# Parallel evolutionary algorithm in Go language for single and multi objective optimisation problems with constraints

Dorival M. Pedroso[a,*]

[a]*School of Civil Engineering, The University of Queensland, St Lucia QLD 4072, Australia*

**Abstract**

This paper presents an evolutionary algorithm employing differential evolution for solving nonlinear optimisation problems with (or without) multiple constraints and objectives. Some new decision strategies to compare trial solutions in order to produce a robust code with respect to its ability to generate the same answer every time it is run is presented. In addition, the combination of good existent techniques is carefully developed. The third contribution is the design of a strategy to implement parallel computations in a way that diversity is preserved. A final goal is the handling of a range of problems involving real numbers only or real numbers and integers. The resulting code is named Goga and is written in Go (golang) as free/open source code. Numerical examples are presented illustrating the capabilities and performance of the algorithm. An ideal speedup is observed during the parallel computations. Two applications are studied: the topology optimisation of trusses and the environmental economical dispatch problem in power generation.

*Keywords:*
golang, differential evolution, constraints handling, many objectives, truss optimisation, economic dispatch, parallel computation

## 1. Introduction

Evolutionary algorithms (EAs), including genetic algorithms (GAs), are optimisation techniques with a good dissemination in engineering and many other areas [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.

Many important problems in Engineering involve multiple constraints; for example 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 (along the transmission lines)—the power balance.

Methods to handle constraints in evolutionary algorithms range from the use of penalty factors to classification strategies when comparing trial solutions. The penalty method is often the worst one because the addition of large numbers to the objective function causes numerical problems. This paper adopts the classification strategy for constraints and presents an algorithm to treat them in single and multi objective optimisation problems. The concept of Pareto dominance is then applied to both the objective values and measures of how close to satisfying constraints a trial solution is. Interesting studies on constraints handling methods are available in [12–21].

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. Algorithms have then been developed along the years to solve multi objective problems as well [22–36].

Real-coded genetic algorithms have been a challenge in the early days; but good solutions were introduced and studied in [37, 38, 2, 39]. A simple attempt that split a float point number

*Corresponding author
*Email address:* d.pedroso@uq.edu.au (Dorival M. Pedroso)

into smaller chunks was also successfully proposed and implemented in [40]. Nonetheless, the performance was not necessarily always the best. The differential evolution (DE) [41, 42] 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 [35]. 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 [43, 44].

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. Therefore, new algorithms have to consider parallel or concurrency techniques in order to make better use of the computer hardware. 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. Nonetheless, care must be taken in order to prevent the loss of diversity when exchanging data between different groups. Thus, a strategy to implement concurrency is also investigated and proposed in this paper.

Go (golang) is a recent new programming language created by Google engineers in 2007, including Robert Griesemer, Rob Pike and Ken Thompson [45]. The language was later made public as open source in 2009. Go has since grown exponentially attracting a large number of co-developers and users. The main goal leading to the introduction of yet a new language was the combination of efficiency (like C/C++) with ease of development (like Python). There are other several innovations and advantages in Go when compared with mainstream languages such as C/C++/C#/Java/Python/Ruby/Lua.

One particular innovation in Go is the concept of concurrency that can lead to easy-to-write parallel algorithms. Another convenient characteristic is the simple development cycle that includes very fast compilation times and efficient modularity by using packages and sub-packages. Go is a garbage-collected language, compiled in Go language, and finally, has at times prioritised practicality over strict conciseness that would be defined by long theoretical justifications. Nonetheless the language definition is very clear and well thought of—note that the creators have indeed proven experience on computer language design. This last pragmatic approach makes Go as pleasing to use as possible while ensuring that the code is well organised, modular and efficient.

The algorithms in this paper are implemented in Go; however they are presented in a general format that can help the implementation in any other language. The resulting code 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 for all tests, examples and graphs are posted in the same web address.

In summary, this paper presents an evolutionary algorithm for solving optimisation problems with multiple objectives and multiple constraints. The best ideas collected from the aforementioned literature are taken into consideration and new algorithms and classification strategies are developed, including: (1) Pareto handling of constraints without penalty weights; (2) crowding/niching for diversity maintenance to be used in parallel computations; (3) use of both a neighbour distance and a crowding distance based on the proposed constraints handling algorithm; (4) random differential evolution operations; (5) two strategies for exchange of solutions during parallel executions; and (6) handling of hybrid representations using integers and real numbers.

## 2. Problem definition

The optimisation problem (solved by Goga) considering multiple objectives, multiple constraints, and mixed types is defined here in a *general format* by means of

$$\begin{array}{lll} \underset{\boldsymbol{x},\boldsymbol{\xi}}{\text{minimise}} & \{f_0(\boldsymbol{x},\boldsymbol{\xi}), f_1(\boldsymbol{x},\boldsymbol{\xi}), \ldots\} & (N_f \text{ objectives}) \\ \text{subject to} & u_i(\boldsymbol{x},\boldsymbol{\xi}) = 0 & (N_u \text{ constraints}) \\ & x_i^L \le x_i \le x_i^U \quad (x_i \in \mathbb{R}) & (N_x \text{ box-constr.}) \\ & \xi_i^L \le \xi_i \le \xi_i^U \quad (\xi_i \in \mathbb{Z}) & (N_\xi \text{ box-constr.}) \end{array} \tag{1}$$

where $f_i(\boldsymbol{x},\boldsymbol{\xi})$ are the $N_f$ objective functions, $\boldsymbol{x} = \{x_0, x_1, \ldots, x_{Nx-1}\}$ are the design variables (unknowns) of real type, and $\boldsymbol{\xi} = \{\xi_0, \xi_1, \ldots, \xi_{N\xi-1}\}$ are the design variables of integer type. Therein, $u_i(\boldsymbol{x},\boldsymbol{\xi})$ are auxiliary functions called here as *out-of-range* (OOR) that are introduced to handle additional constraints with respect to the essential *box constraints* for $\boldsymbol{x}$ and $\boldsymbol{\xi}$. In this paper, when counting constraints, the box constraints are not included; hence, if only box constraints are present, the problem is called *unconstrained*. Hereafter, the acronym OVA is also employed to indicate an objective-value.

The general form in Eq. (1) includes the more *specific problem* expressed by

$$\begin{array}{lll} \underset{\boldsymbol{x}}{\text{minimise}} & \{f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots\} & (N_f \text{ objectives}) \\ \text{subject to} & g_i(\boldsymbol{x}) \ge 0 & (N_g \text{ constraints}) \\ & h_i(\boldsymbol{x}) = 0 & (N_h \text{ constraints}) \\ & x_i^L \le x_i \le x_i^U & (N_x \text{ box-constr.}) \end{array} \tag{2}$$

where $g_i(\boldsymbol{x})$ and $h_i(\boldsymbol{x})$ are inequality and equality constraint functions, respectively. These two types of constraints can be easily converted into the OOR form using the $u_i$ functions in Eq. (1); the number of OOR functions will then be

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

The out-of-range functions $u_i$ must return positive values that become smaller as the variables $\boldsymbol{x}$ and $\boldsymbol{\xi}$ become closer to satisfying the corresponding constraint. If the solution is feasible, the OOR function is simply zero. The pursuit of zero OORs has priority over minimising OVAs as discussed in the next section. Note that the strategy involving the out-of-range function is one step to effectively handle multiple constraints and, in the

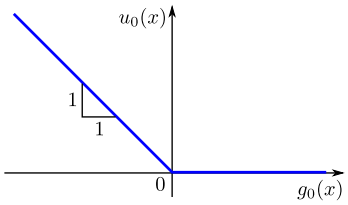



Figure 1: Out-of-range (infeasible) function $u_0$ for inequality constraint $g_0$.

