Parallel evolutionary algorithm for constrained single and multi objective optimisation–Part I: Methods, single and two-objective test cases

Dorival M. Pedroso, *Member, IEEE*

***Abstract*—This paper presents an algorithm employing differential evolution operations for solving nonlinear optimisation problems with many objectives and constraints. Variables can be encoded by integers, real numbers or both. Strategies to efficiently obtain accurate solutions are proposed. Algorithms are developed to compare trial solutions with robustness in both single and multi optimisation problems at the same time. This is done by taking into account some metrics related to diversity and Pareto fronts. The concept of Pareto dominance for multiple constraints is introduced. The resulting code, named goga, is written in Go (golang) and is open source. Numerical examples are presented illustrating the capabilities and performance of the algorithm. Several tests are passed with excellent accuracy and repeatability characteristics and a few challenging tests are solved with an increased number of iterations. An ideal speedup is observed during the parallel computations. Part II presents additional tests and some applications including topology optimisation and environmental economical dispatch planning.**

***Index Terms*—genetic algorithm, differential evolution, niching, tournament, concurrency**

I. Introduction

Evolutionary algorithms (EAs), including genetic algorithms (GAs), are optimisation techniques with a good dissemination in engineering and many other areas; see e.g. [1]–[5]. These techniques avoid the use of derivatives and can tackle quite complex problems, including discontinuous functions or objectives that may depend on a mixture of data types (binary, integer, real). Nonetheless, EAs do not necessarily seek for an exact answer and cannot easily tell whether the solution has converged or not. Therefore, EAs must be well developed and tested under several circumstances in order to be reliable and able to output accurate results after every run.

One essential characteristic for a good performance of an evolutionary algorithm is the ability to maintain diversity during the solution process. Simple GAs in particular suffer from early convergence when, after a number of iterations, most individuals have very similar genetic data and cannot easily produce better solutions unless mutation happens. Nonetheless, the control of mutation in this situation is not straightforward. Several research works discuss this topic in detail and some present strategies to overcome the problem [6]–[9]. This contribution considers one solution known as *crowding approach to niching* inspired by the algorithms in [10], [11]. The diversity in this method is maintained by running tournaments between randomly selected trial solutions and only replacing the previous candidate if the new solution wins; the competitors in the tournament are firstly matched with respect to a minimum distance estimate.

School of Civil Engineering, The University of Queensland, St Lucia QLD 4072, Australia, e-mail:d.pedroso@uq.edu.au

Many important problems in Engineering involve multiple constraints; e.g. structural optimisation of frames where the cross sectional areas are always positive quantities and loads may be unidirectional. Another example is the economic/environmental load dispatch problem where the capacity of power generators is limited and the available power must compensate the losses (e.g. along the transmission lines)—the power balance. These two applications are dealt with in the second part (Part II) [12].

Simple methods to handle constraints in evolutionary algorithms employ the so-called penalty factors that are added to the objective function when the particular trial solution violates a constraint. Nevertheless, this addition "pollutes" the objective value causing problems when comparing solutions. Studies on constraints handling methods are available [13]–[17]. In addition, there are better methods to implement a constraint handling strategy such as the one proposed in [18]; see comprehensive analysis of several methods in [19].

Also of great importance are multi objective optimisation problems. For instance, the economic/environmental dispatch problem mentioned above seeks for a compromise between a minimal cost and minimal emission of pollutants—two opposing objectives. The concept of non-dominated Pareto optimal fronts helps in this decision. Algorithms have then been developed along the years to solve multi objective problems as well; see e.g. [20]–[33]

Real-coded genetic algorithms have been a challenge in the early days; but good solutions were introduced and studied in [2], [34]–[36]. A simple attempt that split a float point number into smaller chunks was also successfully proposed and implemented in [37]. Nonetheless, the performance was not necessarily always the best. The differential evolution (DE) [38], [39] on the other hand naturally implements recombinations of real numbers. DE has proven to be an efficient method and at times even better than classical genetic operators; see e.g. [33]. This behaviour has been observed in the experiments performed by the research reported here leading to a decision to use DE for real numbers only. More details on DE can be found in [40], [41].

Parallel computing has become a common technique to solve large scale problems. Moreover, modern CPU design has focused towards adding multiple computing cores to chips because of power and temperature limitations in silicon. There-

fore, new algorithms have to consider parallel techniques in order to make better use of the computer hardware. Within this subject, concurrency is usually attached to parallel computing but it can be thought of as an independent concept for the design of algorithms that later can be run in parallel or not. For evolutionary algorithms, concurrency is very natural since trial solutions can be assigned into different groups that are run in parallel. Besides speeding computations up, this approach may widen the search space and improve the convergence properties.

This paper presents an EA for solving optimisation problems with multiple objectives and multiple constraints. The best ideas collected from the aforementioned literature are taken into consideration. Strategies to obtain accurate solutions are proposed: (1) direct handling of constraints without penalty weights; (2) crowding/niching for diversity maintenance; (3) random DE vector length coefficient; and (4) parallel execution with groups of solutions. The ability to mix integers and real numbers is a design aim. A method to compare trial solutions considering multiple constraints and objectives is developed in which the number of violations is accounted for and a Pareto comparison is also applied to constraints via an auxiliary variable called out-of-range (OOR).

The code, implemented in Go language (golang) [42], is available in https://github.com/cpmech/goga as free/open source and takes advantage of the native concurrency features of Go. For historical reasons, the code is named goga after 'Go genetic algorithm'; however it only employs genetic operators when dealing with integers. The source code of all examples and graphs are posted in the same web address.

In this paper, the statement *number of constraints* excludes the limits of variables. The word 'unconstrained' disregards these *box constraints*. Hence, all problems here are in fact constrained. Confusion is avoided since the subset of constraints is clearly indicated by the u_i functions.

II. Single and Multi Objective Optimisation

The general nonlinear optimisation problem to be solved is

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimise}} \quad & \{f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots\} \\ \text{subject to} \quad & u_i(\boldsymbol{x}) = 0 \\ & x_j^L \le x_j \le x_j^U \end{aligned} \tag{1}$$

where $f_i(\boldsymbol{x})$ and $\boldsymbol{x} = \{x_0, x_1, x_2, \ldots\}$ are the objective functions and unknowns, respectively. There are N_f objectives and N_x unknowns. An auxiliary set of functions indicated by $u_i(\boldsymbol{x})$ is introduced to handle constraints. These functions measure how much a solution $\boldsymbol{x}$ is *out-of-range* or *unfeasible*—it can be a binary number. Note that the above general form includes unknowns $\boldsymbol{x}$ containing mixed types (e.g. integer and real numbers). In this section, $\boldsymbol{x}$ means both real variables $\boldsymbol{x}$ or a set $(\boldsymbol{x}, \boldsymbol{y})$ with real and integer $(\boldsymbol{y})$ variables. Hereafter, the acronym OVA is also employed to indicate an objective-value.

The out-of-range (OOR) functions $u_i(\boldsymbol{x})$ must return positive values that become smaller as the trial solution $\boldsymbol{x}$ becomes closer to satisfying the corresponding constraint. If the solution is feasible, the OOR function is simply zero. Therefore, the minimisation of $u_i(\boldsymbol{x})$ is indirectly sought by the evolutionary algorithm although only when $u_i = 0$, the problem is considered solved. The pursue of zero OORs has priority over minimising OVAs as discussed in the next section.

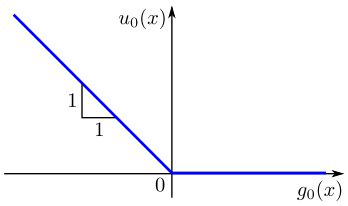

Fig. 1. Out-of-range (unfeasible) function u_0 for inequality constraint g_0.

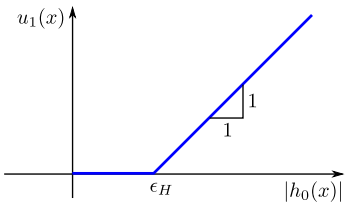

Fig. 2. Out-of-range (unfeasible) function u_1 for equality constraint h_0.

The general form in (1) includes the more specific problem expressed as follows

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimise}} \quad & \{f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots\} \\ \text{subject to} \quad & g_i(\boldsymbol{x}) \ge 0 \\ & h_j(\boldsymbol{x}) = 0 \\ & x_k^L \le x_k \le x_k^U \end{aligned} \tag{2}$$

where $g_i(\boldsymbol{x})$ and $h_i(\boldsymbol{x})$ are inequality and equality constraint functions, respectively. There are N_g inequalities and N_h equalities. These two types of constraints can be easily converted into the OOR form by means of the u_i functions in (1). The number of OOR functions will then be

$$N_u = N_g + N_h \tag{3}$$

For example, in a problem involving two constraints g_0 and h_0, the ramp function defined by

$$u_0(x) = \begin{cases} 0 & \text{if } g_0(x) \ge 0 \\ -g_0(x) & \text{otherwise} \end{cases} \tag{4}$$

can be used for the first constraint and the function defined by

$$u_1(x) = \begin{cases} 0 & \text{if } |h_0(x)| \le \epsilon_H \\ |h_0(x)| - \epsilon_H & \text{otherwise} \end{cases} \tag{5}$$

can be used for the second one. Therein, ϵ_H is a small number, but not very small (i.e. not machine precision), that is required to effectively convert the equality constraint into an inequality. This is necessary in order to allow the algorithm to search within a small gap closer to the equality constraint. An illustration of (4) is given in Fig. 1 and an illustration of (5) is given in Fig. 2.

Note that the strategy involving the out-of-range function is one step to effectively handle multiple constraints. In the evolutionary algorithm, this method will pull unfeasible solutions towards the feasibility region.

To obtain a solution of the optimisation problem defined above, a set of trial solutions is generated and combined. This set is indicated by

$$S = \{\underset{\cdot}{\boldsymbol{x}}_0, \underset{\cdot}{\boldsymbol{x}}_1, \ldots \underset{\cdot}{\boldsymbol{x}}_{Nsol-1}\} \tag{6}$$

where $\underset{\cdot}{\boldsymbol{x}}_i$ are the trial solutions and there are N_{sol} of them. For each trial solution $\underset{\cdot}{\boldsymbol{x}}_i$ corresponds a vector of objective values $\underset{\cdot}{\boldsymbol{f}}_i$ and a vector of out-of-range values $\underset{\cdot}{\boldsymbol{u}}_i$. For example, $\underset{\cdot}{\boldsymbol{f}}_A(\underset{\cdot}{\boldsymbol{x}}_A)$ is the vector of objective values computed with $\underset{\cdot}{\boldsymbol{x}}_A$.

The minimisation problem in (1) considers the concept of Pareto domination; see, e.g. [22]. In this concept, a vector $\underset{\cdot}{\boldsymbol{v}}_A$ dominates another one $\underset{\cdot}{\boldsymbol{v}}_B$ if and only if the following conditions are satisfied at the same time

$$\begin{aligned} v_i^A &\le v_i^B \qquad \text{for all} \qquad i = 0, 1, \ldots N_v - 1 \\ v_i^A &< v_i^B \qquad \text{for at least one} \qquad i = 0, 1, \ldots N_v - 1 \end{aligned} \tag{7}$$

where v_i^j are the components of the vector $\underset{\cdot}{\boldsymbol{v}}_j$ and N_v is the length of the vector.

Therefore, for multi objective optimisation problems, the word *minimise* in (1) means to find the Pareto optimal solution satisfying (7) using the vectors $\underset{\cdot}{\boldsymbol{f}}_i$ of objective values. Single objective optimisation problems are also included in (1); in this case, $N_f = 1$.

In this paper, the command `ParetoComparison` means to apply (7) to any two vectors related to the trial solutions A and B and to find whether solution A dominates solution B ($A_{dom} = \text{true}$), solution B dominates solution A ($B_{dom} = \text{true}$), or none of the above. Thus, it can be implemented in a routine that takes two vector arguments and returns two Boolean results as follows

$$A_{dom}, B_{dom} \leftarrow \texttt{ParetoComparison}\,(\underset{\cdot}{\boldsymbol{v}}_A, \underset{\cdot}{\boldsymbol{v}}_B) \tag{8}$$

III. Constraints handling

The method proposed to find solutions satisfying all constraints is inspired by the one introduced in [18]; but completely avoids adding weights to the objective function. In addition, the method is designed to deal with multi objective problems at the same time as single objective ones.

