Improvements on Biased Random-Key Genetic Algorithms for Non-Linearly Constrained Global Optimization

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

[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, Brazil*

Abstract

Keywords: Metaheuristics, BRKGA, Global Optimization, Local Search.

1. Introduction

2. I-TGO

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.

Let the objective function or fitness $f(\cdot)$ be a mapping $f : \Omega \to \mathbb{R}$, where $\Omega \in \mathbb{R}^N$ is the feasible space of solutions, defined by $\Omega = \{\boldsymbol{x} \in \mathbb{R}^n :$

*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)

$\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}\}$. Here, $\boldsymbol{l}, \boldsymbol{u} \in \mathbb{R}^N$ are vectors defining the lower and upper bound of the solution space, and the comparison is made elementwise. Consequently, $l_i \leq u_i$ for $i = 1, ..., N$. The goal is to find the global minimum $\boldsymbol{x}^*$, such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})\ \forall \boldsymbol{x} \in \Omega$.

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 defined solution space. The TGO performance depends critically 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 have a signifcantly 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 covers the space more uniformly than that of Mersene 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 from $\boldsymbol{x}$, 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:

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

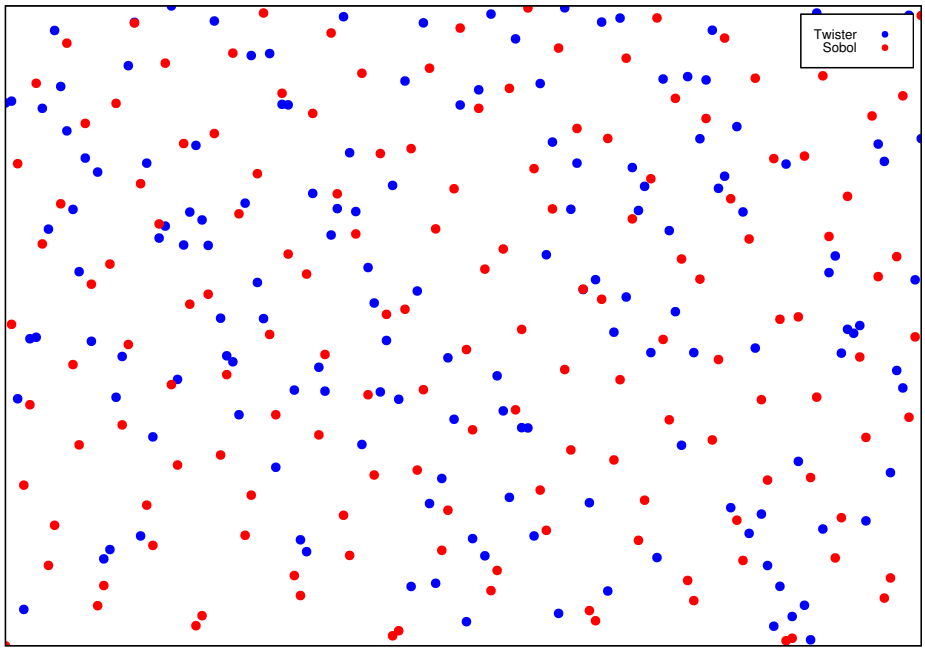


Figure 1: Example showing the distribution of points in the domain $[0, 1]^2$ by the Mersenne Twister generator (blue points) and by a Sobol sequence (red points).

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 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 on 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 tunning is necessary.

The last step of the TGO method consists in applying some sort of local search in every local optimum points 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), smooth (Henderson et al., 2015) and non-smooth (Jardim et al., 2015) optimization methods. The local search algorithm depends directly on the properties of the function to be optimized, and is of critical 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 elements $P = \{(-0.2, 0.16), (1.2, -0.3), (-0.6, 1.2), (-0.9, 2.4), (2.0, 2.0), (2.7, 0.3), (0.3, 2.2), (2.0, -0.2), (1.3, 2.8), (1.3, 1.2)\}$, 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:

$$A = \left[\begin{array}{cccccccccc|c} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

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 *I-TGO* (*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 I-TGO. 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 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 are usually smaller than in the previous iteration.

