Computing*, (pp. 1–22).

Machado-Coelho, T. M., Machado, A. M. C., Jaulin, L., Ekel, P., Pedrycz, W., & Soares, G. L. (2017). An interval space reducing method for constrained problems with particle swarm optimization. *Applied Soft Computing*, *59*, 405–417.

MathWorks, I. (2017). Find minimum of constrained nonlinear multivariable function - MATLAB fmincon. `https://www.mathworks.com/help/optim/ug/fmincon.html`. Accessed: 2017-09-28.

Matsumoto, M., & Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, *8*, 3–30.

Melo, V. V., & Carosio, G. L. C. (2013). Investigating multi-view differential evolution for solving constrained engineering design problems. *Expert Systems with Applications*, *40*, 3370–3377.

Melo, V. V., & Iacca, G. (2014). A modified covariance matrix adaptation evolution strategy with adaptive penalty function and restart for constrained optimization. *Expert Systems with Applications*, *41*, 7077–7094.

Montes, E. M., Reyes, J. V., & C., C. C. A. (2006). Modified differential evolution for constrained optimization. In *IEEE Congress on Evolutionary Computation*.

Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization*. (2nd ed.). Springer Series in Operations Research, Springer Verlag.

Osyczka, A. (2002). *Evolutionary algorithms for single and multicriteria design optimization: studies in fuzzyness and soft computing*. Physica-Verlag, Heidelberg.

Parsopoulos, K. E., & Vrahatis, M. N. (2005). Unified particle swarm optimization for solving constrained engineering optimization problems. In *International Conference on Natural Computation* (pp. 582–591).

Ragsdell, K., & Phillips, D. (1973). Optimization in pre-contract ship design. *Proceedings of international conference on computer application in the automation of shipyard operation and ship design. Tokyo, Japan*, (pp. 327–338).

Ragsdell, K., & Phillips, D. (1976). Optimal design of a class of welded structures using geometric programming. *Journal of Engineering for Industry*, *98*, 1021–1025.

Rao, R. V. (2016). A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *International Journal of Industrial Engineering Computations*, *7*, 19–34.

Rao, R. V., & Saroj, A. (2017). A self-adaptive multi-population based jaya algorithm for engineering optimization. In press, corrected proof, Available online 4 May 2017.

Rueda, J. L., & Erlich, I. (2016). Solving the CEC2016 real-parameter single objective optimization problems through mvmo-phm. In *IEEE Congress on Evolutionary Computation*.

Sacco, W. F., Henderson, N., & Rios Coleho, A. C. (2014). Topographical global optimization applied to nuclear reactor core design: Some preliminary results. *Annals of Nuclear Energy*, *65*, 166–173.

Sadollah, A., Bahreininejad, A., Eskandar, H., & Hamdi, M. (2013). Mine blast algorithm: A new population based algorithm for solving constrained engineering optimization problems. *Applied Soft Computing*, *13*, 2592–2612.

Saha, A., Datta, R., & Deb, K. (2010). Hybrid gradient projection based genetic algorithms for constrained optimization. In *IEEE Congress on Evolutionary Computation* (pp. 2851–2858).

Sandgren, E. (1990). Nonlinear integer and discrete programming in mechanical design optimization. *ASME Journal of Mechanical Design*, *112(2)*, 223–229.

Sarker, R. A., Elsayed, S. M., & Ray, T. (2014). Differential evolution with dynamic parameters selection for optimization problems. *IEEE Transactions on Evolutionary Computation*, *18*, 689–707.

Sobol, I. (1967). The distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, *7*, 86–112.

Takahama, T., & Sakai, S. (2010). Constrained optimization by the epsilon constrained differential evolution with an archive and gradient-based mutation. In *IEEE Congress on Evolutionary Computation* (pp. 1680–1688).

Törn, A., & Viitanen, S. (1992). *Topographical global optimization*. Princeton University Press.

Törn, A., & Viitanen, S. (1996). Iterative topographical global optimization. In *State of the art in global optimization* (pp. 353–363). Springer.

Wang, L., & Li, L. (2009). An effective differential evolution with level comparison for constrained engineering design. *Structural and Multidisciplinary Optimization*, *41*, 947–963.

Yang, X. S. (2010). *Engineering Optimization: an Introduction with Metaheuristic Applications*. John Wiley & Sons, Inc., Hoboken, New Jersey.

Zahara, E., & Kao, Y. T. (2009). Hybrid Nelder–Mead simplex search and particle swarm optimization for constrained engineering design problems. *Expert Systems with Applications*, *36*, 3880–3886.