The method is motivated by the way some constraints are dealt with in engineering. Specifically, a trial solution not satisfying all the constraints, thus termed *unfeasible*, is completely ignored; e.g. an unfeasible circuit would never be even considered. The problem that arises in this approach is when all solutions are unfeasible and therefore comparisons cannot be made. One way around this problem is to rank how unfeasible a solution is, only in this exception, and then perform the comparison between trial solutions. This is facilitated by the OOR functions $u_i(\underset{\cdot}{\boldsymbol{x}})$ discussed earlier.

In essence, the comparison between two trial solutions A and B is carried out by first checking the number of constraint violations N_{viol}^i due to solution i and by performing a Pareto comparison on the OOR values $\underset{\cdot}{\boldsymbol{u}}_i$. This means that being a feasible solution is primal to being an optimal solution. Afterwards, if both solutions are feasible with zero OORs, a Pareto comparison is carried out on the OVA values $\underset{\cdot}{\boldsymbol{f}}_i$. The steps are collected in Algorithm 1 where the results are two flags describing the domination status between A and B; it can happen that yet none dominates another.

Algorithm 1. `CompareSolutions`(A,B) performs the comparison between trial solutions considering the objective values f_i and the handling of constraints via the out-of-range values u_i.

Input: trial solutions A and B
Output: A_{dom} (A dominates) and B_{dom} (B dominates)
$A_{dom}, B_{dom} \leftarrow$ false, false
$N_{viol}^A, N_{viol}^B \leftarrow 0, 0$
for *OOR indices* $0 \le i < N_u$ **do**
  if $u_i^A > 0$ **then** N_{viol}^A += 1
  if $u_i^B > 0$ **then** N_{viol}^B += 1
if $N_{viol}^A > 0$ **then**
  if $N_{viol}^B > 0$ **then**
    if $N_{viol}^A < N_{viol}^B$ **then**
      $A_{dom} \leftarrow$ true
      return
    if $N_{viol}^B < N_{viol}^A$ **then**
      $B_{dom} \leftarrow$ true
      return
    $A_{dom}, B_{dom} \leftarrow$ `ParetoComparison`$(\underset{\cdot}{\boldsymbol{u}}_A, \underset{\cdot}{\boldsymbol{u}}_B)$
    if not A_{dom} and not B_{dom} **then**
      $A_{dom}, B_{dom} \leftarrow$ `ParetoComparison`$(\underset{\cdot}{\boldsymbol{f}}_A, \underset{\cdot}{\boldsymbol{f}}_B)$
    return
  $B_{dom} \leftarrow$ true
  return
if $N_{viol}^B > 0$ **then**
  $A_{dom} \leftarrow$ true
  return
$A_{dom}, B_{dom} \leftarrow$ `ParetoComparison`$(\underset{\cdot}{\boldsymbol{f}}_A, \underset{\cdot}{\boldsymbol{f}}_B)$

IV. Metrics

Given a population of solutions, a number of numeric measures are required in order to take decisions or assess the performance during the solution process. For the sake of organisation, these quantities are collected in this section and called 'metrics'. A data structure and functions can then be programmed to perform the required calculations. The following quantities are computed by the `Metrics`(Ω) function with Ω being a set of solutions

- x_i^{min} and x_i^{max} minimum and maximum solution values of real type in Ω corresponding to dimension i
- y_i^{min} and y_i^{max} minimum and maximum solution values of integer type in Ω corresponding to dimension i
- f_i^{min} and f_i^{max} minimum and maximum objective values in Ω corresponding to objective function i
- η_i *neighbouring distance*: smallest distance to any neighbour around solution i
- w_i *crowding distance* corresponding to solution i
- ϕ_i rank of the Pareto front of solution i (small is better)

The neighbour distance d_{ij} between solutions i and j is simply defined by means of

$$d_{ij} = \frac{1}{N_x}\sum_k^{N_x}\frac{x_k^i - x_k^j}{x_k^{max} - x_k^{min} + \epsilon} + \frac{1}{N_y}\sum_k^{N_y}\frac{y_k^i - y_k^j}{y_k^{max} - y_k^{min} + \epsilon} \tag{9}$$

Algorithm 2. `ParetoFronts`(Ω) computes Pareto fronts for a set of trial solutions Ω.

Input: set of trial solutions Ω
Output: set of solutions in Pareto fronts $F_{i,j}$, size of fronts z_i, and ranks ϕ_i
zero all N^i_{wins}, N^i_{loss}, ϕ_i and z_i variables
`// compute wins/losses data`
for *each solution* A *in* Ω **do**
  for *each* B *in* Ω *with index greater than* A **do**
    $A_{dom}, B_{dom} \leftarrow$ `CompareSolutions`(A, B)
    if A_{dom} **then**
      $W_A[N^A_{wins}] \leftarrow B$
      N^A_{wins} += 1
      N^B_{loss} += 1
    if B_{dom} **then**
      $W_B[N^B_{wins}] \leftarrow A$
      N^B_{wins} += 1
      N^A_{loss} += 1
`// first front`
for *each solution* A *in* Ω **do**
  if $N^A_{loss} = 0$ **then**
    $F_{0,z_0} \leftarrow A$
    z_0 += 1
`// next fronts`
for *each front index* $0 \le r < N_{sol}$ **do**
  if $z_r = 0$ **then**
    break
  $s \leftarrow r + 1$
  for *solution indices* $0 \le i < z_r$ **do**
    $A \leftarrow F_{r,i}$
    for *each solution* B *in* W_A **do**
      N^B_{loss} −= 1
      if $N^B_{loss} = 0$ **then** `// B in next F`
        $\phi_B \leftarrow s$
        $F_{s,z_s} \leftarrow B$
        z_s += 1

Algorithm 3. `CrowdingDistances`(Ω) computes the crowding distance of solutions in Ω.

Input: output data from `ParetoFronts`(Ω)
Output: crowding distances w_i
zero all w_i values
for *each Pareto front* $F_{r,\text{all}}$ *with rank* r **do**
  if *size of front* $z_r = 1$ **then**
    continue
  for *OVA indices* $0 \le j < N_f$ **do**
    $K \leftarrow$ `Sorted`($F_{r,\text{all}}$, "by obj value j")
    $\delta \leftarrow f^{max}_j - f^{min}_j + \epsilon$
    $w_{K_0}, w_{K_m} \leftarrow$ INF, INF
    for *indices* i *in* K, *except first and last* **do**
      w_i += $\frac{f^i_j - f^{i-1}_j}{\delta} \times \frac{f^{i+1}_j - f^i_j}{\delta}$

where N_x is the number of real values and N_y the number of integers. Therein $\epsilon = 10^{-15}$. The smallest value of d_{ij} corresponding to solution i is recorded in a variable η_i which is called the *neighbouring distance* of i.

The computation of the first three variables listed above is quite straightforward; they all go in the `Metrics` function.

The rank ϕ_i of the Pareto front of solution i is computed after finding all fronts using an algorithm similar to the one presented in [27]. Nonetheless, instead of using a standard Pareto dominance comparison using objective values only, the proposed comparison with constraints handling `CompareSolutions` listed in Algorithm 1 is employed instead. In this way, the unfeasible solutions will have a smaller chance to appear in the best fronts (small ϕ).

The required steps are collected in Algorithm 2 where W_i is a subset of solutions such that solution i wins the comparison against any item in this subset when using `CompareSolutions`. The algorithm is designed in such a way to allocate memory just once. To this end, the set of sets of solution indices in all fronts $F_{i,j}$ (front i and solution j) is pre-allocated as a bi-dimensional array with a maximum size equal to $N_{sol} \times N_{sol}$.

The crowding distance w_i corresponding to solution i is computed by a slightly modified version of the method given in [33]. The only difference is that extreme solutions are assigned an infinite (10^{30}) crowding distance. Note that this distance is only computed (required) for solutions that belong to the same Pareto front; i.e. with the same ϕ. For each front, if the front has more than two solutions, the w_i of an interior solution i is computed by

$$w_i = \sum_j^{N_f} \frac{f^i_j - f^{i-1}_j}{\delta} \times \frac{f^{i+1}_j - f^i_j}{\delta} \tag{10}$$

where $\delta = f^{max}_j - f^{min}_j + \epsilon$ and the solutions in the same front must be sorted with respect to the objective value j. The above equation aims for a good spread of solution points along the Pareto front. The reason why a large number is set to the extremities of the front is to induce an 'opening' of the front in order to cover a wider range of values.

V. Evolutionary Algorithm

The evolutionary algorithm with a differential evolution (DE) operator is presented in this section. A set (population) of trial solutions $\vec{x}$ is firstly generated and later organised into smaller and distinct groups. Within each group, trial solutions are combined in order to randomly walk in the search space aiming at finding better solutions. The pairing of solutions is randomly made after which the DE operator is applied and tournaments are carried out in order to build the next population. From time to time, solutions are exchanged between groups.

Prior to the evolution process, the set S of N_{sol} trial solutions is randomly generated. The generation considers the limits of variables; thus the search space is 'boxed'. In this work, the Latin Hypercube method is employed to this task.

The trial solutions array is mapped into N_{cpu} smaller groups G that can be independently evolved. With integers, the start s and end-plus-one E indices corresponding to each group i are simply equal to $s = i\,N_{sol}/N_{cpu}$ and $E = (i+1)\,N_{sol}/N_{cpu}$, respectively. Thus, one group i is a subset of all solutions S indicated by $G = S[s : E]$ where the notation $S[s : E]$ means a view to array S starting at s and ending at $E - 1$, inclusive. In Go, this structure is known as 'slice' and basically

is implemented with pointers to the initial and final memory locations where the S values are stored.

The word *CPU* here has a fairly general meaning and is not attached to any particular piece of hardware—in Go it represents a *goroutine* which is simply a lightweight thread of execution [42]. The way memory is shared in Go is by communication which can be done by using *channels*. The concept of channels makes easy the use of concurrency in Go. For example, to 'spawn' a thread of execution on the background the command '`go`' is simply attached to any function call (including anonymous/lambda ones) and information is passed through by the 'pipe' indicator '<−'. Below is an example of a simple parallel execution:

```
done := make(chan int, NCPU)
for icpu := 0; icpu < NCPU; icpu++ {
    go func(cpu int) {
        /* do something on the background */
        done <- 1 // flag completion
    }(icpu) // notice that this is a function call
}
for icpu := 0; icpu < NCPU; icpu++ {
    <-done // empty channel
}
```

In the above code, note that the index $icpu$ is passed during the call to the anonymous function where it becomes cpu. This is an important step because the simultaneous spawn of a thread means that $icpu$ may be already modified for the next one. More details are available in [42].

Having the initial set of trial solutions generated, the evolution process is simulated from an initial time $t = 0$ to the final time $t = t_{max}$. This happens in larger increments Δt_{exc} corresponding to the time passed between *exchange* of solutions among groups. Four major steps are followed in the main loop of the proposed algorithm. This is explained in Algorithm 4 where '%' represents the modulo operator.

After the parallel evolution, the `Metrics` function is called with all solutions in order to update the information required for the exchange of solutions. This is necessary for the execution of tournaments in particular. In this way, although chunks of solutions are grouped, their effect in the whole set is considered by means of values such as neighbouring distance and Pareto fronts. Therefore, the exchange will be biased towards the best candidates from all groups.

The parallel code works by exchanging solutions in time intervals (Δt_{exc}). In this way, each independent group has a chance to search for solutions in different spaces. The best solutions are then transferred from group to group. The proposed algorithm implements the exchange in a cyclic fashion; from groups with lower indices to groups with higher indices, except the last pair where the order of indices is reversed.

The exchange step happens by invoking a tournament with two randomly selected solutions (A, B) from one group and two randomly selected solutions (a, b) from another group. The function `Tournament`, given in the next subsection, is then called with the output being stored in the global set S (by replacement).

To further couple the influence of one group to the whole set, a random exchange of one solution between two randomly selected groups is also performed. In this case, the two solutions are simply swapped and no tournament is carried out.

Algorithm 4. `Solve()` solves the optimisation problem.