2.2. Constrained Optimization

Until the moment, the I-TGO algorithm was presented only for unconstrained optimization problems. To tackle general non-linear constraints, we adopt a mechanism for comparing solutions proposed in Deb

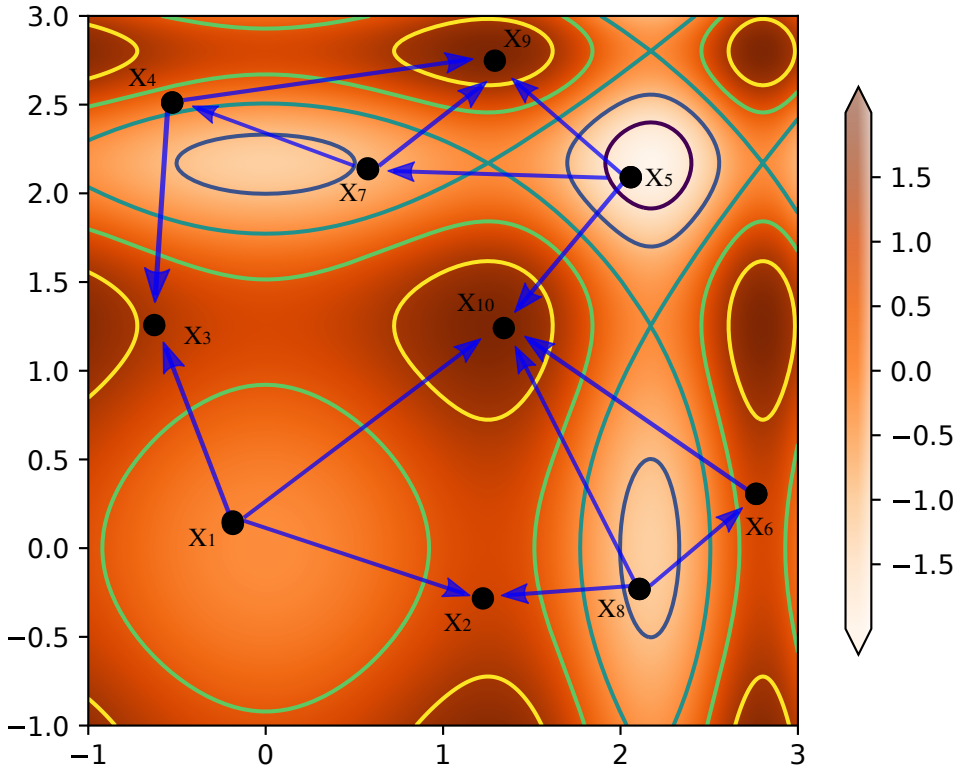
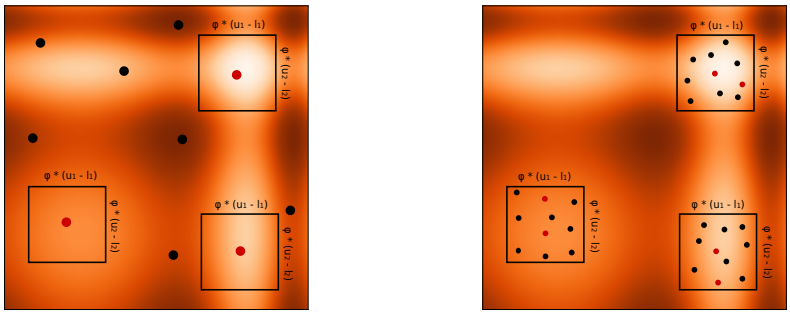


Figure 2: Contour plot of the function $f(x, y) = sen(x^2) + cos(y^2)$, and the graph based on the population.



(a) (b)

Figure 3: Example showing the application of the space redution 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.