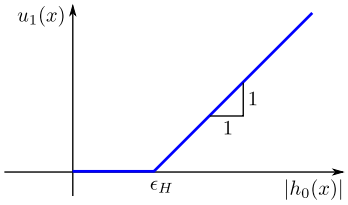



Figure 2: Out-of-range (infeasible) function $u_1$ for equality constraint $h_0$.

proposed evolutionary algorithm, this method will pull infeasible solutions towards the feasibility region.

To exemplify, in a problem involving two constraints $g_0$ and $h_0$, the ramp function defined by

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

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

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

can be used for the second one. Therein, $\epsilon_H$ is a small number, but not very small (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 Eq. (4) is given in Fig. 1 and an illustration of Eq. (5) is given in Fig. 2.

The solution of the optimisation problem defined above is obtained in a stochastic manner starting with the definition of a set of trial solutions indicated by

$$S = \{(\mathbf{x}, \boldsymbol{\xi})_0, (\mathbf{x}, \boldsymbol{\xi})_1, \dots (\mathbf{x}, \boldsymbol{\xi})_{Nsol-1}\} \tag{6}$$

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

The minimisation problem considers the concept of Pareto domination [24]. In this concept, a vector $\mathbf{v}_A$ dominates another one $\mathbf{v}_B$ if and only if the following conditions are satisfied at the same time

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

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

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

In this paper, the command `ParetoComparison` means to apply Eq. (7) to any two vectors related to the trial solutions $A$ and $B$ and to find whether solution $A$ dominates solution $B$ ($A_{dom}$ = true), solution $B$ dominates solution $A$ ($B_{dom}$ = 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

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

## 3. Constraints handling

The method employed to find solutions satisfying all constraints is inspired by the one introduced in [15]; but avoids adding weights to the objective function (penalty method). The proposed algorithm is designed in such a way to deal with multi objective problems at the same time as single objective ones. This is accomplished by means of a Pareto comparison of OORs in addition to a Pareto comparison of OVAs. There are other algorithms that avoid penalties as well [29], including the use of Pareto comparisons and a ranking procedure [46, 47]. Nonetheless, the algorithm presented here is different from those in the way it applies the non-dominance comparison of OORs without ranking and by additionally considering the number of constraint violations. Other interesting approaches are available in [48–50].

The motivation for the method is based on the way some constraints are dealt with in engineering. Specifically, a trial solution not satisfying all the constraints, thus termed *infeasible*, is completely ignored at design stages—for example, an infeasible circuit would never be even considered. The problem that arises in this approach is when all solutions are infeasible and therefore comparisons cannot be made. One way around this problem is to classify how infeasible a solution is, only in this exception, and then perform the comparison between trial solutions. This is facilitated by the OOR functions discussed earlier.

The constraints handling algorithm is presented in Algorithm 1. 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 $\mathbf{u}_i$. 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 $\mathbf{f}_i$. 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 the other.

## 4. 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,

**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
$N^A_{viol}, N^B_{viol} \leftarrow 0, 0$
**for** *OOR indices* $0 \le i < N_u$ **do**
  **if** $u^A_i > 0$ **then** $N^A_{viol}$ += 1
  **if** $u^B_i > 0$ **then** $N^B_{viol}$ += 1
**if** $N^A_{viol} > 0$ **then**
  **if** $N^B_{viol} > 0$ **then**
    **if** $N^A_{viol} < N^B_{viol}$ **then**
      $A_{dom} \leftarrow$ true
      **return**
    **if** $N^B_{viol} < N^A_{viol}$ **then**
      $B_{dom} \leftarrow$ true
      **return**
    $A_{dom}, B_{dom} \leftarrow$ `ParetoComparison`($\dot{\boldsymbol{u}}_A, \dot{\boldsymbol{u}}_B$)
    **if** not $A_{dom}$ and not $B_{dom}$ **then**
      $A_{dom}, B_{dom} \leftarrow$ `ParetoComparison`($\dot{\boldsymbol{f}}_A, \dot{\boldsymbol{f}}_B$)
    **return**
  $B_{dom} \leftarrow$ true
  **return**
**if** $N^B_{viol} > 0$ **then**
  $A_{dom} \leftarrow$ true
  **return**
$A_{dom}, B_{dom} \leftarrow$ `ParetoComparison`($\dot{\boldsymbol{f}}_A, \dot{\boldsymbol{f}}_B$)

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`($\Omega$) function with $\Omega$ being a set of solutions:

- $x^{min}_i$ and $x^{max}_i$ minimum and maximum solution values of real type in $\Omega$ corresponding to dimension $i$
- $y^{min}_i$ and $y^{max}_i$ minimum and maximum solution values of integer type in $\Omega$ corresponding to dimension $i$
- $f^{min}_i$ and $f^{max}_i$ minimum and maximum objective values in $\Omega$ corresponding to objective function $i$
- $\eta_i$ *neighbour distance*: smallest distance to any neighbour around solution $i$
- $w_i$ *crowding distance* corresponding to solution $i$
- $\phi_i$ 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

$$d_{ij} = \frac{1}{N_x} \sum_k^{N_x} \frac{x^i_k - x^j_k}{x^{max}_k - x^{min}_k + \epsilon} + \frac{1}{N_y} \sum_k^{N_y} \frac{y^i_k - y^j_k}{y^{max}_k - y^{min}_k + \epsilon} \quad (9)$$

**Algorithm 2.** `ParetoFronts`($\Omega$) computes Pareto fronts for a set of trial solutions $\Omega$.

**Input:** set of trial solutions $\Omega$
**Output:** set of solutions in Pareto fronts $F_{i,j}$, size of fronts $z_i$, and ranks $\phi_i$
zero all $N^i_{wins}$, $N^i_{loss}$, $\phi_i$ and $z_i$ variables
`// compute wins/losses data`
**for** *each solution A in* $\Omega$ **do**
  **for** *each B in* $\Omega$ *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* $\Omega$ **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

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 $\eta_i$ which is called the *neighbour distance* of $i$.

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

The rank $\phi_i$ of the Pareto front of solution $i$ is computed after finding all fronts using an algorithm similar to the one presented in [29]. 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 in this paper. In this way, the infeasible solutions will have a smaller chance to appear in the best fronts (those with small $\phi$).

The required steps are collected in Algorithm 2 where $W_i$ is a subset of solutions such that solution $i$ wins the

**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 $j$")
    $\delta \leftarrow f_j^{max} - f_j^{min} + \epsilon$
    $w_{K_0}, w_{K_m} \leftarrow$ INF, INF
    **for** *indices i in K, except first and last* **do**
      $w_i$ += $\frac{f_j^i - f_j^{i-1}}{\delta} \times \frac{f_j^{i+1} - f_j^i}{\delta}$

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 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 [35]. The only difference is that extreme solutions are assigned a very large ($10^{30}$) crowding distance. Note that this distance is only computed (required) for solutions that belong to the same Pareto front (with the same $\phi$). For each front, if the front has more than two solutions, the $w_i$ of a solution $i$ (not at the extremities of the front) is computed by

$$w_i = \sum_{j}^{N_f} \frac{f_j^i - f_j^{i-1}}{\delta} \times \frac{f_j^{i+1} - f_j^i}{\delta} \tag{10}$$

where $\delta = f_j^{max} - f_j^{min} + \epsilon$ 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. The steps are summarised in Algorithm 3.