Input: configuration parameters
Output: optimal set of solutions S
$t \leftarrow 0$ `// evolution pseudo time`
$t_{exc} \leftarrow \Delta t_{exc}$ `// exchange time`
while $t < t_{max}$ **do**
  `// parallel execution up` t_{exc}
  for *each group* i **do**
    spawn goroutine with $I \leftarrow i$
      for $time \leftarrow t$ **to** $time < t_{exc}$ **do**
        `EvolveOneGroup(`I`)`
      send *done* flag through *channel*
  for *each group* **do**
    collect *done* flag from *channel*
  `// compute metrics`
  `Metrics(`S`)`
  `// exchange solutions via tournament`
  for *each group* i **do**
    $j \leftarrow (i+1) \,\%\, N_{cpu}$
    $A, B \leftarrow$ random solutions from G_i
    $a,\ b \leftarrow$ random solutions from G_j
    `Tournament(`$S,\ \ A, B, a, b$`)`
  `// exchange one solution randomly`
  select non-repeated pairs of groups
  for *each random pair* (G_0, G_1) **do**
    $A \leftarrow$ random solution from G_0
    $B \leftarrow$ random solution from G_1
    swap A by B in S
  `// update time variables`
  $t \mathrel{+}= \Delta t_{exc}$
  $t_{exc} \mathrel{+}= \Delta t_{exc}$

It is expected that this step might reduce diversity; however it might have some beneficial effect.

A. Evolution, niching and tournaments

The key step in the evolutionary optimiser is the update of trial solutions to a next generation. This is the *evolution* step and can be independently carried out for a group of solutions G. In Algorithm 4, this step is the `EvolveOneGroup` command. The crowding approach to niching explained in [10] is adopted in this work. Small modifications are applied though; in particular with intentions to tackle multi objective optimisation problems as well.

The procedure described below is carried out for each group of solutions. First, a bi-dimensional array P (pairs) with the indices of trial solutions in the group is randomly generated. The array has half the size (truncated) of the group size; thus a pair of solutions A and B will be the basis for generating new solutions a and b. For example, with 6 solutions in the group, this array looks like:

$$P = \begin{bmatrix} 0 & 2 \\ 4 & 1 \\ 3 & 5 \end{bmatrix} \tag{11}$$

The DE operator, presented later on, requires other three trial solutions $\dot{\boldsymbol{x}}_{A0}$, $\dot{\boldsymbol{x}}_{A1}$ and $\dot{\boldsymbol{x}}_{A2}$ in order to compute a new solution $\dot{\boldsymbol{x}}_a$ from $\dot{\boldsymbol{x}}_A$. Because two new solutions will be generated in this step, other three trial solutions $\dot{\boldsymbol{x}}_{B0}$, $\dot{\boldsymbol{x}}_{B1}$ and $\dot{\boldsymbol{x}}_{B2}$ are needed to compute $\dot{\boldsymbol{x}}_b$ from $\dot{\boldsymbol{x}}_B$. The eight solutions

Algorithm 5. `EvolveOneGroup`(G) evolves group using tournaments. `RealPart`($G[\alpha]$) returns the float-point part of solution α in the set of solutions G and `IntPart`($G[\alpha]$) returns its integer part (if any).

Input: group G and backup set of solutions Γ with the same size as G
Output: evolved group G

```
// find random pairs
generate table P with N_p pairs
// create new solutions
for k ← 0 to k < N_p do
    l ← (k+1) % N_p
    m ← (k+2) % N_p
    n ← (k+3) % N_p
    x_A  ← RealPart(G[P_{k,0}])
    x_A0 ← RealPart(G[P_{l,0}])
    x_A1 ← RealPart(G[P_{m,0}])
    x_A2 ← RealPart(G[P_{n,0}])
    x_B  ← RealPart(G[P_{k,1}])
    x_B0 ← RealPart(G[P_{l,1}])
    x_B1 ← RealPart(G[P_{m,1}])
    x_B2 ← RealPart(G[P_{n,1}])
    x_a ← DiffEvol(x_A, x_A0, x_A1, x_A2)
    x_b ← DiffEvol(x_B, x_B0, x_B1, x_B2)
    y_A ← IntPart(G[P_{k,0}])
    y_B ← IntPart(G[P_{k,1}])
    y_a, y_b ← Crossover(y_A, y_B)
    y_a ← Mutation(y_a)
    y_b ← Mutation(y_b)
    store {x_a, x_b, y_a, y_b} in Γ
// metrics
Metrics(G ∪ Γ)
// tournaments
for k ← 0 to k < N_p do
    A ← G[P_{k,0}]
    B ← G[P_{k,1}]
    a ← Γ[P_{k,0}]
    b ← Γ[P_{k,1}]
    Tournament(G, A, B, a, b)
```

Algorithm 6. `Fight`(A, B) performs the comparison between trial solutions for a tournament. All data such as OORs, OVAs and measures of diversity are used.

Input: trial solutions A and B
Output: returns **true** if A wins

```
// compare solutions using OOR and OVA
A_dom, B_dom ← CompareSolutions(A, B)
if A_dom then return true
if B_dom then return false
// tie: single-OVA
if N_f = 1 then
    if η_A > η_B then return true
    if η_B > η_A then return false
    return FlipCoin(0.5)
// tie: multi-OVA: same Pareto front
if φ_A = φ_B then
    if w_A > w_B then return true
    if w_B > w_A then return false
    return FlipCoin(0.5)
// tie: multi-OVA: different fronts
if φ_A < φ_B then return true
if φ_B < φ_A then return false
if η_A > η_B then return true
if η_B > η_A then return false
// tie: nothing else can be done
return FlipCoin(0.5)
```

are picked from G using a cyclic permutation of indices in P as indicated in Algorithm 5. In this algorithm, a classical genetic operation on integers $\underset{\cdot}{\boldsymbol{y}}$ is performed as well.

It is worth noting that the way the P array is designed aims for accommodating conventional genetic operators for integers as well as differential evolution recombinations. To clarify, if DE would be the operator alone, it would not matter how the eight solutions were selected, as long as the process is random and involved unique candidates. Nonetheless, the 'pairing' of solutions helps especially when mixing integers and float point numbers encodings.

As indicated in Algorithm 5, new solutions are created by employing DE to real variables and conventional GA to integers. The resulting solutions indicated by a and b are stored in a backup set represented by Γ. After that, the `Metrics` function is called with the union of the old (G) and new populations (Γ). In this way, the information required for tournaments such as the Pareto front, neighbouring and crowding distances and limits will consider the presence of new solutions in the space of solutions. This feature, for instance, will induce new solutions to be less crowded when finally added to the next generation.

Up to this point, the set G is unmodified. The `Tournament` function is the one in charge of deciding whether or not new solutions a and b will replace the old ones A and B. The tournament starts by firstly matching the closest pairs of old (A,B) and new (a,b) solutions. The neighbouring distance d_{ij} (Section IV) between solutions is employed to this task. The pair of competitors is then

$$\begin{array}{ll} \{A,\, a\} \text{ and } \{B,\, b\} & \text{if } d_{Aa} + d_{Bb} < d_{Ab} + d_{Ba} \\ \{A,\, b\} \text{ and } \{B,\, a\} & \text{otherwise} \end{array} \tag{12}$$

This strategy is essential to the crowding approach to niching; see e.g. [10]. In this way, a new solution will compete against the closest match; if it wins, it will replace the old solution in set G; otherwise, the corresponding item in G will remain unchanged.

The `Tournament` function is quite straightforward; it simply requires a function called `Fight` where two solutions will 'fight' each other. It is in this last function that the improvement of diversity is effectively implemented. Although the comparison function `CompareSolutions` may return an indecisive response (non-dominance), the `Fight` function must return **true** or **false** deciding whether A wins or not; even if the flipping of a fair coin has to be carried out. This function is listed in Algorithm 6 where four additional branches are considered in case there is a tie between solutions A and B; i.e. A and B produce the same OOR and OVA at the same time. The `Tournament` function is given in Algorithm 7.

If there is a tie in a single objective problem, the solution having the largest neighbouring distance η wins; hence inducing diversity. In multiple objectives problems, two cases arise: (1) both solutions are in the same Pareto front—in this case the one with the largest crowding distance wins; and (2) the

Algorithm 7. `Tournament`$(\Omega,\ A, B, a, b)$ performs the tournament with replacement of winners in Ω.

Input: trial solutions A, B, a, b
Output: Ω: replace A or B, or both, in the right position in the set Ω
compute distances d_{Aa}, d_{Ab}, d_{Ba} and d_{Bb}
if $d_{Aa} + d_{Bb} < d_{Ab} + d_{Ba}$ **then**
  if not `Fight`(A, a) **then** // a wins over A
    replace A with a in Ω
  if not `Fight`(B, b) **then** // b wins over B
    replace B with b in Ω
else
  if not `Fight`(A, b) **then** // b wins over A
    replace A with b in Ω
  if not `Fight`(B, a) **then** // a wins over B
    replace B with a in Ω

Algorithm 8. `DiffEvol`$(\overset{\cdot}{\boldsymbol{x}}, \overset{\cdot}{\boldsymbol{x}}_0, \overset{\cdot}{\boldsymbol{x}}_1, \overset{\cdot}{\boldsymbol{x}}_2)$ generates a new solution using the differential evolution operator.

Input: trial solutions $\overset{\cdot}{\boldsymbol{x}}, \overset{\cdot}{\boldsymbol{x}}_0, \overset{\cdot}{\boldsymbol{x}}_1, \overset{\cdot}{\boldsymbol{x}}_2$
Output: new solution $\overset{\cdot}{\boldsymbol{x}}^{new}$
$F \leftarrow$ `RealRand`$(0, 1)$
$I \leftarrow$ `IntRand`$(0, N_x - 1)$
for $i \leftarrow 0$ **to** $i < N_x$ **do**
  if `FlipCoin`(C_{DE}) or $i = I$ **then**
    $x_i^{\text{new}} = x_{0,i} + F\,(x_{1,i} - x_{2,i})$
    if $x_i^{\text{new}} < x_i^L$ **then**
      $x_i^{\text{new}} \leftarrow x_i^L$
    if $x_i^{\text{new}} > x_i^U$ **then**
      $x_i^{\text{new}} \leftarrow x_i^U$
  else
    $x_i^{\text{new}} \leftarrow x_i$

solutions are in different fronts—in this case, first, the one with the smallest Pareto front rank ϕ wins; second, the one with the largest neighbouring distance η wins. Again, this strategy aims to induce diversity and seemed to work quite well. In the end, if the two solutions are really equal one with another, the decision is made by a Bernoulli variable.

B. Differential evolution

The differential evolution [38], [39] method is selected to combine real numbers. Other methods are available such as the SBX [34]; however some experiments with the proposed code demonstrated that DE is more efficient. This has been partially observed by others as well [33]. Furthermore, the DE produces the same results after every run (good repeatability). Actually, as demonstrated later on, the repeatability behaviour with DE is impressive.

As mentioned earlier, three auxiliary solutions are needed for computing the new solution using the DE operator. Each component of the new solution will either be the unmodified component of the trial solution or a combination of the components of the other three auxiliary solutions. The combination is known as the *mutation* step [39] and is accomplished by means of

$$x_{\text{new},i} = x_{0,i} + F\,(x_{1,i} - x_{2,i}) \tag{13}$$

where F is a scaling factor that induces a displacement of the trial solution along a random direction. In the implementation proposed here, F is an uniform random variable in $[0, 1)$. This strategy seemed to be the best to attack many single and multi objective problems at the same time. The DE requires a second parameter C_{DE} that controls the probability of one component of the solution vector $\overset{\cdot}{\boldsymbol{x}}$ being modified or not.

The steps required for the differential evolution operator are collected in Algorithm 8. Note that after the mutation step, components of the solution vector may be outside the limits. In this situation, the values are simply truncated to the limiting ones. This effectively implements a *box approach* for constraining the unknown variables $\overset{\cdot}{\boldsymbol{x}}$.

VI. Test cases: constrained one-objective

This section presents numerical solutions with the intentions of verifying and demonstrating the limitations and capabilities of the proposed code. First, some single objective nonlinear optimisation problems with many constraints are studied. The next section presents a study of unconstrained two-objective problems and Part II [12] presents constrained multi/many-objective problems and some applications.

Attention is given in particular to the repeatability of results. Therefore, each one of the problems is run 1000 times and the statistics of results are discussed. The expected and best reference values from the literature are listed alongside the results in the output tables. Note that the limits of variables are also constraints; but not counted like so in the following discussion.

The next 9 problems are more or less listed in order of difficulty faced by goga. Other codes may encounter differences. The first 7 problems are satisfactorily solved with less than 500 iterations; actually the first ones could be solved with 100-200 iterations. Nonetheless, the number is fixed to $t_{max} = 500$ in order to generate an initial set of results covering all problems; without much tweaking of parameters. Later, problem 9 is individually re-analysed with more iterations.