(2000), and used in some very competitive meta-heuristics, specially on differential evolution algorithms (Sarker et al., 2014; Takahama & Sakai, 2006; Montes et al., 2006). As in this work is proposed a Contrained version of I-TGO, in the rest of this work, it is named as CI-TGO. The method for comparison comprise 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 smallest sum of constraint violation 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 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, 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 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.

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 CI-TGO algorithm, that stands for Constrained I-TGO.

The parameters are the functions $f(\cdot)$ and $v(\cdot)$, which return the function value and sum of constraints violation respectively, 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 three step comparison α.

Algorithm 1 CI-TGO($f(\cdot)$, $v(\cdot)$, $\boldsymbol{l}$, $\boldsymbol{u}$, $\boldsymbol{PS}$, $\boldsymbol{KS}$, 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 $\boldsymbol{PS}(1)$ elements and again 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 is the last space reduction, we have to save the local optimum points in the set $TopoBest$ to apply local search (line 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 $0.5 * (\boldsymbol{u}_i - \boldsymbol{l}_i),\ i = 1, ..., n$, 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, and returns 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 found element in the whole search at line 19. If it is better than

the best found element, 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 the number of iterations to LS_2.

At line 21 we compare the solution generated by the second local search ($\boldsymbol{x}$) against the best found element ($\boldsymbol{best}$) using the three step 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. 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.

The only thing left is the $TopographicalHeuristic$ procedure, shown in algorithm 3. The function takes as parameter the objective function $f(\cdot)$ and 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 on 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 indice 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 $\boldsymbol{best}$. 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 optimum points (or the $\boldsymbol{best}$ element).

2.4. Local Search

We used Matlab to program CI-TGO, so 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 Program-

ming), 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 *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 CI-TGO. 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 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 in 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 nearly optimum solutions.

3. Computational Results

To evaluate the performance of the developed method, we use eight difficult constrained engineering optimization problems, with non-convex and non-linear functions, and some with integer variables. We now present a general overview of each problem, along with comparison with state of the art techniques and their results..

In all tests, given the stochastic characteristic of CI-TGO, we run it 25 times, saving the best and worst feasible solutions, as well as the mean value and standard deviation of the fitness after all runs. Given the great variability between the results found in literature for most problems, we stop the execution of CI-TGO as soon as the best found solution in a run is considerable close to the optimum. Also, a fixed number of maximum number of function evaluations was set for each problem. If the number of FEs exceeds this number, the algorithm immediately stops, and the best found solution is returned. At the end, the average number of function calls is reported for all runs.

In our tests, all solutions reported by CI-TGO are completely feasible, for all problems, so we exclude from comparison methods that violate any constraint.

We compare 27 different optimization methods against CI-TGO. The most common meta-heuristic is the particle swarm optimization, including PSO-DE (Lio et al., 2010), a hybrid particle swarm optimization method combined with differential evolution, 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 APSO, standind for accelerated PSO (Yang, 2010) and gaussian quantum-behaved particle swarm optimization methods, named QPSO and G-QPSO (Coelho, 2010). More recently, and among the best state of the art PSO methods, we have IPSO (Machado-Coelho et al., 2017), which incorporates an interval reducing procedure and IAPSO (Guedria, 2016), which proposes some modifications and improves the

APSO method.

Many other different methods were also used, such as the Mine Blast Algorithm (MBA) Sadollah et al. (2013) that is a population-based optimization method based on mine bomb explosion concept, the League Championship Algorithm (LCA) **?**, which models a league championship environment with artificial teams playing in an artificial league,

3.1. Welded bean design problem

The welded bean design (Ragsdell & Phillips, 1976) is the problem of minimizing the fabrication cost of a welded bean, subject to seven inequality constraints, being two linear and five non linear. The constraints include shear stress, bending stress in the beam, buckloading on the bar and end deflection on the beam. The four design variables are continuous. Figure 4 shows the structure of the problem.

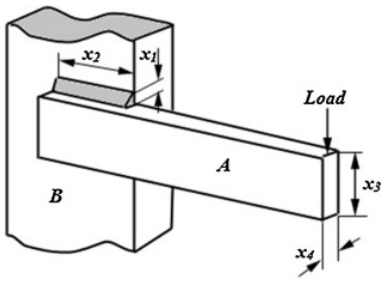


Figure 4: Welded bean design problem structure.