## 5. Evolutionary algorithm

The evolutionary algorithm with a differential evolution (DE) operator is presented in this section (Algorithm 4). A set (population) of trial solutions is firstly generated and later organised into smaller and distinct groups. Within each group, evolution is simulated in parallel where trial solutions are combined in order to randomly walk 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 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

**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**
    **begin**
      **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`
  $t$ += $\Delta t_{exc}$
  $t_{exc}$ += $\Delta t_{exc}$

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. 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 [45], 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 [45]. The way memory is shared in Go is by communication which can be done by using *channels*. The concept of channels makes it 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) // note function call here
}
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 [45].

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 $\Delta t_{exc}$ 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 neighbour 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 ($\Delta t_{exc}$). 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). This approach exploits the diversity preserving characteristic of the crowding approach to niching that is also applied during the evolution process.

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. It is expected that this step might reduce diversity to some small extent; however it has some beneficial effect.

### *5.1. 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 one 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 may look like:

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

The DE operator, presented shortly, 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 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 $\dot{\boldsymbol{\xi}}$ 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 $\Gamma$. After that, the `Metrics` function is called with the union of the old ($G$) and new populations ($\Gamma$). In this way, the information required for tournaments such as the Pareto front, neighbour 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 neighbour distance $d_{ij}$ (Section 4) 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 [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.

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

**Input:** $G$ and backup set of solutions $\Gamma$ 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)
    ξ_A ← IntPart(G[P_{k,0}])
    ξ_B ← IntPart(G[P_{k,1}])
    ξ_a, ξ_b ← Crossover(ξ_A, ξ_B)
    ξ_a ← Mutation(ξ_a)
    ξ_b ← Mutation(ξ_b)
    store {x_a, x_b, ξ_a, ξ_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)
```

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 $B$; for instance, $A$ 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 neighbour distance $\eta$ 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 solutions are in different fronts—in this case, first, the one with the smallest Pareto front rank $\phi$ wins; second, the one with the largest neighbour distance $\eta$ 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.

**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)
```