Problem 1 is a simple problem presented in [18] with 2 variables (unknowns). It has 2 inequalities and is specified by

$$\begin{aligned} f_0 &= (x_0^2 + x_1 - 11)^2 + (x_0 + x_1^2 - 7)^2 \\ g_0 &= 4.84 - (x_0 - 0.05)^2 - (x_1 - 2.5)^2 \\ g_1 &= x_0^2 + (x_1 - 2.5)^2 - 4.84 \end{aligned} \tag{14}$$

in the search space $0 \le x_i \le 6$ $(i = 0, 1)$. This example was particularly challenging for earlier codes as discussed in [18]. This problem is relative easy although the solution space is a narrow band as illustrated in Fig. 3. The solutions are illustrated in this figure where the best candidate is marked with a red star. A statistical analysis is presented just after all problems have been described.

Problem 2 has 13 variables, 9 inequalities and is relatively easy because it involves only quadratic terms in the objective function and has linear constraints. It corresponds to *case 1*

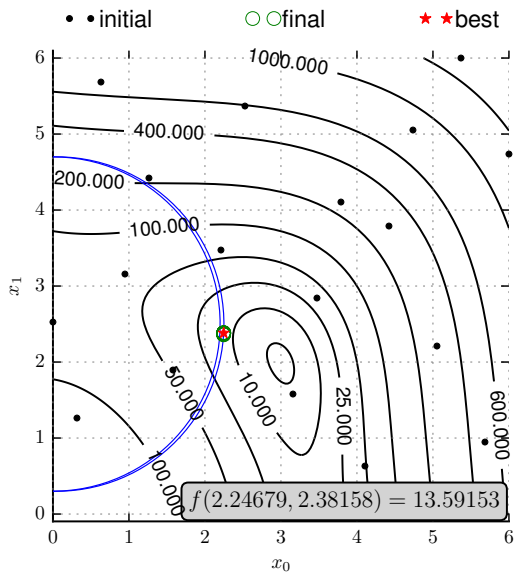

Fig. 3. Problem 1: Narrow crescent-shaped region with equality constraint.

in [14] and *test 3* in [18] and is defined by

$$\begin{aligned}
f_0 &= 5\left(\textstyle\sum_{i=0}^{3} x_i - \sum_{i=0}^{3} x_i^2\right) - \sum_{i=4}^{12} x_i \\
g_0 &= 10 - 2\,x_0 - 2\,x_1 - x_9 - x_{10} \\
g_1 &= 10 - 2\,x_0 - 2\,x_2 - x_9 - x_{11} \\
g_2 &= 10 - 2\,x_1 - 2\,x_2 - x_{10} - x_{11} \\
g_3 &= 8\,x_0 - x_9 \\
g_4 &= 8\,x_1 - x_{10} \\
g_5 &= 8\,x_2 - x_{11} \\
g_6 &= 2\,x_3 + x_4 - x_9 \\
g_7 &= 2\,x_5 + x_6 - x_{10} \\
g_8 &= 2\,x_7 + x_8 - x_{11}
\end{aligned} \tag{15}$$

where the limits of variables are

$$\begin{aligned}
0 \le x_i &\le 1 \quad (i \in [0, 8]) \\
0 \le x_i &\le 100 \quad (i \in [9, 11]) \\
0 \le x_{12} &\le 1
\end{aligned} \tag{16}$$

Problem 3 has 5 variables, 6 inequalities and has normal difficulty since it involves a quadratic objective function and has quadratic constraint functions. It corresponds to *problem 83 (page 102)* in [43] and *test 6* in [18] and is defined by

$$\begin{aligned}
f_0 &= 5.3578547\,x_2^2 + 0.8356891\,x_0\,x_4 + 37.293239\,x_0 \\
&\quad - 40792.141 \\
g_0 &= c_0 \\
g_1 &= 92 - c_0 \\
g_2 &= c_1 - 90 \\
g_3 &= 110 - c_1 \\
g_4 &= c_2 - 20 \\
g_5 &= 25 - c_2
\end{aligned} \tag{17}$$

where the c_i coefficients are given by

$$\begin{aligned}
c_0 &= 85.334407 + 0.0056858\,x_1\,x_4 + 0.0006262\,x_0\,x_3 \\
&\quad - 0.0022053\,x_2\,x_4 \\
c_1 &= 80.51249 + 0.0071317\,x_1\,x_4 + 0.0029955\,x_0\,x_1 \\
&\quad + 0.0021813\,x_2\,x_2 \\
c_2 &= 9.300961 + 0.0047026\,x_2\,x_4 + 0.0012547\,x_0\,x_2 \\
&\quad + 0.0019085\,x_2\,x_3
\end{aligned} \tag{18}$$

and the limits of variables are

$$\begin{aligned}
78 \le x_0 &\le 102 \\
33 \le x_1 &\le 45 \\
27 \le x_i &\le 45 \quad (i \in [2, 4])
\end{aligned} \tag{19}$$

Problem 4 has 5 variables, 38 inequalities and is of moderate difficulty because it involves a large number of nonlinear constraints up to the second order. It corresponds to *problem 85 (page 104)* in [43] and *test 2* in [18] and is defined by

$$\begin{aligned}
f_0 &= -5.843 \cdot 10^{-7}\,y_{16} + 1.17 \cdot 10^{-4}\,y_{13} \\
&\quad + 2.358 \cdot 10^{-5}\,y_{12} + 1.502 \cdot 10^{-6}\,y_{15} \\
&\quad + 0.0321\,y_{11} + 0.004324\,y_4 + 1.0e-4\,c_{14}/c_{15} \\
&\quad + 37.48\,y_1/c_{11} + 0.1365 \\
g_0 &= 1.5\,x_1 - x_2 \\
g_1 &= y_0 - 213.1 \\
g_2 &= 405.23 - y_0 \\
g_{i+2} &= y_i - a_i \quad (i \in [1, 16]) \\
g_{i+18} &= b_i - y_i \quad (i \in [1, 16]) \\
g_{35} &= y_3 - 0.28/0.72\,y_4 \\
g_{36} &= 21 - 3496\,y_1/c_{11} \\
g_{37} &= 62212/c_{16} - 110.6 - y_0
\end{aligned} \tag{20}$$

Where the a and b coefficients are

$$\begin{aligned}
a = \{&0,\ 17.505,\ 11.275,\ 214.228,\ 7.458,\ 0.961,\ 1.612, \\
&0.146,\ 107.99,\ 922.693,\ 926.832,\ 18.766,\ 1072.163, \\
&8961.448,\ 0.063,\ 71084.33,\ 2802713\}
\end{aligned} \tag{21}$$

and

$$\begin{aligned}
b = \{&0,\ 1053.6667,\ 35.03,\ 665.585,\ 584.463,\ 265.916,\ 7.046, \\
&0.222,\ 273.366,\ 1286.105,\ 1444.046,\ 537.141,\ 3247.039, \\
&26844.086,\ 0.386,\ 140000,\ 12146108\}
\end{aligned} \tag{22}$$

Therein, a number of auxiliary variables have to be calculated

in the following order

$$
\begin{aligned}
y_0 &= x_1 + x_2 + 41.6 \\
c_0 &= 0.024\,x_3 - 4.62 \\
y_1 &= 12.5/c_0 + 12 \\
c_1 &= 0.0003535\,x_0\,x_0 + 0.5311\,x_0 + 0.08705\,y_1\,x_0 \\
c_2 &= 0.052\,x_0 + 78 + 0.002377\,y_1\,x_0 \\
y_2 &= c_1/c_2 \\
y_3 &= 19\,y_2 \\
c_3 &= 0.04782\,(x_0 - y_2) + 0.1956\,(x_0 - y_2)\,(x_0 - y_2)/x_1 \\
&\quad + 0.6376\,y_3 + 1.594\,y_2 \\
c_4 &= 100\,x_1 \\
c_5 &= x_0 - y_2 - y_3 \\
c_6 &= 0.95 - c_3/c_4 \\
y_4 &= c_5\,c_6 \\
y_5 &= x_0 - y_4 - y_3 - y_2 \\
c_7 &= 0.995\,(y_3 + y_4) \\
y_6 &= c_7/y_0 \\
y_7 &= c_7/3798 \\
c_8 &= y_6 - 0.0663\,y_6/y_7 - 0.3153 \\
y_8 &= 96.82/c_8 + 0.321\,y_0 \\
y_9 &= 1.29\,y_4 + 1.258\,y_3 + 2.29\,y_2 + 1.71\,y_5 \\
y_{10} &= 1.71\,x_0 - 0.452\,y_3 + 0.58\,y_2 \\
c_9 &= 12.3/752.3 \\
c_{10} &= 1.75\,y_1\,0.995\,x_0 \\
c_{11} &= 0.995\,y_9 + 1998 \\
y_{11} &= c_9\,x_0 + c_{10}/c_{11} \\
y_{12} &= c_{11} - 1.75\,y_1 \\
y_{13} &= 3623 + 64.4\,x_1 + 58.4\,x_2 + 146312/(y_8 + x_4) \\
c_{12} &= 0.995\,y_9 + 60.8\,x_1 + 48\,x_3 - 0.1121\,y_{13} - 5095 \\
y_{14} &= y_{12}/c_{12} \\
y_{15} &= 148000 - 331000\,y_{14} + 40\,y_{12} - 61\,y_{14}\,y_{12} \\
c_{13} &= 2324\,y_9 - 28740000\,y_1 \\
y_{16} &= 14130000 - 1328\,y_9 - 531\,y_{10} + c_{13}/c_{11} \\
c_{14} &= y_{12}/y_{14} - y_{12}/0.52 \\
c_{15} &= 1.104 - 0.72\,y_{14} \\
c_{16} &= y_8 + x_4
\end{aligned}
\tag{23}
$$

In problem 4, the limits of variables are

$$
\begin{aligned}
704.4148 &\le x_0 \le 906.3855 \\
68.6 &\le x_1 \le 288.88 \\
0.0 &\le x_2 \le 134.75 \\
193.0 &\le x_3 \le 287.0966 \\
25.0 &\le x_4 \le 84.1988
\end{aligned}
\tag{24}
$$

Problem 5 has 4 variables, 5 inequalities and is of moderate difficulty because of the cubic expressions in the objective and constraint functions. It corresponds to the design of a welded beam firstly presented in [44] (see also [45]) where the objective function is the system cost. As shown in [45] (page 592), after substitutions, the problem is defined by

$$
\begin{aligned}
f_0 &= 1.10471\,x_0^2\,x_1 + 0.04811\,x_2\,x_3\,(14 + x_1) \\
g_0 &= \frac{\tau_d}{F} - \left\{ \frac{1}{2\,x_0^2\,x_1^2} + \frac{3\,(28 + x_1)}{x_0^2\,x_1\,c_1} \right. \\
&\quad \left. + \frac{4.5\,(28 + x_1)^2\,[x_1^2 + (x_0 + x_2)^2]}{x_0^2\,x_1^2\,c_1^2} \right\}^{1/2} \\
g_1 &= x_2\,x_2\,x_3 - 12.8 \\
g_2 &= x_3 - x_0 \\
g_3 &= x_2\,x_3^3\,(1 - 0.02823\,x_2) - 0.09267 \\
g_4 &= x_2^3\,x_3 - 8.7808
\end{aligned}
\tag{25}
$$

where $F = 6000$, $\tau_d = 13600$, $c_1 = x_1^2 + 3\,(x_0 + x_2)^2$. The limits of variables selected here are

$$
\begin{aligned}
0.125 &\le x_0 \le 10 \\
0.0 &\le x_1 \le 10 \\
0.0 &\le x_2 \le 10 \\
0.125 &\le x_3 \le 10
\end{aligned}
\tag{26}
$$

Problem 6 has 7 variables, 4 inequalities and is of moderate difficulty with a fourth order term in the objective function and nonlinear constraints. It corresponds to *problem 100 (page 111)* in [43], *case 3* in [14] and *test 5* in [18]. The problem is defined by

$$
\begin{aligned}
f_0 &= (x_0 - 10)^2 + 5\,(x_1 - 12)^2 + x_2^4 + 3\,(x_3 - 11)^2 \\
&\quad + 10\,x_4^6 + 7\,x_5^2 + x_6^4 - 4\,x_5\,x_6 - 10\,x_5 - 8\,x_6 \\
g_0 &= 127 - 2\,x_0\,x_0 - 3\,x_1^4 - x_2 - 4\,x_3\,x_3 - 5\,x_4 \\
g_1 &= 282 - 7\,x_0 - 3\,x_1 - 10\,x_2\,x_2 - x_3 + x_4 \\
g_2 &= 196 - 23\,x_0 - x_1\,x_1 - 6\,x_5\,x_5 + 8\,x_6 \\
g_3 &= -4\,x_0\,x_0 - x_1\,x_1 + 3\,x_0\,x_1 - 2\,x_2\,x_2 \\
&\quad - 5\,x_5 + 11\,x_6
\end{aligned}
\tag{27}
$$