We compare the results obtained by CI-TGO against a number of state of the art methods used to solve this problem, including PSO-DE, HPSO, MBA, CPSO, LCA, NM-PSO, IABC-MAL, IAPSO and SAMP-Jaya. Table 1 shows the comparison of the results obtained by all cited methods to solve the welded bean design problem. All the methods, with exception of SAMP-Jaya and CI-TGO, took more than 10,000 iterations to achieve good quality solutions. CI-TGO achieved optimal solutions with a very small standard deviation of $6.7E-16$ with ten times less function evaluations in average than most of the methods.

Method	Best	Mean	Worst	SD	NFEs
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
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
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
CI-TGO	1.7248523	1.7248523	1.7248523	6.7E-16	1,166.84

Table 1: Welded bean.

The results for CI-TGO 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 rest of the 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 string (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 non linear and one linear inequality constraints.

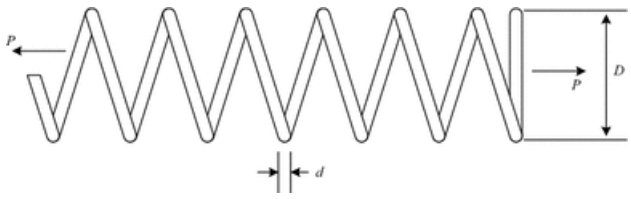


Figure 5: Schematic view of the tension/compression spring design problem

A variety of methods were used in literature to solve the tension / compression spring design problem. Between them, we can cite HPSO, NM-PSO, MBA, PSO-DE, ABC, PSO, CB-ABC, CPSO, GA1, GA2, QPSO, G-QPSO, CAEP, DELC, IAPSO, IABC-MAL and SAMP-Jaya. Table 2 shows

Method	Best	Mean	Worst	SD	NFEs
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
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
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
CI-TGO	0.01266523	0.01266523	0.01266525	2.81E-9	535.08

Table 2: Tension Compression.

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 four of the methods (PSO, 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 found solution, 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 in rounding between the methods, we consider $f(\boldsymbol{x}) = 0.012665$ to be the best found solution.

The CI-TGO method performed remarkably well on this problem, converging to the best found solution among all methods 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 order of $2.81E-9$.

Given the relatively small domain of the problem, we believe that the CI-TGO method was able to cover the space efficiently, providing

Method	Best	Mean	Worst	SD	NFEs
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
CI-TGO	263.895843	263.895843	263.895843	6.4E-12	141.8

Table 3: Three Bar Truss.

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. Theere are two continuous design variables and three non linear inequality constraints, based on the stress (σ) on each of the truss members. Figure 6 shows the schematic view of the problem.

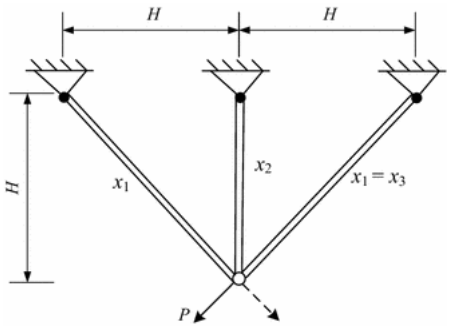


Figure 6: Structure of three-bar truss design problem

Here we present only the five good 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 7.

We can 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 less variables and has a smaller domain than any other problem used in this work.

CI-TGO achieved convergence quickly, taking 141.8 function evaluations in average. In this problem, we have set a small population, as well as small number of function calls allowed in the local search procedure. 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 CI-TGO method still has a worst standard deviation than DELC and IABC-MAL.

3.4. Speed reducer design 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 pinion, first and second length of shafts between bearings and first and second shafts diameter (Golinski, 1973). The problem has 11 non linear constraints, and the third variable is constrained to be 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.