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

**Input:** trial solutions $A, B, a, b$
**Output:** $\Omega$: replace $A$ or $B$, or both, in the right position in the set $\Omega$

```
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`$(\dot{\boldsymbol{x}}, \dot{\boldsymbol{x}}_0, \dot{\boldsymbol{x}}_1, \dot{\boldsymbol{x}}_2)$ generates a new solution using the differential evolution operator.

**Input:** trial solutions $\dot{\boldsymbol{x}}, \dot{\boldsymbol{x}}_0, \dot{\boldsymbol{x}}_1, \dot{\boldsymbol{x}}_2$
**Output:** new solution $\dot{\boldsymbol{x}}^{new}$
$F \leftarrow$ `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**
    $x_i^{\text{new}} \leftarrow x_i$

### 5.2. Differential evolution

The differential evolution [41, 42] method is selected to combine real numbers. Other methods are available such as the SBX [37]; however some experiments with the proposed code demonstrated that DE is more efficient. This has been partially observed by others as well [35].

As mentioned earlier, three auxiliary solutions are needed to compute 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 [42] and is accomplished by means of

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

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 $\dot{\boldsymbol{x}}$ being modified or not. A default value of $C_{DE} = 0.8$ is proposed here after studying a number of test cases.

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.

## 6. 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 optimisation problems with many constraints are studied. The next sections presents studies of unconstrained and constrained multi-objective problems, followed by some applications.

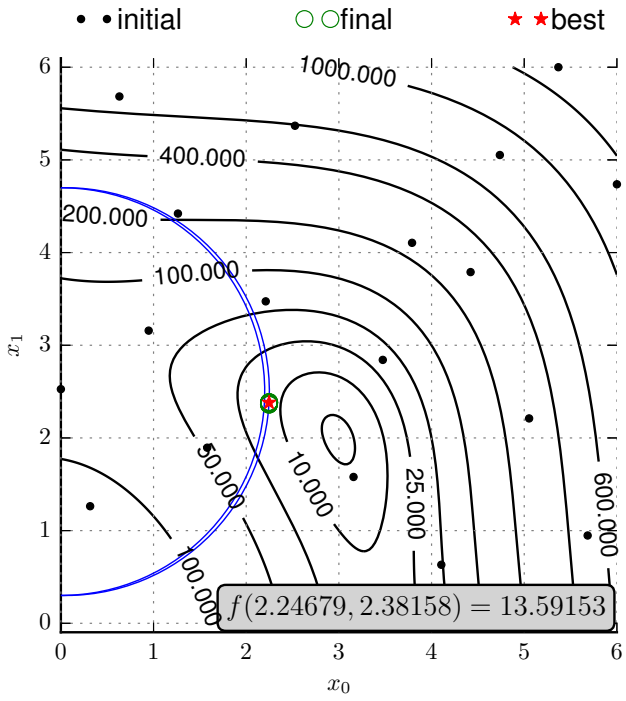



Figure 3: Problem 1: Narrow crescent-shaped region with equality constraint.

Attention is given in particular to the repeatability of results. Therefore, each problem 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 avoid much tweaking of parameters.

*Problem 1* is a simple problem presented in [15] with 2 variables. 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

$$0 \le x_i \le 6 \qquad (i \in [0, 1])$$

This example was particularly challenging for earlier codes as discussed in [15]. 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* in [13] and *test 3* in [15] and is defined by

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

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 [51] and *test 6* in [15] 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 [51] and *test 2* in [15]. The objective function is defined as follows

$$\begin{aligned}
f_0 &= -5.843 \cdot 10^{-7}\,y_{16} + 1.17 \cdot 10^{-4}\,y_{13} \\
&\quad + 2.358 \cdot 10^{-5}\,y_{12} + 1.502 \cdot 10^{-6}\,y_{15} \\
&\quad + 0.0321\,y_{11} + 0.004324\,y_4 + 10^{-4}\,c_{14}/c_{15} \\
&\quad + 37.48\,y_1/c_{11} + 0.1365
\end{aligned} \tag{20}$$

the constraints are given by

$$\begin{aligned}
g_0 &= 1.5\,x_1 - x_2 \\
g_1 &= y_0 - 213.1 \\
g_2 &= 405.23 - y_0 \\
g_{i+2} &= y_i - a_i \quad (i \in [1, 16]) \\
g_{i+18} &= b_i - y_i \quad (i \in [1, 16]) \\
g_{35} &= y_3 - 0.28/0.72\,y_4 \\
g_{36} &= 21 - 3496\,y_1/c_{11} \\
g_{37} &= 62212/c_{16} - 110.6 - y_0
\end{aligned} \tag{21}$$

and the limits of variables are

$$\begin{aligned}
704.4148 \le x_0 &\le 906.3855 \\
68.6 \le x_1 &\le 288.88 \\
0.0 \le x_2 &\le 134.75 \\
193.0 \le x_3 &\le 287.0966 \\
25.0 \le x_4 &\le 84.1988
\end{aligned} \tag{22}$$

In addition, a number of constants and coefficients are required. The $a$ and $b$ constants are given by

$$\begin{aligned}
a = \{&0,\ 17.505,\ 11.275,\ 214.228,\ 7.458,\ 0.961,\ 1.612, \\
&0.146,\ 107.99,\ 922.693,\ 926.832,\ 18.766,\ 1072.163, \\
&8961.448,\ 0.063,\ 71084.33,\ 2802713\}
\end{aligned} \tag{23}$$

and

$$\begin{aligned}
b = \{&0,\ 1053.6667,\ 35.03,\ 665.585,\ 584.463,\ 265.916,\ 7.046, \\
&0.222,\ 273.366,\ 1286.105,\ 1444.046,\ 537.141,\ 3247.039, \\
&26844.086,\ 0.386,\ 140000,\ 12146108\}
\end{aligned} \tag{24}$$

The additional dependent variables that have to be computed in

the presented order are

$$
\begin{aligned}
y_0 &= x_1 + x_2 + 41.6 \\
c_0 &= 0.024\, x_3 - 4.62 \\
y_1 &= 12.5/c_0 + 12 \\
c_1 &= 0.0003535\, x_0\, x_0 + 0.5311\, x_0 + 0.08705\, y_1\, x_0 \\
c_2 &= 0.052\, x_0 + 78 + 0.002377\, y_1\, x_0 \\
y_2 &= c_1/c_2 \\
y_3 &= 19\, y_2 \\
c_3 &= 0.04782\,(x_0 - y_2) + 0.1956\,(x_0 - y_2)\,(x_0 - y_2)/x_1 \\
&\quad + 0.6376\, y_3 + 1.594\, y_2 \\
c_4 &= 100\, x_1 \\
c_5 &= x_0 - y_2 - y_3 \\
c_6 &= 0.95 - c_3/c_4 \\
y_4 &= c_5\, c_6 \\
y_5 &= x_0 - y_4 - y_3 - y_2 \\
c_7 &= 0.995\,(y_3 + y_4) \\
y_6 &= c_7/y_0 \\
y_7 &= c_7/3798 \\
c_8 &= y_6 - 0.0663\, y_6/y_7 - 0.3153 \\
y_8 &= 96.82/c_8 + 0.321\, y_0 \\
y_9 &= 1.29\, y_4 + 1.258\, y_3 + 2.29\, y_2 + 1.71\, y_5 \\
y_{10} &= 1.71\, x_0 - 0.452\, y_3 + 0.58\, y_2 \\
c_9 &= 12.3/752.3 \\
c_{10} &= 1.75\, y_1\, 0.995\, x_0 \\
c_{11} &= 0.995\, y_9 + 1998 \\
y_{11} &= c_9\, x_0 + c_{10}/c_{11} \\
y_{12} &= c_{11} - 1.75\, y_1 \\
y_{13} &= 3623 + 64.4\, x_1 + 58.4\, x_2 + 146312/(y_8 + x_4) \\
c_{12} &= 0.995\, y_9 + 60.8\, x_1 + 48\, x_3 - 0.1121\, y_{13} - 5095 \\
y_{14} &= y_{12}/c_{12} \\
y_{15} &= 148000 - 331000\, y_{14} + 40\, y_{12} - 61\, y_{14}\, y_{12} \\
c_{13} &= 2324\, y_9 - 28740000\, y_1 \\
y_{16} &= 14130000 - 1328\, y_9 - 531\, y_{10} + c_{13}/c_{11} \\
c_{14} &= y_{12}/y_{14} - y_{12}/0.52 \\
c_{15} &= 1.104 - 0.72\, y_{14} \\
c_{16} &= y_8 + x_4
\end{aligned}
\tag{25}
$$

*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 [52] (see also [53]) where the objective function is the system cost. As shown in [53] (page 592), after substitutions, the problem is defined by

$$
\begin{aligned}