According to [43], there are no bounds for the unknowns. Nonetheless, the search space is defined as suggested in [14], [18]. Thus, the limits for variables are

$$
-10 \le x_i \le 10 \quad (i \in [0, 6])
\tag{28}
$$

Problem 7 has 10 variables, 8 inequalities and is also of moderate difficulty with quadratic terms and nonlinear constraints. It corresponds to *problem 113 (page 122)* in [43], *case 5* in [14] and *test 8* in [18]. The problem is defined by

$$
\begin{aligned}
f_0 &= x_0^2 + x_1^2 + x_0\,x_1 - 14\,x_0 - 16\,x_1 + (x_2 - 10)^2 \\
&\quad + 4\,(x_3 - 5)^2 + (x_4 - 3)^2 + 2\,(x_5 - 1)^2 + 5\,x_6^2 \\
&\quad + 7\,(x_7 - 11)^2 + 2\,(x_8 - 10)^2 + (x_9 - 7)^2 + 45 \\
g_0 &= 105 - 4\,x_0 - 5\,x_1 + 3\,x_6 - 9\,x_7 \\
g_1 &= -10\,x_0 + 8\,x_1 + 17\,x_6 - 2\,x_7 \\
g_2 &= 8\,x_0 - 2\,x_1 - 5\,x_8 + 2\,x_9 + 12 \\
g_3 &= -3\,(x_0 - 2)^2 - 4\,(x_1 - 3)^2 - 2\,x_2^2 + 7\,x_3 + 120 \\
g_4 &= -5\,x_0^2 - 8\,x_1 - (x_2 - 6)^2 + 2\,x_3 + 40 \\
g_5 &= -0.5\,(x_0 - 8)^2 - 2\,(x_1 - 4)^2 - 3\,x_4^2 + x_5 + 30 \\
g_6 &= -x_0^2 - 2\,(x_1 - 2)^2 + 2\,x_0\,x_1 - 14\,x_4 + 6\,x_5 \\
g_7 &= 3\,x_0 - 6\,x_1 - 12\,(x_8 - 8)^2 + 7\,x_9
\end{aligned}
\tag{29}
$$

As in the previous problem, there are no bounds for the unknowns. Thus, the following limits as in [14], [18] are adopted

$$-10 \le x_i \le 10 \quad (i \in [0,9]) \tag{30}$$

Problem 8 has 8 variables, 6 inequalities and is a difficult problem as observed in [15], [18]. It has a linear objective function and nonlinear constraints and corresponds to *problem 106 (page 115; heat exchanger design)* in [43], *case 2* in [14] and *test 4* in [18]. The problem is defined by

$$\begin{aligned}
f_0 &= x_0 + x_1 + x_2 \\
g_0 &= 1 - 0.0025\,(x_3 + x_5) \\
g_1 &= 1 - 0.0025\,(x_4 + x_6 - x_3) \\
g_2 &= 1 - 0.01\,(x_7 - x_4) \\
g_3 &= x_0\,x_5 - 833.33252\,x_3 - 100\,x_0 + 83333.333 \\
g_4 &= x_1\,x_6 - 1250\,x_4 - x_1\,x_3 + 1250\,x_3 \\
g_5 &= x_2\,x_7 - x_2\,x_4 + 2500\,x_4 - 1250000
\end{aligned} \tag{31}$$

where the limits of variables are

$$\begin{aligned}
100 \le x_0 &\le 10000 \\
1000 \le x_i &\le 10000 \quad (i \in [1,2]) \\
10 \le x_i &\le 1000 \quad (i \in [3,7])
\end{aligned} \tag{32}$$

Problem 9 has 5 variables, 3 equality constraints and is a very difficult problem. The objective function is the exponential of all variables multiplied together. The equality constraints are nonlinear (quadratic and cubic) expressions. To handle these constraints $\epsilon_h = 10^{-3}$ is selected. The problem corresponds to *problem 80 (page 100)* in [43], *case 4* in [14] and *test 7* in [18]. The problem is defined by

$$\begin{aligned}
f_0 &= \exp(x_0\,x_1\,x_2\,x_3\,x_4) \\
h_0 &= x_0^2 + x_1^2 + x_2^2 + x_3^2 + x_4^2 - 10 \\
h_1 &= x_1\,x_2 - 5\,x_3\,x_4 \\
h_2 &= x_0^3 + x_1^3 + 1
\end{aligned} \tag{33}$$

where the limits of variables are

$$\begin{aligned}
-2.3 \le x_i &\le 2.3 \quad (i \in [0,1]) \\
-3.2 \le x_i &\le 3.2 \quad (i \in [2,4])
\end{aligned} \tag{34}$$

The nine problems are solved 1000 times (samples) each with $t_{max} = 500$ iterations. After one sample is finished, all values are initialised, the population is randomly re-created, and the evolution is run all over again. The results are collected in Table I alongside other input data and histograms of computed objective values. In this table, T_{sys} is the computer (system) time measured during the complete analysis of all samples, including the 1000×500 loops. The resulting number of function evaluations *per* sample is N_{eval}; calling f_i, g_i, and h_i at the same time means 1 evaluation. Below the histogram, *count* refers to the number of feasible solutions in the final set. The computed solutions are indicated by X_{best} and the reference ones by $X_{ref.}$ and f_{ref}.

In these experiments, the number of trial solutions N_{sol} is fixed and calculated as $10\,N_x$; i.e. ten times the number of variables as in [18]. Sometimes more solutions are needed and sometimes the problem could be easily solved with less; but this has not been attempted. It is worth noting also that the number of groups (N_{cpu}) is selected to reduce a little the number of solutions in the same group (no more than 50). When needed, the exchange time Δt_{exc} is simply computed by means of $\Delta t_{exc} = t_f/10$.

An important parameter is the differential evolution coefficient C_{DE} (indicated in all tables with results). By default, this value is fixed as $C_{DE} = 0.8$. Later on, the value is changed for some multi and many objective tests whenever the performance can be improved.

The code works really well for problems 1 to 7 with the standard deviation f_{dev} being very small. This means that every time the code is run the same solution is obtained; except by 7 out of 1000 non optimal results in problem 2 (to be discussed soon next). In problem 7, the results are not bad; but there is a small standard deviation and a slightly larger spread in the histogram. In these problems (1-7), minimum values (f_{min}) close to the reference ones are obtained. Problem 8 is more difficult to be solved and will require longer evolution times.

In problem 2, the failure happens because the optimal solution is actually at the boundaries of the search space; i.e. the solution reaches the maximum limit. This can be remediated by employing a larger search space and adding extra g_i constraints as follows

$$\begin{aligned}
g_{9+i} &= x_i && (i \in [0,8]) \\
g_{18+i} &= 1 - x_i && (i \in [0,8]) \\
g_{27+i} &= x_{9+i} && (i \in [0,3]) \\
g_{30+i} &= 100 - x_{9+i} && (i \in [0,3]) \\
g_{33} &= x_{12} \\
g_{34} &= 1 - x_{12}
\end{aligned} \tag{35}$$

Thus, the number of constraints increases to $N_f = 35$. The limits of variables defining the search space can now be specified as

$$\begin{aligned}
-0.5 \le x_i &\le 1.5 \quad (i \in [0,8]) \\
-1.0 \le x_i &\le 101 \quad (i \in [9,11]) \\
-0.5 \le x_{12} &\le 1.5
\end{aligned} \tag{36}$$

Note that this is a natural way to define the constrained optimisation problem. The convenience of the *box approach* where the unknowns are always limited is apparent in reducing the number of constraint functions. The problem is run again and the minimum is always obtained. The results are listed in Table II and are quite good now.

Problem 9 requires longer times and is actually a challenging one. The reason this problem is difficult is the five multiplications $x_0\,x_1\,x_2\,x_3\,x_4$ inside the exponential function in (33) and the 'narrow' equality constraints. This means that the sign of pairs $x_i\,x_j$ $(i \neq j)$ can vary freely causing problems to the differential evolution operator. Moreover, the equality constraint makes it harder to find feasible solutions because the search space is largely reduced to fall within the band specified by ϵ_h.

TABLE I
CONSTRAINED SINGLE OBJECTIVE PROBLEMS.