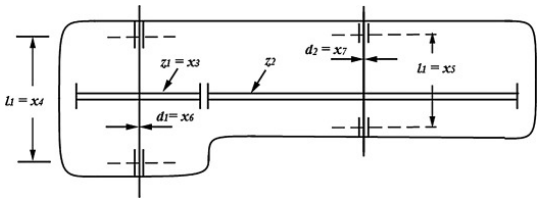


Figure 7: Schematic view of the speed reducer design problem

For SRI, we compare CI-TGO results against 5 optimization methods, namely COPSO, LCA, CiP-PSO, IPSO and IAPSO, shown in table 4. The reported results for the SAMP-Jaya method seems to violate the integer constraint at variable three, so we do not include it in our comparison. The LCA, IPSO, IAPSO and CI-TGO present similar results, all finding the global optimum solution and having very low standard deviation.

Method	Best	Mean	Worst	SD	NFEs
SiC-PSO	2996.34816	2996.4085	NA	0.0	24,000
COPSO	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
IAPSO	2996.34816497	2996.34816497	2996.34816497	6,88E-13	6,000
CI-TGO	2996.34816497	2996.34816497	2996.34816497	7.5E-13	856.40

Table 4: SP1.

The SiC-PSO and COPSO methods present inferior results, with higher average fitness. Besides having significant difference between the best from the mean fitness value, the SiC-PSO reports a standard deviation of 0.

The first four methods take more than 20,000 iterations to converge to good quality solutions, while IAPSO takes only 6,000 FEs. CI-TGO outperforms the other methods by a large margin, converging in 858.40 function calls in average, while maintaining negligible worst 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, CI-TGO 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, DELC, CB-ABC, CMA-ES, IABC-MAL, IPSO and IAPSO. From Table 5, we can see that DELC, CMA-ES, IAPSO and CI-TGO are the only methods that converged very close to the optimum in every run, having very small standard deviation. Although similar results are observed regarding the quality of the solutions, CI-TGO converges much quicker than any competing method, using six times less function evaluations than IAPSO. The standard deviation achieved by CI-TGO is also the smallest among all compared methods.

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.

Method	Best	Mean	Worst	SD	NFEs
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
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
IPSO	2994.471067	2994.47108	2994.4711	9.27E-06	20,000
IAPSO	2994.471066	2994.471066	2994.471066	2.65E-09	6,000
CI-TGO	2994.471066	2994.471066	2994.471066	9.28E-13	906.32

Table 5: SP2.

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 non linear 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.

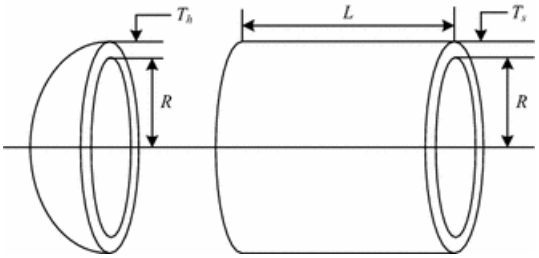


Figure 8: Pressure vessel design problem structure.

Some methods in literature optimized the pressure vessel design problem without taking into account the integer constraints at the first and second variables. So, to compare the CI-TGO performance against other methods, we will only consider methods that respect the integer constraints.

Method	Best	Mean	Worst	SD	NFEs
PSO	6693.7212	14076.324	8756.6803	1.49E+03	8,000
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
CB-ABC	6059.7143	6126.6237	NA	1.14E+02	15,000
IABC-MAL	6059.7143	6072.5972	6089.2720	1.88E-06	15,000
IAPSO	6059.7143	6068.7539	6090.5314	14.0057	7,500
CI-TGO	6059.7143	6059.7143	6059.7143	9.8E-13	1101.64

Table 6: Pressure Vessel.

We compare CI-TGO against the PSO, GA1, GA2, QPSO, G-QPSO, CB-ABC, IABC-MAL and IAPSO methods. The statistical results are shown in table 6. The algorithms CB-ABC, IABC-MAl, IAPSO and CI-TGO are the only who found the optimal solution, and also the ones with smaller mean fitness value. All other methods presented relatively poor performance, with much higher FEs in average.