f_0 &= 1.10471\, x_0^2\, x_1 + 0.04811\, x_2\, x_3\, (14 + x_1) \\
g_0 &= \frac{\tau_d}{F} - \left\{ \frac{1}{2\, x_0^2\, x_1^2} + \frac{3\,(28 + x_1)}{x_0^2\, x_1\, c_1} \right. \\
&\quad \left. + \frac{4.5\,(28 + x_1)^2\,[x_1^2 + (x_0 + x_2)^2]}{x_0^2\, x_1^2\, c_1^2} \right\}^{1/2} \\
g_1 &= x_2\, x_2\, x_3 - 12.8 \\
g_2 &= x_3 - x_0 \\
g_3 &= x_2\, x_3^3\,(1 - 0.02823\, x_2) - 0.09267 \\
g_4 &= x_2^3\, x_3 - 8.7808
\end{aligned}
\tag{26}
$$

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

$$
\begin{aligned}
0.125 &\le x_0 \le 10 \\
0.0 &\le x_1 \le 10 \\
0.0 &\le x_2 \le 10 \\
0.125 &\le x_3 \le 10
\end{aligned}
\tag{27}
$$

*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 [51], *case 3* in [13] and *test 5* in [15]. The problem is defined by

$$
\begin{aligned}
f_0 &= (x_0 - 10)^2 + 5\,(x_1 - 12)^2 + x_2^4 + 3\,(x_3 - 11)^2 \\
&\quad + 10\, x_4^6 + 7\, x_5^2 + x_6^4 - 4\, x_5\, x_6 - 10\, x_5 - 8\, x_6 \\
g_0 &= 127 - 2\, x_0\, x_0 - 3\, x_1^4 - x_2 - 4\, x_3\, x_3 - 5\, x_4 \\
g_1 &= 282 - 7\, x_0 - 3\, x_1 - 10\, x_2\, x_2 - x_3 + x_4 \\
g_2 &= 196 - 23\, x_0 - x_1\, x_1 - 6\, x_5\, x_5 + 8\, x_6 \\
g_3 &= -4\, x_0\, x_0 - x_1\, x_1 + 3\, x_0\, x_1 - 2\, x_2\, x_2 \\
&\quad - 5\, x_5 + 11\, x_6
\end{aligned}
\tag{28}
$$

According to [51], there are no bounds for the variables. Nonetheless, the search space is defined as suggested in [13, 15]. Thus, the limits for variables are

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

*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 [51], *case 5* in [13] and *test 8* in [15]. The objective function for this problem is defined by

$$
\begin{aligned}
f_0 &= x_0^2 + x_1^2 + x_0\, x_1 - 14\, x_0 - 16\, x_1 + (x_2 - 10)^2 + 4\,(x_3 - 5)^2 + (x_4 - 3)^2 \\
&\quad + 2\,(x_5 - 1)^2 + 5\, x_6^2 + 7\,(x_7 - 11)^2 + 2\,(x_8 - 10)^2 + (x_9 - 7)^2 + 45
\end{aligned}
\tag{29}
$$

As in the previous problem, there are no bounds for the variables. Thus, the following limits as in [13, 15] are adopted

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

The constraints for problem 7 are given by

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

*Problem 8* has 8 variables, 6 inequalities and is a difficult problem as observed in [14, 15]. It has a linear objective function and nonlinear constraints and corresponds to *problem 106 (page 115; heat exchanger design)* in [51], *case 2* in [13] and *test 4* in [15]. 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

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

*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 [51], *case 4* in [13] and *test 7* in [15]. 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 (with a different seed), and the evolution is run all over again. The results are collected in Table 1 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 $1000 \times 500$ loops. The resulting number of function evaluations *per* sample is $N_{eval}$; calling $f_i$, $g_i$, and $h_i$ at the same time counts as 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 $10\,N_x$—ten times the number of variables as in [15]. 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 the number of solutions in the same group (no more than 50). When needed, the exchange time $\Delta t_{exc}$ is simply set equal to $\Delta t_{exc} = t_f/10$.

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 9 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 problems 1 to 7, minimum values ($f_{min}$) close to the reference ones are obtained. Problem 8 is more difficult to be solved and requires longer evolution times.

In problem 2, the failure happens because the optimal solution is actually at the boundaries of the search space. In this case, the solution reaches the maximum limit. This can be improved 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 variables 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 2 and are quite good now.

Table 1: Constrained single objective problems.

| P | settings | settings/info | objective | histogram ($N_{samples}$ = 1000) |
|---|---|---|---|---|
| 1 | $N_{sol} = 20$ | $f_{ref} = \mathbf{13.59085}$ | $f_{min} = \mathbf{13.59084}$ | [13.54,13.56) \| 0<br>[13.56,13.58) \| 0<br>[13.58,13.60) \| 1000 ##############<br>[13.60,13.62) \| 0<br>[13.62,13.64) \| 0<br>count = 1000 |
| | $N_{cpu} = 1$ | $C_{DE} = 0.8$ | $f_{ave} = 13.59084$ | |
| | $t_{max} = 500$ | $N_{eval} = 10020$ | $f_{max} = 13.59084$ | |
| | $\Delta t_{exc} = 1$ | $T_{sys} = 18.511s$ | $f_{dev} = \mathbf{1.15 \cdot 10^{-9}}$ | |
| | | | $x_{best}$=[2.24683 2.38186] | |
| | | | $x_{ref.}$=[2.24683 2.38186] | |
| 2 | $N_{sol} = 130$ | $f_{ref} = \mathbf{-15.00000}$ | $f_{min} = \mathbf{-15.00000}$ | [-15.05,-14.43) \| 991 ##############<br>[-14.43,-13.81) \| 1 #<br>[-13.81,-13.19) \| 0<br>[-13.19,-12.57) \| 2 #<br>[-12.57,-11.95) \| 6 #<br>count = 1000 |
| | $N_{cpu} = 4$ | $C_{DE} = 0.8$ | $f_{ave} = -14.97683$ | |
| | $t_{max} = 500$ | $N_{eval} = 64130$ | $f_{max} = -12.00000$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 3m12.349s$ | $f_{dev} = \mathbf{2.51 \cdot 10^{-1}}$ | |
| | | $x_{best}$=[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] | | |
| 3 | $N_{sol} = 50$ | $f_{ref} = \mathbf{-30665.50}$ | $f_{min} = \mathbf{-30665.54}$ | [-30665.59,-30665.57) \| 0<br>[-30665.57,-30665.55) \| 0<br>[-30665.55,-30665.53) \| 1000 ######<br>[-30665.53,-30665.51) \| 0<br>[-30665.51,-30665.49) \| 0<br>count = 1000 |
| | $N_{cpu} = 1$ | $C_{DE} = 0.8$ | $f_{ave} = -30665.54$ | |
| | $t_{max} = 500$ | $N_{eval} = 25050$ | $f_{max} = -30665.54$ | |
| | $\Delta t_{exc} = 1$ | $T_{sys} = 2m27.616s$ | $f_{dev} = \mathbf{1.25 \cdot 10^{-12}}$ | |
| | | $x_{best}$=[78.00000 33.00000 29.99526 45.00000 36.77581] | | |