P	settings	settings/info	objective	histogram ($N_{samples} = 1000$)
1	$N_{sol} = 20$ $N_{cpu} = 1$ $t_{max} = 500$ $\Delta t_{exc} = 1$	$f_{ref} = \mathbf{13.5908500}$ $C_{DE} = 0.8$ $N_{eval} = 10020$ $T_{sys} = 25.551s$	$f_{min} = \mathbf{13.5908417}$ $f_{ave} = 13.5908417$ $f_{max} = 13.5908418$ $f_{dev} = \mathbf{3.067 \cdot 10^{-9}}$	[13.54,13.56) | 0 [13.56,13.57) | 0 [13.57,13.58) | 0 [13.58,13.60) | 1000 #################### [13.60,13.61) | 0 [13.61,13.63) | 0 [13.63,13.64) | 0 count = 1000
			X_{best}=[2.2468258 2.3818635] $X_{ref.}$=[2.2468260 2.3818650]	
2	$N_{sol} = 130$ $N_{cpu} = 4$ $t_{max} = 500$ $\Delta t_{exc} = 50$	$f_{ref} = \mathbf{-15.0000000}$ $C_{DE} = 0.8$ $N_{eval} = 64130$ $T_{sys} = 3m26.888s$	$f_{min} = \mathbf{-15.0000000}$ $f_{ave} = -14.9789579$ $f_{max} = -12.0000000$ $f_{dev} = \mathbf{2.505 \cdot 10^{-1}}$	[-15.05,-14.61) | 991 #################### [-14.61,-14.16) | 0 [-14.16,-13.72) | 0 [-13.72,-13.28) | 0 [-13.28,-12.84) | 0 [-12.84,-12.39) | 0 [-12.39,-11.95) | 7 # count = 998
			X_{best}=[1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 3.0000000 3.0000000 3.0000000 1.0000000] $X_{ref.}$=[1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 3.0000000 3.0000000 3.0000000 1.0000000]	
3	$N_{sol} = 50$ $N_{cpu} = 1$ $t_{max} = 500$ $\Delta t_{exc} = 1$	$f_{ref} = \mathbf{-30665.500}$ $C_{DE} = 0.8$ $N_{eval} = 25050$ $T_{sys} = 3m11.965s$	$f_{min} = \mathbf{-30665.539}$ $f_{ave} = -30665.539$ $f_{max} = -30665.539$ $f_{dev} = \mathbf{1.247 \cdot 10^{-12}}$	[-30665.59,-30665.57) | 0 [-30665.57,-30665.56) | 0 [-30665.56,-30665.55) | 0 [-30665.55,-30665.53) | 1000 #################### [-30665.53,-30665.52) | 0 [-30665.52,-30665.50) | 0 [-30665.50,-30665.49) | 0 count = 1000
			X_{best}=[78.0000000 33.0000000 29.9952560 45.0000000 36.7758129] $X_{ref.}$=[78.0000000 33.0000000 29.9950000 45.0000000 36.7760000]	
4	$N_{sol} = 50$ $N_{cpu} = 1$ $t_{max} = 500$ $\Delta t_{exc} = 1$	$f_{ref} = \mathbf{-1.9051338}$ $C_{DE} = 0.8$ $N_{eval} = 25050$ $T_{sys} = 3m26.662s$	$f_{min} = \mathbf{-1.9051553}$ $f_{ave} = -1.9051553$ $f_{max} = -1.9051553$ $f_{dev} = \mathbf{2.837 \cdot 10^{-10}}$	[-1.96,-1.94) | 0 [-1.94,-1.93) | 0 [-1.93,-1.91) | 0 [-1.91,-1.90) | 1000 #################### [-1.90,-1.88) | 0 [-1.88,-1.87) | 0 [-1.87,-1.85) | 0 count = 1000
			X_{best}=[705.1745371 68.6000000 102.9000000 282.3249316 37.5841164] $X_{ref.}$=[705.1803000 68.6000500 102.9000100 282.3249990 37.5850413]	
5	$N_{sol} = 40$ $N_{cpu} = 1$ $t_{max} = 500$ $\Delta t_{exc} = 1$	$f_{ref} = \mathbf{2.3402700}$ $C_{DE} = 0.8$ $N_{eval} = 20040$ $T_{sys} = 1m48.341s$	$f_{min} = \mathbf{2.3402145}$ $f_{ave} = 2.3402145$ $f_{max} = 2.3402145$ $f_{dev} = \mathbf{1.053 \cdot 10^{-9}}$	[2.29,2.30) | 0 [2.30,2.32) | 0 [2.32,2.33) | 0 [2.33,2.35) | 1000 #################### [2.35,2.36) | 0 [2.36,2.38) | 0 [2.38,2.39) | 0 count = 1000
			X_{best}=[0.2536388 7.1415452 7.1039050 0.2536388] $X_{ref.}$=[0.2536000 7.1410000 7.1044000 0.2536000]	
6	$N_{sol} = 70$ $N_{cpu} = 2$ $t_{max} = 500$ $\Delta t_{exc} = 50$	$f_{ref} = \mathbf{680.6300573}$ $C_{DE} = 0.8$ $N_{eval} = 34070$ $T_{sys} = 2m54.944s$	$f_{min} = \mathbf{680.6300606}$ $f_{ave} = 680.6302939$ $f_{max} = 680.6390398$ $f_{dev} = \mathbf{4.582 \cdot 10^{-4}}$	[680.58,680.60) | 0 [680.60,680.61) | 0 [680.61,680.63) | 0 [680.63,680.64) | 1000 #################### [680.64,680.66) | 0 [680.66,680.67) | 0 [680.67,680.69) | 0 count = 1000
			X_{best}=[2.3305500 1.9514279 -0.4771279 4.3655307 -0.6242891 1.0379862 1.5941325] $X_{ref.}$=[2.3304990 1.9513720 -0.4775414 4.3657260 -0.6244870 1.0381310 1.5942270]	
7	$N_{sol} = 100$ $N_{cpu} = 2$ $t_{max} = 500$ $\Delta t_{exc} = 50$	$f_{ref} = \mathbf{24.3062091}$ $C_{DE} = 0.8$ $N_{eval} = 50100$ $T_{sys} = 8m42.495s$	$f_{min} = \mathbf{24.3141038}$ $f_{ave} = 24.3382487$ $f_{max} = 24.4374485$ $f_{dev} = \mathbf{0.0152218}$	[24.26,24.30) | 0 [24.30,24.33) | 237 ####### [24.33,24.36) | 689 #################### [24.36,24.39) | 60 ## [24.39,24.42) | 11 # [24.42,24.46) | 3 # [24.46,24.49) | 0 count = 1000
			X_{best}=[2.1674985 2.3749150 8.7711338 5.0848566 1.0012935 1.4505353 1.3128994 9.8215013 8.2634398 8.3639281] $X_{ref.}$=[2.1719960 2.3636830 8.7739260 5.0959840 0.9906548 1.4305740 1.3216440 9.8287260 8.2800920 8.3759270]	
8	$N_{sol} = 80$ $N_{cpu} = 2$ $t_{max} = 500$ $\Delta t_{exc} = 50$	$f_{ref} = \mathbf{7049.3309230}$ $C_{DE} = 0.8$ $N_{eval} = 40080$ $T_{sys} = 3m57.211s$	$f_{min} = \mathbf{7059.6392776}$ $f_{ave} = 7207.8073495$ $f_{max} = 7640.4986437$ $f_{dev} = \mathbf{73.3615349}$	[7059.59,7142.58) | 226 ########## [7142.58,7225.58) | 282 ############# [7225.58,7308.57) | 460 #################### [7308.57,7391.57) | 17 # [7391.57,7474.56) | 10 # [7474.56,7557.55) | 2 # [7557.55,7640.55) | 2 # count = 999
			X_{best}=[566.8809503 1384.7701939 5107.9881334 180.6658815 295.8315595 219.2099133 284.6950948 395.7883219] $X_{ref.}$=[579.3167000 1359.9430000 5110.0710000 182.0174000 295.5985000 217.9799000 286.4162000 395.5979000]	
9	$N_{sol} = 50$ $N_{cpu} = 1$ $t_{max} = 500$ $\Delta t_{exc} = 1$	$f_{ref} = \mathbf{0.0539498}$ $C_{DE} = 0.8$ $N_{eval} = 25050$ $T_{sys} = 3m14.833s$	$f_{min} = \mathbf{0.0538671}$ $f_{ave} = 0.2551282$ $f_{max} = 1.0057637$ $f_{dev} = \mathbf{0.2567274}$	[0.00,0.15) | 274 #################### [0.15,0.30) | 100 ######## [0.30,0.45) | 34 ### [0.45,0.61) | 39 ### [0.61,0.76) | 17 ## [0.76,0.91) | 20 ## [0.91,1.06) | 23 ## count = 507
			X_{best}=[-1.7172233 1.5959302 1.8271190 -0.7647451 -0.7628557] $X_{ref.}$=[-1.7171430 1.5957090 1.8272470 -0.7636413 -0.7636450]	

TABLE II
CONSTRAINED SINGLE OBJECTIVE PROBLEMS 2 AND 9

P	settings	settings/info	objective	histogram ($N_{trials} = 1000$)
2	$N_{sol} = 130$	$f_{ref} = \mathbf{-15.0000000}$	$f_{min} = \mathbf{-14.9997587}$	[-15.05,-15.04) | 0 [-15.04,-15.02) | 0 [-15.02,-15.01) | 0 [-15.01,-14.99) | 1000 #################### [-14.99,-14.98) | 0 [-14.98,-14.96) | 0 [-14.96,-14.95) | 0 count = 1000
	$N_{cpu} = 4$	$C_{DE} = 0.8$	$f_{ave} = -14.9990503$	
	$t_{max} = 500$	$N_{eval} = 64130$	$f_{max} = -14.9979020$	
	$\Delta t_{exc} = 50$	$T_{sys} = 3m33.091s$	$f_{dev} = \mathbf{2.679 \cdot 10^{-4}}$	
		X_{best}=[1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 3.000 3.000 3.000 1.000]		
		$X_{ref.}$=[1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 3.000 3.000 3.000 1.000]		
9	$N_{sol} = 50$	$f_{ref} = \mathbf{0.0539498}$	$f_{min} = \mathbf{0.0538666}$	[0.00,0.15) | 931 #################### [0.15,0.30) | 0 [0.30,0.45) | 58 ## [0.45,0.60) | 0 [0.60,0.75) | 0 [0.75,0.90) | 0 [0.90,1.05) | 10 # count = 999
	$N_{cpu} = 1$	$C_{DE} = 0.8$	$f_{ave} = 0.0854687$	
	$t_{max} = 7000$	$N_{eval} = 350050$	$f_{max} = 1.0000000$	
	$\Delta t_{exc} = 1$	$T_{sys} = 42m13.971s$	$f_{dev} = \mathbf{0.1273616}$	
		X_{best}=[-1.71713 1.59583 1.82729 -0.76381 -0.76381]		
		$X_{ref.}$=[-1.71714 1.59571 1.82725 -0.76364 -0.76365]		

Problem 9 is run again with 7000 iterations as in [18]. The results are collected in Table II where it can bee seen the minimum is found with higher frequency. One failure occurs as well; i.e. the count of feasible solutions is 999 and not 1000. By tweaking the parameters and extending the number of iterations the results would improve; but for the sake of keeping the presentation short this has not been pursued.

VII. TEST CASES: UNCONSTRAINED TWO-OBJECTIVE

Multi objective optimisation problems may be more challenging than single objective problems at times, especially if the whole extent of the Pareto-optimal front is sought and the search space involves local optima. Now, two quantities are required in order to assess the accuracy and quality of results: (1) a measure of how well points are located close to the optimal front (E); and (2) a measure of the spread of points along the optimal front (L). In this paper, the accuracy is assessed by an error measure given by the root mean square error between the numerical and analytical solutions. The spread is calculated by summing the distance of points along the optimal front.

In this section, unconstrained problems with two objective functions $f_0(\boldsymbol{x})$ and $f_1(\boldsymbol{x})$ are considered. Part II [12] deals with constraints and more than two objectives. Here, six problems are studied. Five of them, named *ZDTi* were proposed in [24]. These problems include concave and convex curves in the (f_0, f_1) space and situations of multiple local minima. The interesting problem *FON* [21] in the form presented in [3] is also studied.

In these problems, C_{DE} is reduced to $C_{DE} = 0.1$ except for *FON* because it was found that this value produces better results. With $C_{DE} = 0.8$, the results are not bad, except that *ZDT4* requires more iterations because of the local fronts.

Problem ZDT1 has a convex Pareto-optimal front and is relatively easy to solve. It is defined by

$$\begin{aligned} f_0 &= x_0 \\ c_0 &= 1 + \frac{9}{N_x - 1} \sum_{i=1}^{N_x - 1} x_i \\ c_1 &= 1 - \sqrt{f_0 / c_0} \\ f_1 &= c_0\, c_1 \end{aligned} \tag{37}$$

where the limits of variables are

$$0 \le x_i \le 1 \quad (i \in [0, N_x - 1]) \tag{38}$$

$N_x = 30$ is selected and the optimal front is expressed by

$$f_1^{ana}(f_0) = 1 - \sqrt{f_0} \tag{39}$$

for which a line integral in $0 \le f_0 \le 1$ produces an arc length of $L_{ref} = \sqrt{5}/2 + \log(\sqrt{5} + 2)/4 \approx 1.478943$.

Problem ZDT2 has a concave Pareto-optimal front and is easy to solve as well. It is defined by

$$\begin{aligned} f_0 &= x_0 \\ c_0 &= 1 + \frac{9}{N_x - 1} \sum_{i=1}^{N_x - 1} x_i \\ c_1 &= 1 - (f_0 / c_0)^2 \\ f_1 &= c_0\, c_1 \end{aligned} \tag{40}$$

where the limits of variables are

$$0 \le x_i \le 1 \quad (i \in [0, N_x - 1]) \tag{41}$$

$N_x = 30$ is selected and the optimal front is expressed by

$$f_1^{ana}(f_0) = 1 - f_0^2 \tag{42}$$

for which a line integral in $0 \le f_0 \le 1$ also produces an arc length of $L_{ref} = \sqrt{5}/2 + \log(\sqrt{5} + 2)/4 \approx 1.478943$.

Problem ZDT3 has a number of disconnected Pareto-optimal fronts but is not difficult to solve. It is defined by

$$\begin{aligned} f_0 &= x_0 \\ c_0 &= 1 + \frac{9}{N_x - 1} \sum_{i=1}^{N_x - 1} x_i \\ c_1 &= 1 - \sqrt{f_0 / c_0} - \sin(10\,\pi\, f_0)\, f_0 / c_0 \\ f_1 &= c_0\, c_1 \end{aligned} \tag{43}$$

where the limits of variables are

$$0 \le x_i \le 1 \quad (i \in [0, N_x - 1]) \tag{44}$$

$N_x = 30$ is selected and the optimal front is expressed by

$$f_1^{ana}(f_0) = 1 - \sqrt{f_0} - \sin(10\,\pi\, f_0)\, f_0 \tag{45}$$

The arc length of the Pareto front in the (f_0, f_1) plane has to be computed in five steps (a to e) by performing line integrals along the following ranges

$$\begin{aligned} f_0^a &\in [0.000000100000000, 0.083001534925223] \\ f_0^b &\in [0.182228728029413, 0.257762363387862] \\ f_0^c &\in [0.409313674808657, 0.453882104088830] \\ f_0^d &\in [0.618396794416602, 0.652511703804663] \\ f_0^e &\in [0.823331798326633, 0.851832865436414] \end{aligned} \quad (46)$$

The above numbers were computed by means of solving a simple nonlinear problem for the local minima on the curve $f_1(f_0)$ using Newton's method. The first value is chosen equal to 10^{-7} though. The arc length is the sum of five integrals (numerically computed) resulting in $L_{ref} \approx 1.811$.