CI-TGO takes less than six times the number of function evaluations to converge than the best competing method, IAPSO, which reported a result of 7,500 FEs. Also, CI-TGO achieves a extremely small standard deviation of $9.8E-13$, many orders of magnitude smaller than IAPSO. All runs of CI-TGO converged quickly to the global optimum, showing unarguably better results than any other method used for comparison in this work.

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.

We compare the CI-TGO results, shown in table 9, against other five methods: CS, APSO, MBA, UPSO 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 in the number of function calls, varying from 100,00

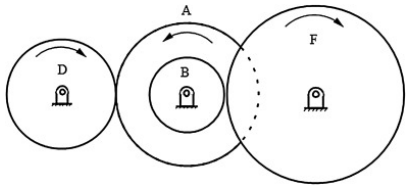


Figure 9: Pressure vessel design problem structure.

Method	Best	Mean	Worst	SD	NFEs
MBA[12]	2.700857E-12	2.062904E-08	2.471635E-09	3.94E-09	1120
UPSO[63]	2.700857E-12	3.80562E-08	N/A	1.09E-07	100,000
CS[62]	2.7009E-12	1.9841E-9	2.3576E-9	3.5546E-9	5,000
APSO[34]*	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
CI-TGO	2.700857E-12	4.6504232E-09	2.7264505E-08	6.85E-09	773.0

Table 7: Three Bar Truss.

for UPSO to 800, for IAPSO and CI-TGO.

Since this is a 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 $6.85E-09$ standard deviation, more than IAPSO, CS and MBA. The mean fitness value, however, is only worst than the MBA method, which takes 1120 function calls to converge. Besides these drawbacks, CI-TGO 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 non linear inequalities.

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

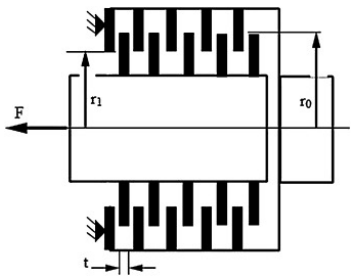


Figure 10: Schematic view of the multiple disk clutch brake design problem.

Method	Best	Mean	Worst	SD	NFEs
WCA	0.313656	0.313656	0.313656	1.69E-16	500
IPSO	0.313656	0.313656	0.313656	0.0	20,000
IAPSO	0.313656	0.313656	0.313656	1.13E-16	400
CI-TGO	0.313656	0.313656	0.313656	1.13E-16	286.48

Table 8: Multiple Disk.

achieved good performance in this problem, finding the optimal solution and having very small variance, as shown in table 8. 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 CI-TGO 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 FEs up.

4. Conclusion and Future Work

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.

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. *Comput. Methods Appl. Mech. Eng*, *186*, 311–338.

Golinski, J. (1973). An adaptive optimization system applied to machine synthesis. *Mech. Mach. 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). 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.

Le Digabel, S. (2011). Nomad: Nonlinear optimization with the mads algorithm. *ACM Transactions on Mathematical Software*, *37(4)*, 44:1–44:15.

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.

Machado-Coelho, A. M. C., M. T. abd Machado, 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 Trans. Model. Comput. Simul.*, *8*, 3–30.

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

Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization, Second Edition*. 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*, (p. 218).

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. *J. Eng. Ind.*, (pp. 1021–1025).

Rueda, J. L., & Erlich, I. (2016). Solving the cec2016 real-parameter single objective optimization problems through mvmo-phm. *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.

Sandgren, E. (1990). Nonlinear integer and discrete programming in mechanical design optimization. *ASME J Mech Des*, *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 Comput. Math. Math. Phys*, *7*, 86–112.

Takahama, T., & Sakai, S. (2006). Constrained optimization by the ϵ constrained differential evolution with gradient-based mutation and feasible elites. *IEEE Congress on Evolutionary Computation*, .

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

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.