| | | $x_{ref.}$=[78.00000 33.00000 29.99500 45.00000 36.77600] | | |
| 4 | $N_{sol} = 50$ | $f_{ref} = \mathbf{-1.90513}$ | $f_{min} = \mathbf{-1.90516}$ | [-1.96,-1.94) \| 0<br>[-1.94,-1.92) \| 0<br>[-1.92,-1.90) \| 1000 ##############<br>[-1.90,-1.88) \| 0<br>[-1.88,-1.85) \| 0<br>count = 1000 |
| | $N_{cpu} = 1$ | $C_{DE} = 0.8$ | $f_{ave} = -1.90516$ | |
| | $t_{max} = 500$ | $N_{eval} = 25050$ | $f_{max} = -1.90516$ | |
| | $\Delta t_{exc} = 1$ | $T_{sys} = 2m38.08s$ | $f_{dev} = \mathbf{2.89 \cdot 10^{-10}}$ | |
| | | $x_{best}$=[705.17454 68.60000 102.90000 282.32493 37.58412] | | |
| | | $x_{ref.}$=[705.18030 68.60005 102.90001 282.32500 37.58504] | | |
| 5 | $N_{sol} = 40$ | $f_{ref} = \mathbf{2.34027}$ | $f_{min} = \mathbf{2.34021}$ | [2.29,2.31) \| 0<br>[2.31,2.33) \| 0<br>[2.33,2.35) \| 1000 ##############<br>[2.35,2.37) \| 0<br>[2.37,2.39) \| 0<br>count = 1000 |
| | $N_{cpu} = 1$ | $C_{DE} = 0.8$ | $f_{ave} = 2.34021$ | |
| | $t_{max} = 500$ | $N_{eval} = 20040$ | $f_{max} = 2.34021$ | |
| | $\Delta t_{exc} = 1$ | $T_{sys} = 1m24.611s$ | $f_{dev} = \mathbf{1.01 \cdot 10^{-9}}$ | |
| | | $x_{best}$=[0.25364 7.14155 7.10391 0.25364] | | |
| | | $x_{ref.}$=[0.25360 7.14100 7.10440 0.25360] | | |
| 6 | $N_{sol} = 70$ | $f_{ref} = \mathbf{680.63006}$ | $f_{min} = \mathbf{680.63006}$ | [680.58,680.60) \| 0<br>[680.60,680.62) \| 0<br>[680.62,680.64) \| 1000 ##############<br>[680.64,680.67) \| 0<br>[680.67,680.69) \| 0<br>count = 1000 |
| | $N_{cpu} = 2$ | $C_{DE} = 0.8$ | $f_{ave} = 680.63027$ | |
| | $t_{max} = 500$ | $N_{eval} = 34070$ | $f_{max} = 680.63692$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 1m33.628s$ | $f_{dev} = \mathbf{3.762 \cdot 10^{-4}}$ | |
| | | $x_{best}$=[2.33043 1.95132 -0.47792 4.36587 -0.62434 1.03832 1.59431] | | |
| | | $x_{ref.}$=[2.33050 1.95137 -0.47754 4.36573 -0.62449 1.03813 1.59423] | | |
| 7 | $N_{sol} = 100$ | $f_{ref} = \mathbf{24.30621}$ | $f_{min} = \mathbf{24.31374}$ | [24.26,24.31) \| 3 #<br>[24.31,24.37) \| 939 ##############<br>[24.37,24.42) \| 53 #<br>[24.42,24.47) \| 4 #<br>[24.47,24.52) \| 1 #<br>count = 1000 |
| | $N_{cpu} = 2$ | $C_{DE} = 0.8$ | $f_{ave} = 24.33941$ | |
| | $t_{max} = 500$ | $N_{eval} = 50100$ | $f_{max} = 24.46721$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 3m51.452s$ | $f_{dev} = \mathbf{0.01640}$ | |
| | | $x_{best}$=[2.17119 2.36631 8.77119 5.09315 0.99958 1.45022 1.31990 9.82703 8.27854 8.37838] | | |
| | | $x_{ref.}$=[2.17200 2.36368 8.77393 5.09598 0.99065 1.43057 1.32164 9.82873 8.28009 8.37593] | | |
| 8 | $N_{sol} = 80$ | $f_{ref} = \mathbf{7049.33092}$ | $f_{min} = \mathbf{7056.23004}$ | [7056.18,7152.67) \| 287 ##########<br>[7152.67,7249.15) \| 365 ############<br>[7249.15,7345.64) \| 333 ###########<br>[7345.64,7442.12) \| 11 #<br>[7442.12,7538.61) \| 4 #<br>count = 1000 |
| | $N_{cpu} = 2$ | $C_{DE} = 0.8$ | $f_{ave} = 7201.83708$ | |
| | $t_{max} = 500$ | $N_{eval} = 40080$ | $f_{max} = 7538.55603$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 2m6.579s$ | $f_{dev} = \mathbf{71.34316}$ | |
| | $x_{best}$=[612.84830 1252.35030 5191.03143 184.74401 292.41085 215.25368 292.29725 392.40929] | | | |
| | $x_{ref.}$=[579.31670 1359.94300 5110.07100 182.01740 295.59850 217.97990 286.41620 395.59790] | | | |
| 9 | $N_{sol} = 50$ | $f_{ref} = \mathbf{0.05395}$ | $f_{min} = \mathbf{0.05387}$ | [0.00,0.66) \| 507 ##############<br>[0.66,1.31) \| 48 ##<br>[1.31,1.97) \| 0<br>[1.97,2.62) \| 0<br>[2.62,3.28) \| 1 #<br>count = 556 |
| | $N_{cpu} = 1$ | $C_{DE} = 0.8$ | $f_{ave} = 0.23522$ | |
| | $t_{max} = 500$ | $N_{eval} = 25050$ | $f_{max} = 3.22740$ | |
| | $\Delta t_{exc} = 1$ | $T_{sys} = 2m32.787s$ | $f_{dev} = \mathbf{0.26948}$ | |
| | | $x_{best}$=[-1.71408 1.59228 1.83296 0.76407 0.76421] | | |
| | | $x_{ref.}$=[-1.71714 1.59571 1.82725 -0.76364 -0.76365] | | |

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 Eq. (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 $\epsilon_h$. Problem 9 is run again with 7000 iterations as in [15]. The results are collected in Table 2 where it can bee seen the minimum is found with higher frequency. By tweaking the parameters and extending the number of iterations the results would improve; but this has not been pursued.

## 7. Test cases: unconstrained two-objective

Multi objective optimisation problems may be more challenging than single objective problems at times, especially if

Table 2: Constrained single objective problems: Problems 2 and 9

| P | settings | settings/info | objective | histogram ($N_{samples}$ = 1000) |
|---|---|---|---|---|
| 2 | $N_{sol} = 130$ | $f_{ref} = \mathbf{-15.00000}$ | $f_{min} = \mathbf{-14.99966}$ | [-15.05,-15.03) \| 0<br>[-15.03,-15.01) \| 0<br>[-15.01,-14.99) \| 1000 ##########<br>[-14.99,-14.97) \| 0<br>[-14.97,-14.95) \| 0<br>count = 1000 |
| | $N_{cpu} = 4$ | $C_{DE} = 0.8$ | $f_{ave} = -14.99907$ | |
| | $t_{max} = 500$ | $N_{eval} = 64130$ | $f_{max} = -14.99795$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 3m17.206s$ | $f_{dev} = \mathbf{2.67 \cdot 10^{-4}}$ | |
| | $x_{best}$=[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 ##########<br>[0.15,0.30) \| 0<br>[0.30,0.45) \| 58 ##<br>[0.45,0.60) \| 0<br>[0.60,0.75) \| 0<br>[0.75,0.90) \| 0<br>[0.90,1.05) \| 10 #<br>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] | | | |

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 (two-objective problems only).

In this section, unconstrained problems with two objective functions $f_0(\underset{\cdot}{\boldsymbol{x}})$ and $f_1(\underset{\cdot}{\boldsymbol{x}})$ are considered; six problems are studied. Five of them, named *ZDTi* were proposed in [26]. These problems include concave and convex curves in the $(f_0, f_1)$ space and situations of multiple local minima. The interesting problem *FON* [23] in the form presented in [3] is also studied.