Problem ZDT4 has a convex Pareto-optimal front and is a difficult one. It presents challenges to the solver because there are many local optimal fronts. It is defined by

$$\begin{aligned} f_0 &= x_0 \\ c_0 &= 1 + 10\,(N_x - 1) + \sum_{i=1}^{N_x - 1} \left[x_i^2 - 10\,\cos\left(4\,\pi\,x_i\right)\right] \\ c_1 &= 1 - \sqrt{f_0 / c_0} \\ f_1 &= c_0\,c_1 \end{aligned} \quad (47)$$

where the limits of variables are

$$\begin{aligned} 0 &\leq x_0 \leq 1 \\ -5 &\leq x_i \leq 5 \quad (i \in [1, N_x - 1]) \end{aligned} \quad (48)$$

$N_x = 10$ is selected and the optimal front is defined by

$$f_1^{ana}(f_0) = 1 - \sqrt{f_0} \quad (49)$$

for which a line integral in $0 \leq f_0 \leq 1$ produces an arc length of $L_{ref} = \sqrt{5}/2 + \log\left(\sqrt{5} + 2\right)/4 \approx 1.478943$ as well.

Problem ZDT6 has a concave Pareto-optimal front and is of moderate difficulty. Actually goga solves this problem very well. The difficulty in this problem is related to the non-uniformity of the Pareto-optimal region. It is defined by

$$\begin{aligned} f_0 &= 1 - \exp\left(-4\,x_0\right)\,\left[\sin\left(6\,\pi\,x_0\right)\right]^6 \\ c_0 &= 1 + 9\,\left(\frac{1}{N_x - 1}\,\sum_{i=1}^{N_x - 1} x_i\right)^{0.25} \\ c_1 &= 1 - \left(f_0 / c_0\right)^2 \\ f_1 &= c_0\,c_1 \end{aligned} \quad (50)$$

where the limits of variables are

$$0 \leq x_i \leq 1 \quad (i \in [0, N_x - 1]) \quad (51)$$

$N_x = 10$ is selected and the optimal front is defined by

$$f_1^{ana}(f_0) = 1 - f_0^2 \quad (52)$$

In this problem, the curve representing the Pareto-optimal front in the (f_0, f_1) plane is limited to the left by a point $(f_0^\star, f_1^\star)$. This point can be found by using

$$x_0^\star = \frac{\mathrm{atan}(9\,\pi)}{6\,\pi} \quad (53)$$

in (50) to compute $f_0^\star = 0.280775$ which can then be inserted into (52) to calculate $f_1^\star = 0.921165$. The arc length can now be computed by a line integral in $f_0^\star \leq f_0 \leq 1$ resulting in $L_{ref} \approx 1.184$.

Problem FON has a mostly concave Pareto-optimal front, except near $f_0 = 0$ or $f_0 = 1$ and has moderate difficulty. It is defined by

$$\begin{aligned} f_0 &= 1 - \exp\left[-\sum_{i=0}^{N_x - 1}\left(x_i - \frac{1}{N_x}\right)^2\right] \\ f_1 &= 1 - \exp\left[-\sum_{i=0}^{N_x - 1}\left(x_i + \frac{1}{N_x}\right)^2\right] \end{aligned} \quad (54)$$

where the limits of variables are

$$0 \leq x_i \leq 1 \quad (i \in [0, N_x - 1]) \quad (55)$$

$N_x = 10$ is selected and the optimal front is defined by

$$f_1^{ana}(f_0) = 1 - \exp\left\{-\left[2 - \sqrt{-\log\left(1 - f_0\right)}\right]^2\right\} \quad (56)$$

for which a line integral in $0 \leq f_0 \leq 0.98$ produces an arc length of $L_{ref} \approx 1.458$.

The results are discussed next. Again, 1000 samples are run in order to compute some statistics on the error E and spread L. Each sample is executed with $t_f = 500$ iterations and the number of solutions is $N_{sol} = 10\,N_x$, as before. The number of groups N_{cpu} is equal to 6 for the problems with $N_{sol} = 30$ and equal to 2 for problems with $N_{sol} = 10$. The exchange time is $T_{exc} = t_f/10$.

For each sample, the following root mean square error E is computed

$$E = \sqrt{\frac{1}{n}\,\sum_{i=0}^{n-1}\left[f_1^i - f_1^{ana}(f_0^i)\right]^2} \quad (57)$$

where f_0^i is the numerical solution corresponding to $\boldsymbol{x}_i$ and n is the number of solutions (points) that are feasible and non-dominated at the best front ($\phi_i = 0$).

The spread is computed by adding the Euclidean distance between successive points along the optimal front; from left to right in a (f_0, f_1) graph. This measure is then normalised by the reference arc length L_{ref} determined analytically. The definition of L is thus

$$L = \frac{1}{L_{ref}}\,\sum_{\substack{1 \leq j < n \\ 0 \leq i < n-1}} \sqrt{\left(f_0^j - f_0^i\right)^2 + \left(f_1^j - f_1^i\right)^2} \quad (58)$$

where n indicates the number of feasible solutions on the optimal front with $\phi_i = 0$. Therefore, contrary to the error measure, the spread is better if closer to 1. Note that the above measure does not account for how equally spaced points are.

Plots of Pareto-optimal fronts for the six problems are presented in Figs. 4-9 corresponding to one (any) sample.

Table III collects the results from all 1000 samples where it can be observed that the performance is very good with the error achieving the machine precision 0 for four tests. A good spread is also observed in all problems. Finally, the repeatability behaviour is also excellent with a very small standard deviation both in terms of accuracy and spread.

TABLE III
UNCONSTRAINED TWO OBJECTIVE PROBLEMS (1000 SAMPLES).

P	settings	settings/info	error	spread
ZDT1	$N_{sol} = 300$	$count = 1000$	$E_{min} = 0$	$L_{min} = 0.999925$
	$N_{cpu} = 6$	$C_{DE} = 0.1$	$E_{ave} = 0$	$L_{ave} = 0.999988$
	$t_{max} = 500$	$N_{eval} = 150300$	$E_{max} = 0$	$L_{max} = 0.999995$
	$\Delta t_{exc} = 50$	$T_{sys} = 15m1.86s$	$E_{dev} = \mathbf{0}$	$L_{dev} = \mathbf{5.7014 \cdot 10^{-6}}$
ZDT2	$N_{sol} = 300$	$count = 1000$	$E_{min} = 0$	$L_{min} = 0.999990$
	$N_{cpu} = 6$	$C_{DE} = 0.1$	$E_{ave} = 0$	$L_{ave} = 0.999995$
	$t_{max} = 500$	$N_{eval} = 150300$	$E_{max} = 0$	$L_{max} = 0.999998$
	$\Delta t_{exc} = 50$	$T_{sys} = 15m1.091s$	$E_{dev} = \mathbf{0}$	$L_{dev} = \mathbf{9.8781 \cdot 10^{-7}}$
ZDT3	$N_{sol} = 300$	$count = 1000$	$E_{min} = 0$	$L_{min} = 0.907713$
	$N_{cpu} = 6$	$C_{DE} = 0.1$	$E_{ave} = 0$	$L_{ave} = 0.971073$
	$t_{max} = 500$	$N_{eval} = 150300$	$E_{max} = 0$	$L_{max} = 0.994236$
	$\Delta t_{exc} = 50$	$T_{sys} = 14m39.82s$	$E_{dev} = \mathbf{0}$	$L_{dev} = \mathbf{1.2775 \cdot 10^{-2}}$
ZDT4	$N_{sol} = 100$	$count = 1000$	$E_{min} = 4.631 \cdot 10^{-11}$	$L_{min} = 0.97533341$
	$N_{cpu} = 2$	$C_{DE} = 0.1$	$E_{ave} = 1.826 \cdot 10^{-9}$	$L_{ave} = 0.99992874$
	$t_{max} = 500$	$N_{eval} = 50100$	$E_{max} = 3.868 \cdot 10^{-8}$	$L_{max} = 0.99998386$
	$\Delta t_{exc} = 50$	$T_{sys} = 10m18.506s$	$E_{dev} = \mathbf{2.616 \cdot 10^{-9}}$	$L_{dev} = \mathbf{9.1248 \cdot 10^{-4}}$
ZDT6	$N_{sol} = 100$	$count = 1000$	$E_{min} = 0$	$L_{min} = 0.9987929209$
	$N_{cpu} = 2$	$C_{DE} = 0.1$	$E_{ave} = 0$	$L_{ave} = 0.9999792791$
	$t_{max} = 500$	$N_{eval} = 50100$	$E_{max} = 0$	$L_{max} = 1.0000227029$
	$\Delta t_{exc} = 50$	$T_{sys} = 9m49.271s$	$E_{dev} = \mathbf{0}$	$L_{dev} = \mathbf{9.111 \cdot 10^{-5}}$
FON	$N_{sol} = 100$	$count = 1000$	$E_{min} = 0.01184852$	$L_{min} = 1.00175484$
	$N_{cpu} = 2$	$C_{DE} = 0.8$	$E_{ave} = 0.01672835$	$L_{ave} = 1.03712483$
	$t_{max} = 500$	$N_{eval} = 50100$	$E_{max} = 0.02693546$	$L_{max} = 1.24856398$
	$\Delta t_{exc} = 50$	$T_{sys} = 11m5.085s$	$E_{dev} = \mathbf{1.8358 \cdot 10^{-3}}$	$L_{dev} = \mathbf{1.7911 \cdot 10^{-2}}$

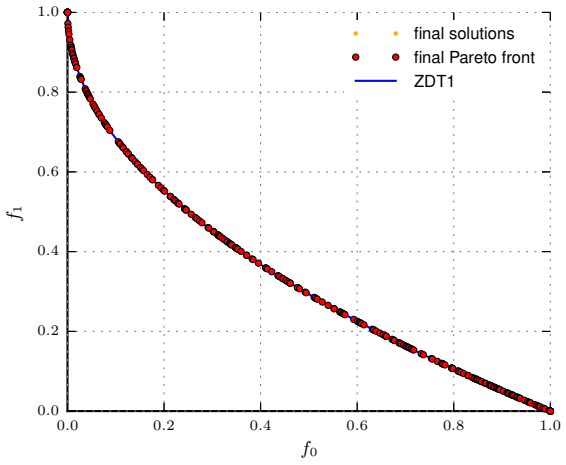

Fig. 4. ZDT1: Results.

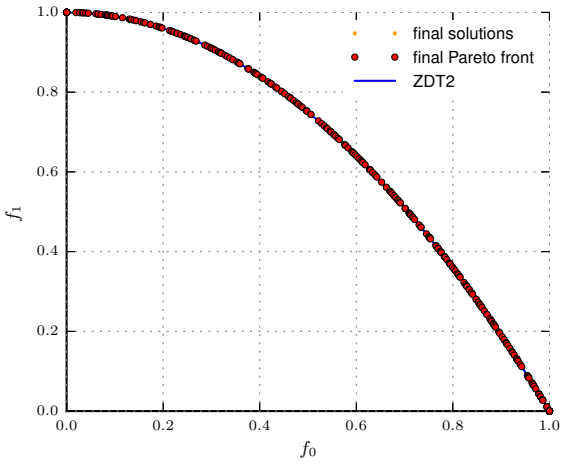

Fig. 5. ZDT2: Results.

VIII. CONCLUSIONS

This paper presented an evolutionary algorithm capable of solving a wide range of optimisation problems, including some with many constraints and more than one objective. The repeatability of the code is demonstrated by observing a small standard deviation from several runs with the same problem but different initial trial solutions. This characteristic is essential to obtain a level of reliability in the code. Nonetheless, one problem posed some challenges; namely, problem 9 with a single objective. This problem has an exponential function with a multiplicative term that makes the problem difficult. With more iterations (evolution time), this problem can be solved though.

The solution method is a combination of algorithms based on other research works with modifications. These include a crowding approach to niching using tournaments, differential evolution (DE) operations and Pareto non-dominance comparisons. One DE coefficient (vector length F) is randomly generated in order to better search solutions with variable spacings. The other one (probability C_{DE}) is suggested as 0.8;

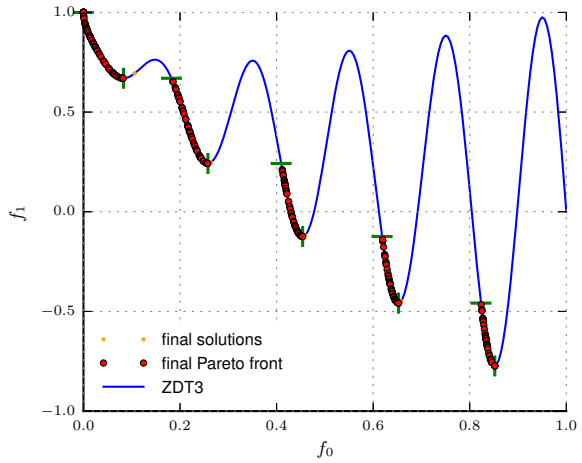

Fig. 6. ZDT3: Results.

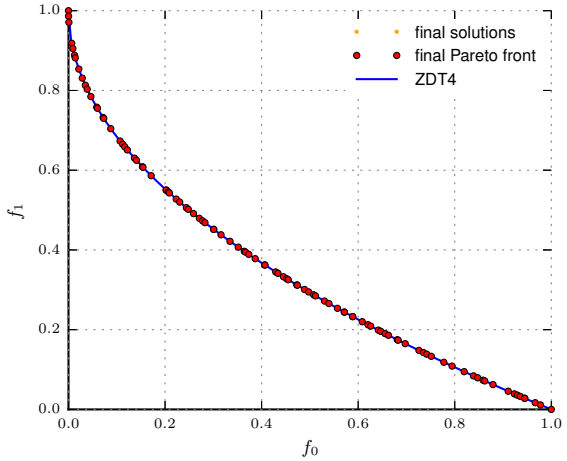

Fig. 7. ZDT4: Results.

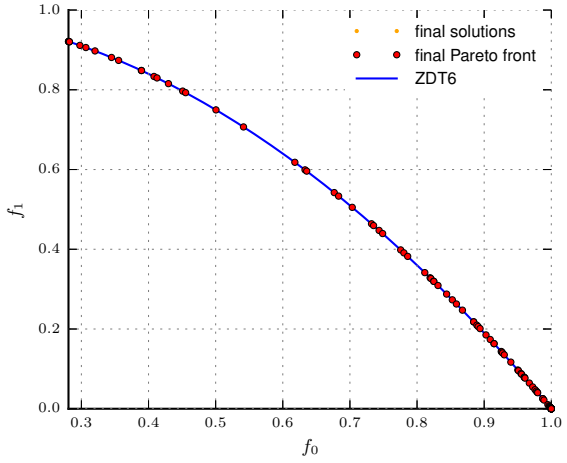

Fig. 8. ZDT6: Results.

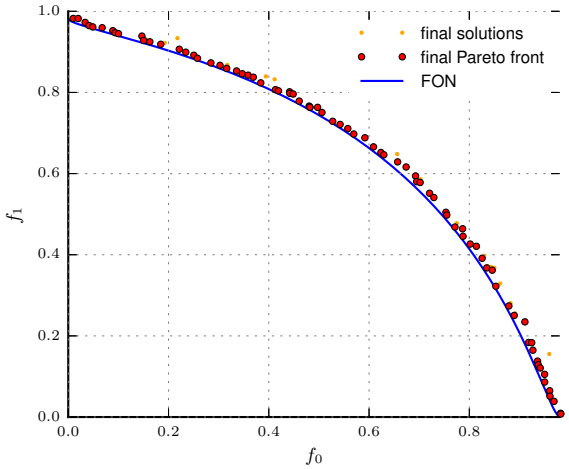

Fig. 9. FON: Results.

but can be decreased in some problems for better performance.

The concept of out-of-range (OOR) values is introduced to handle not only constraints but also failures. A Pareto comparison of constraints via OOR values is proposed. In addition, a constraints handling based on a decision tree prioritising OORs is introduced. These last two strategies pull infeasible solutions into feasible regions and worked very well.

The algorithm is designed to be used in multiple cores machines and be run in parallel. This is accomplished by splitting the space of trial solutions into smaller groups, each one running in parallel until a time for exchange of solutions is reached. Two strategies for exchanging solutions are combined: by tournament and randomly. These two strategies help with widening the search space in addition of making computations faster. Part II presents further evidence of the good characteristics of the proposed code including the ideal speedup properties.

Overall, the combination of all proposed strategies leads to a good performance (accuracy, efficiency, repeatability).

ACKNOWLEDGMENT

The support from the Australian Research Council under grant DE120100163 is gratefully acknowledged.

REFERENCES

[1] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Addison-Wesley, 1989.
[2] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.
[3] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley, 2001.
[4] K. C. Tan, E. F. Khor, and T. H. Lee, *Multiobjective Evolutionary Algorithms and Applications*. Springer-Verlag London, 2005.
[5] C. C. Coello, G. B. Lamont, and D. A. van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, 2007.
[6] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," in *Proc. of the 2nd Int. Conf. on Genetic Algorithms and Their Application*. L. Erlbaum Associates Inc., 1987, pp. 41–49.
[7] S. W. Mahfoud, "Niching methods for genetic algorithms," Ph.D. dissertation, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 1995.

[8] C. H. Chen, T. K. Liu, and J. H. Chou, "A Novel Crowding Genetic Algorithm and Its Applications to Manufacturing Robots," *IEEE Trans. on Industrial Informatics*, vol. 10, no. 3, pp. 1705–1716, 2014.
[9] S. M. Elsayed, R. a. Sarker, and D. L. Essam, "A new genetic algorithm for solving optimization problems," *Engineering Applications of Artificial Intelligence*, vol. 27, pp. 57–69, 2014.
[10] O. J. Mengshoel and D. E. Goldberg, "The crowding approach to niching in genetic algorithms," *Evolutionary Computation*, vol. 16, no. 3, pp. 315–354, 2008.
[11] O. J. Mengshoel, S. F. Galn, and A. de Dios, "Adaptive generalized crowding for genetic algorithms," *Information Sciences*, vol. 258, pp. 140–159, 2014.
[12] D. M. Pedroso, "Parallel evolutionary algorithm for constrained single and multi objective optimisation - Part II: three-objective test cases and applications," *IEEE Trans. Evol. Comput.*, vol. -, no. -, pp. –, 2016, submitted.
[13] Z. Michalewicz and N. Attia, "Evolutionary optimization of constrained problems," in *Proc. of the 3rd Annual Conf. on Evolutionary Programming*, A. Sebald and L. Fogel, Eds. World Scientific Publishing, River Edge, NJ, 1994, pp. 98–108.
[14] Z. Michalewicz, "Genetic algorithms, numerical optimization and constraints," in *Proc. of the 6th Int. Conf. on Genetic Algorithms*, 1995, pp. 151–158.
[15] Z. Michalewicz and M. Schoenauer, "Evolutionary algorithms for constrained parameter optimization problems," *Evolutionary Computation*, vol. 4, no. 1, pp. 1–32, 1996.
[16] D. Vieira, R. Adriano, J. Vasconcelos, and L. Krahenbuhl, "Treating Constraints as Objectives in Multiobjective Optimization Problems Using Niched Pareto Genetic Algorithm," *IEEE Trans. on Magnetics*, vol. 40, no. 2, pp. 1188–1191, 2004.
[17] Q. Long, "A constraint handling technique for constrained multi-objective genetic algorithm," *Swarm and Evolutionary Computation*, vol. 15, pp. 66–79, 2014.
[18] K. Deb, "An efficient constraint handling method for genetic algorithms," *Comput. Methods in Appl. Mech. and Engrg.*, vol. 186, no. 2-4, pp. 311–338, 2000.
[19] C. A. C. Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art," *Comput. Methods in Appl. Mech. and Engrg.*, vol. 191, no. 11–12, pp. 1245–1287, 2002.
[20] M. Tanaka, H. Watanabe, Y. Furukawa, and T. Tanino, "Ga-based decision support system for multicriteria optimization," in *IEEE Int. Conf. on Systems, Man and Cybernetics*, vol. 2, Oct 1995, pp. 1556–1561 vol.2.
[21] C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimization," *Evolutionary Computation*, vol. 3, no. 1, pp. 1–16, 1995.
[22] ——, "Multiobjective optimization and multiple constraint handling with evolutionary algorithms. I. a unified formulation," *IEEE Trans. on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 28, no. 1, pp. 26–37, 1998.
[23] ——, "Multiobjective optimization and multiple constraint handling with evolutionary algorithms. II. Application example," *IEEE Trans. on Systems, Man, and Cybernetics, Part A: Systems and Humans*, vol. 28, no. 1, pp. 1–13, 1998.
[24] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evol. Comput.*, vol. 8, no. 2, pp. 173–195, 2000.
[25] K. Tan, T. Lee, and E. Khor, "Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization," *IEEE Trans. Evol. Comput.*, vol. 5, no. 6, pp. 565–588, 2001.
[26] K. Deb, "Nonlinear goal programming using multi-objective genetic algorithms," *Journal of the Operational Research Society*, vol. 52, no. 3, pp. 291–302, 2001.
[27] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr 2002.
[28] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable test problems for evolutionary multiobjective optimization," in *Evolutionary Multi-objective Optimization*, A. Abraham, L. Jain, and R. Goldberg, Eds. Springer, 2005, pp. 105–145.
[29] S. Kukkonen and K. Deb, "A Fast and Effective Method for Pruning of Non-dominated Solutions in Many-Objective Problems," *Parallel Problem Solving from Nature*, vol. 4193, pp. 553–562, 2006.
[30] K. Deb and S. Tiwari, "Omni-optimizer: A generic evolutionary algorithm for single and multi-objective optimization," *European Journal of Operational Research*, vol. 185, no. 3, pp. 1062–1087, 2008.
[31] S. Tiwari, G. M. Fadel, P. Koch, and K. Deb, "AMGA : An Archive-based Micro Genetic Algorithm for," *Gecco*, pp. 729–736, 2008.
[32] K. Deb, K. Miettinen, and S. Chaudhuri, "Toward an estimation of nadir objective vector using a hybrid of evolutionary and local search approaches," *IEEE Trans. Evol. Comput.*, vol. 14, no. 6, pp. 821–841, 2010.
[33] S. Tiwari, G. Fadel, and K. Deb, "AMGA2: improving the performance of the archive-based micro-genetic algorithm for multi-objective optimization," *Engineering Optimization*, vol. 43, no. 4, pp. 377–401, 2011.
[34] K. Deb and R. B. Agrawal, "Simulated Binary Crossover for Continuous Search Space," *Complex Systems*, vol. 9, pp. 115–148, 1995.
[35] K. Deb and A. Kumar, "Real-coded Genetic Algorithms with Simulated Binary Crossover: Studies on Multimodal and Multiobjective Problems," *Complex Systems*, vol. 9, pp. 431–454, 1995.
[36] F. Herrera, M. Lozano, and J. Verdegay, "Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis," *Artificial Intelligence Review*, vol. 12, no. 4, pp. 265–319, 1998.
[37] D. M. Pedroso and D. J. Williams, "Automatic calibration of soil–water characteristic curves using genetic algorithms," *Computers and Geotechnics*, vol. 38, no. 3, pp. 330–340, 2011.
[38] R. Storn and K. V. Price, "Differential evolution–a simple and efficient adaptive scheme for global optimization over continuous spaces," *Technical Report TR-95-012, ICSI*, 1995.
[39] K. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*. Springer-Verlag, 2005.
[40] S. Kukkonen and J. Lampinen, "GDE3: the third evolution step of generalized differential evolution," in *The 2005 IEEE Congress on Evolutionary Computation*, vol. 1, 2005, pp. 443–450.
[41] ——, "An Empirical Study of Control Parameters for The Third Version of Generalized Differential Evolution (GDE3)," in *IEEE Int. Conf. on Evolutionary Computation*, 2006, pp. 2002–2009.
[42] The Go Authors, "The go programming language," *https://golang.org*, 2016. [Online]. Available: https://golang.org
[43] W. Hock and K. Schittkowski, *Test Examples for Nonlinear Programming Codes*. Springer, 1981.
[44] K. M. Ragsdell and D. T. Phillips, "Optimal design of a class of welded structures using geometric programming," *Journal of Engineering for Industry*, vol. 98, pp. 1021–1025, 1976.
[45] A. Ravindran, K. M. Ragsdell, and G. V. Reklaitis, *Engineering Optimization: Methods and Applications, Second Edition*. Wiley, 2007.

Dorival Pedroso received a BSc degree in Civil Engineering from the Catholic University of Goias, Brazil and a MSc and PhD in Geomechanics from the University of Brasilia, Brazil, partially by the Nagoya Institute of Technology, Japan. He has developed material models for mechanical behaviour and methods for solving differential equations. His current research interests include nonlinear optimisation, artificial intelligence, and reliability in engineering. He is the leader of the Computational Engineering and Mechanics research group (www.cpmech.com) at The University of Queensland, Australia.