Except for *FON*, in the problems presented in this section, $C_{DE}$ has been reduced to $C_{DE} = 0.1$ because it was found that this value produces better results (accuracy and spread). With $C_{DE} = 0.8$, the results are not bad though; however *ZDT4* requires more iterations due to 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

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

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

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

where the limits of variables are

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

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

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

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

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

where the limits of variables are

$$0 \le x_i \le 1 \quad (i \in [0, N_x - 1]) \tag{51}$$

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

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

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

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

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

*Problem FON* has a somewhat concave Pareto-optimal front in part of the domain (except near $f_0 = 0$ or $f_0 = 1$) and has moderate difficulty. It is defined by

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

where the limits of variables are

$$0 \le x_i \le 1 \qquad (i \in [0, N_x - 1]) \tag{55}$$

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

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

for which a line integral in $0 \le f_0 \le 0.98$ 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

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

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

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

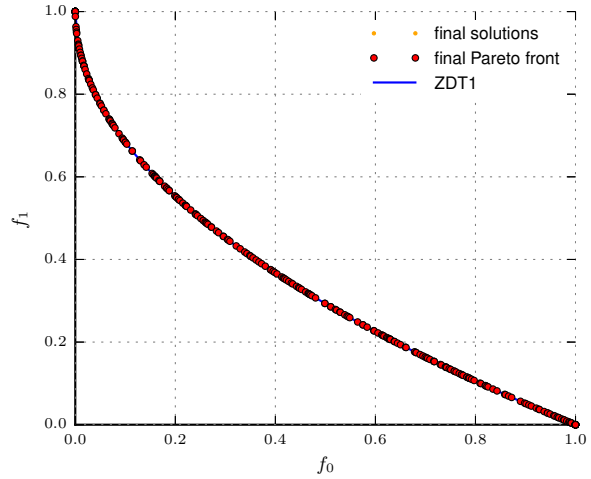
where $n$ indicates the number of *feasible* solutions on the optimal front with $\phi_i = 0$. Therefore, contrary to the error measure, the spread is better if closer to 1. Note that the above measure does not account for how equally spaced points are.

Table 3 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 ($E$) and spread ($L$). Plots of Pareto-optimal fronts for the six problems are presented in Fig. 4 corresponding to one (any) sample.

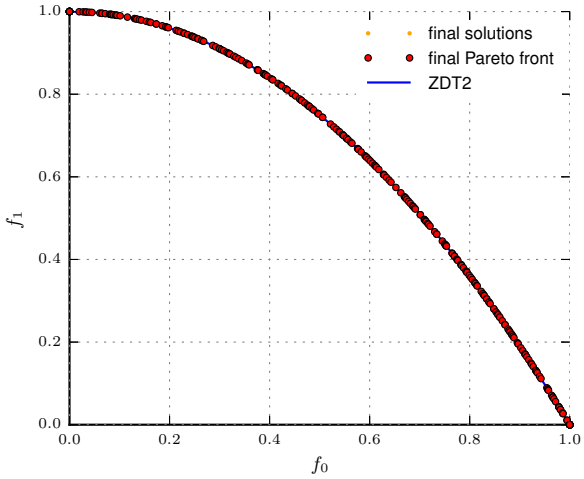
## 8. Test cases: constrained two objective

The next set of tests deals with constrained two objective optimisation problems. In this case, the accuracy is assessed by computing a scalar function $\psi(f_0, f_1, f_2, \ldots) = 0$ expressing the optimal front with the numerical results. A root mean square error is thus computed by means of

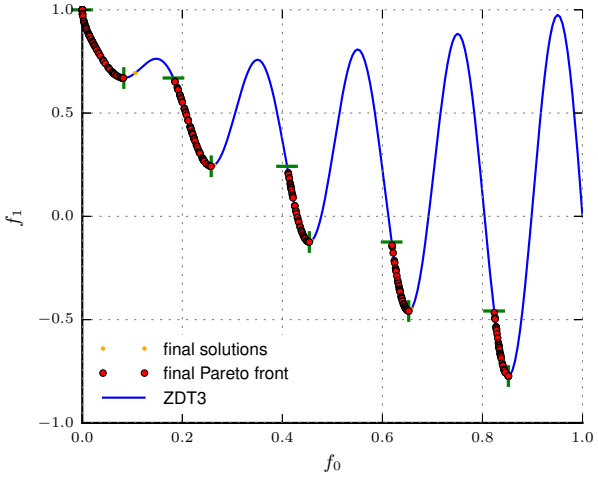
$$E = \sqrt{\frac{1}{n}\left[\sum_{i=0}^{n-1}\psi(f_0^i, f_1^i, f_2^i, \ldots)^2\right]} \tag{59}$$



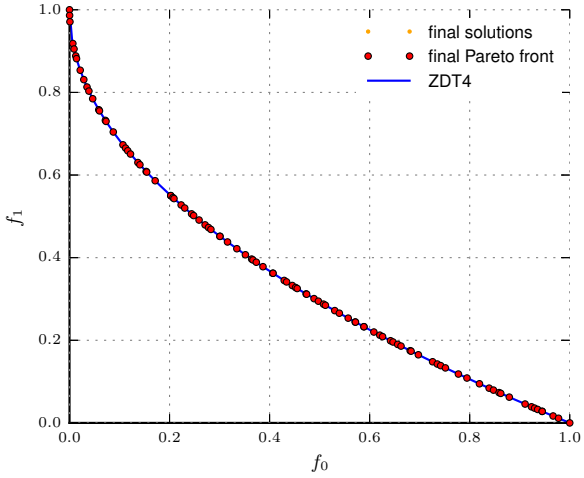

(a) ZDT1



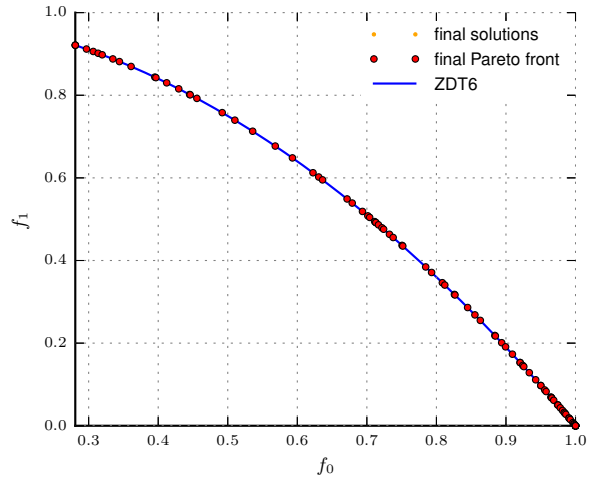

(b) ZDT2



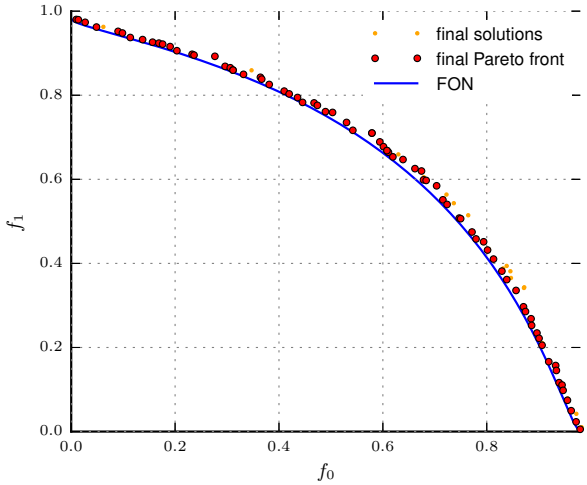

(c) ZDT3




(d) ZDT4




(e) ZDT6




(f) FON

Figure 4: Unconstrained two-objective problems.

Table 3: Unconstrained two objective problems.

| P | settings | settings/info | error | spread |
|---|---|---|---|---|
| ZDT1 | $N_{sol} = 300$ | $count = 1000$ | $E_{min} = 0.0000$ | $L_{min} = 9.9993 \cdot 10^{-1}$ |
| | $N_{cpu} = 6$ | $C_{DE} = 0.1$ | $E_{ave} = 0.0000$ | $L_{ave} = 9.9999 \cdot 10^{-1}$ |
| | $t_{max} = 500$ | $N_{eval} = 150300$ | $E_{max} = 0.0000$ | $L_{max} = 1.0000$ |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 20m1.972s$ | $E_{dev} = \mathbf{0.000}$ | $L_{dev} = \mathbf{5.924 \cdot 10^{-6}}$ |
| ZDT2 | $N_{sol} = 300$ | $count = 1000$ | $E_{min} = 0.0000$ | $L_{min} = 9.9999 \cdot 10^{-1}$ |
| | $N_{cpu} = 6$ | $C_{DE} = 0.1$ | $E_{ave} = 0.0000$ | $L_{ave} = 1.0000$ |
| | $t_{max} = 500$ | $N_{eval} = 150300$ | $E_{max} = 0.0000$ | $L_{max} = 1.0000$ |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 20m9.505s$ | $E_{dev} = \mathbf{0.000}$ | $L_{dev} = \mathbf{9.385 \cdot 10^{-7}}$ |
| ZDT3 | $N_{sol} = 300$ | $count = 1000$ | $E_{min} = 0.0000$ | $L_{min} = 9.1117 \cdot 10^{-1}$ |
| | $N_{cpu} = 6$ | $C_{DE} = 0.1$ | $E_{ave} = 0.0000$ | $L_{ave} = 9.7213 \cdot 10^{-1}$ |
| | $t_{max} = 500$ | $N_{eval} = 150300$ | $E_{max} = 0.0000$ | $L_{max} = 1.4139$ |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 19m31.8s$ | $E_{dev} = \mathbf{0.000}$ | $L_{dev} = \mathbf{1.887 \cdot 10^{-2}}$ |
| ZDT4 | $N_{sol} = 100$ | $count = 1000$ | $E_{min} = 3.8190 \cdot 10^{-11}$ | $L_{min} = 9.9986 \cdot 10^{-1}$ |
| | $N_{cpu} = 2$ | $C_{DE} = 0.1$ | $E_{ave} = 1.7571 \cdot 10^{-9}$ | $L_{ave} = 9.9997 \cdot 10^{-1}$ |
| | $t_{max} = 500$ | $N_{eval} = 50100$ | $E_{max} = 3.3731 \cdot 10^{-8}$ | $L_{max} = 9.9998 \cdot 10^{-1}$ |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 6m52.5s$ | $E_{dev} = \mathbf{2.554 \cdot 10^{-9}}$ | $L_{dev} = \mathbf{1.107 \cdot 10^{-5}}$ |
| FON | $N_{sol} = 100$ | $count = 1000$ | $E_{min} = 1.1448 \cdot 10^{-2}$ | $L_{min} = 1.0104$ |
| | $N_{cpu} = 2$ | $C_{DE} = 0.8$ | $E_{ave} = 1.6708 \cdot 10^{-2}$ | $L_{ave} = 1.0363$ |
| | $t_{max} = 500$ | $N_{eval} = 50100$ | $E_{max} = 3.6715 \cdot 10^{-2}$ | $L_{max} = 1.3273$ |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 7m17.575s$ | $E_{dev} = \mathbf{1.944 \cdot 10^{-3}}$ | $L_{dev} = \mathbf{1.835 \cdot 10^{-2}}$ |
| ZDT6 | $N_{sol} = 100$ | $count = 1000$ | $E_{min} = 0.0000$ | $L_{min} = 9.9612 \cdot 10^{-1}$ |
| | $N_{cpu} = 2$ | $C_{DE} = 0.1$ | $E_{ave} = 0.0000$ | $L_{ave} = 9.9998 \cdot 10^{-1}$ |
| | $t_{max} = 500$ | $N_{eval} = 50100$ | $E_{max} = 0.0000$ | $L_{max} = 1.0000$ |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 6m26.332s$ | $E_{dev} = \mathbf{0.000}$ | $L_{dev} = \mathbf{1.474 \cdot 10^{-4}}$ |

where $f_j^i$ is the numerical solution $i$ for objective function $j$ and $n$ indicates the number of feasible solutions on the optimal front with $\phi_i = 0$.

A series of problems presented in [3] are considered. These problems are symbolised by *CTPi* and include some with local minima and nonuniform distribution of points on the Pareto-optimal fronts. The problem *TNK* from [22] is studied as well.

Nine problems are considered as described below. The number of solutions for these problems is chosen as $N_{sol} = 120$ in order to better draw the optimal front. The differential evolution coefficient $C_{DE}$ is set equal to $C_{DE} = 0.1$ because this value seemed to give better results when solving the problems in this section. A statistical analysis is presented just after all problems are described.

*Problem TNK* [22] has two constraints and a discontinuous optimal front. $N_x = 2$ and the problem is defined by

$$\begin{aligned} f_0 &= x_0 \\ f_1 &= x_1 \\ g_0 &= x_0^2 + x_1^2 - 1 - 0.1 \cos\left[16 \operatorname{atan}\left(x_0/x_1\right)\right] \\ g_1 &= 0.5 - (x_0 - 0.5)^2 - (x_1 - 0.5)^2 \end{aligned} \tag{60}$$

where the limits of variables are

$$0 \leq x_i \leq \pi \quad (i \in [0, 1]) \tag{61}$$

The results are plotted in Fig. 5 where it can be seen a good response: accurate and spread. In addition, all solutions satisfy the constraints.

The next *CTPi* problems are all defined with $N_x = 10$ and two groups ($N_{cpu} = 2$). The limits of variables for all of them

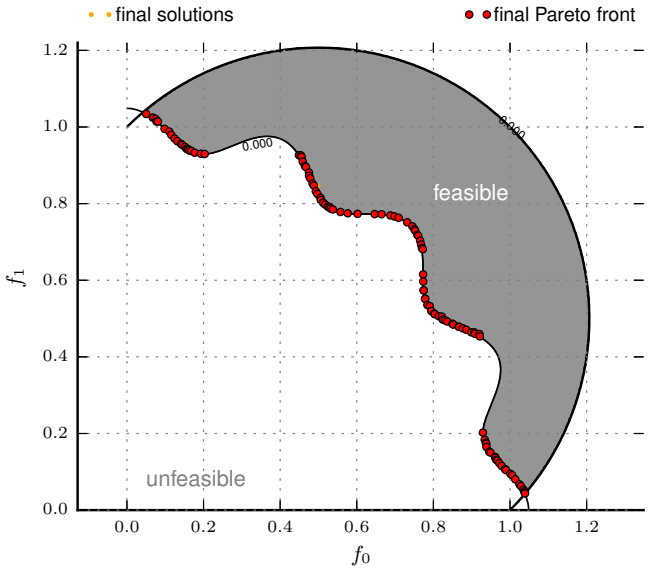



Figure 5: Constrained two objective test: TNK.

are

$$0 \leq x_i \leq 1 \quad (i \in [0, N_x - 1]) \tag{62}$$

*Problem CTP1* [3], as considered in this work, has two con-

Table 4: Coefficients for CTP2 to CTP7.

| P | $\theta$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|---|
| CTP2 | $-0.2\,\pi$ | 0.2 | 10.0 | 1.0 | 6.0 | 1.0 |
| CTP3 | $-0.2\,\pi$ | 0.1 | 10.0 | 1.0 | 0.5 | 1.0 |
| CTP4 | $-0.2\,\pi$ | 0.75 | 10.0 | 1.0 | 0.5 | 1.0 |
| CTP5 | $-0.2\,\pi$ | 0.1 | 10.0 | 2.0 | 0.5 | 1.0 |
| CTP6 | $0.1\,\pi$ | 40.0 | 0.5 | 1.0 | 2.0 | $-2.0$ |
| CTP7 | $-0.05\,\pi$ | 40.0 | 5.0 | 1.0 | 6.0 | 0.0 |

straints and is defined by

$$\begin{aligned} c_0 &= 1 + \textstyle\sum_{i=1}^{N_x-1} x_i \\ f_0 &= x_0 \\ f_1 &= c_0 \, \exp\left(-x_0/c_0\right) \\ g_0 &= f_1 - a_0 \, \exp\left(-b_0 \, f_0\right) \\ g_1 &= f_1 - a_1 \, \exp\left(-b_1 \, f_0\right) \end{aligned} \tag{63}$$

where the following auxiliary coefficients are selected

$$a_0 = 0.858 \quad b_0 = 0.541 \quad a_1 = 0.728 \quad b_1 = 0.295 \tag{64}$$

Good results are obtained as illustrated in Fig. 6a—a statistical analysis is given shortly.

*Problems CTP2-CTP7* [3] have one constraint and, in this work, are all defined by the following expressions

$$\begin{aligned} c_0 &= 1 + \textstyle\sum_{i=1}^{N_x-1} x_i \\ f_0 &= x_0 \\ f_1 &= c_0 \left(1 - f_0/c_0\right) \\ c_1 &= \cos\left(\theta\right)\left(f_1 - e\right) - \sin\left(\theta\right) f_0 \\ c_2 &= \sin\left(\theta\right)\left(f_1 - e\right) + \cos\left(\theta\right) f_0 \\ c_3 &= \sin\left(b\,\pi\,(c_2)^c\right) \\ g_0 &= c_1 - a\,(|c_3|)^d \end{aligned} \tag{65}$$

where $a$, $b$, $c$, $d$, $e$ and $\theta$ are auxiliary coefficients that help to design many combinations. The selected values are listed in Table 4.

*Problem CTP2* has a number of narrow discontinuous Pareto-optimal fronts as illustrated in Fig. 6b. The presented code can obtain satisfactory results every time it is run.

*Problem CTP3* is more challenging because the optimal front reduces to a set of single points (Fig. 6c).

*Problem CTP4* (Fig. 6d) is also more challenging because the 'path' available for points to move towards the best front is very narrow. The proposed code is able to obtain some reasonable results.

*Problem CTP5* is similar to Problem CTP3 but has a nonuniform distribution of points along the optimal front. The results are plotted in Fig. 6e and the response is similar to CTP3 where the points do approach the optimal front with good spread.

*Problem CTP6* has bands defining feasible and infeasible regions; hence additional problems would arise if the solver was not able to properly handle and 'jump over' constraints. Fig. 6f show the results where the optimal front lies along the constraint line.

*Problem CTP7* (Fig. 7a) has also multiple infeasible regions. The Pareto-optimal front is discontinuous and follows a straight line defined by $f_1 = 1 - f_0$.

*Problem CTP8* [3] is now introduced. This problem is similar to the previous CTP ones; however two constraints using the same generator are considered. For the sake of clarity, the equations are written below

$$\begin{aligned} c_1 &= \cos\left(\theta_1\right)\left(f_1 - e\right) - \sin\left(\theta_1\right) f_0 \\ c_2 &= \sin\left(\theta_1\right)\left(f_1 - e\right) + \cos\left(\theta_1\right) f_0 \\ c_3 &= \sin\left(b\,\pi\,(c_2)^c\right) \\ g_0 &= c_1 - a\,(|c_3|)^d \\ d_1 &= \cos\left(\theta_2\right)\left(f_1 - E\right) - \sin\left(\theta_2\right) f_0 \\ d_2 &= \sin\left(\theta_2\right)\left(f_1 - E\right) + \cos\left(\theta_2\right) f_0 \\ d_3 &= \sin\left(B\,\pi\,(d_2)^C\right) \\ g_1 &= d_1 - A\,(|d_3|)^D \end{aligned} \tag{66}$$

where $c_0$, $f_0$ and $f_1$ are the same as in Eq. (65). The auxiliary coefficients are

$$\begin{aligned} a &= 40 \quad & b &= 0.5 \quad & c &= 1 \quad & d &= 2 \quad & e &= -2 \\ A &= 40 \quad & B &= 2.0 \quad & C &= 1 \quad & D &= 6 \quad & E &= \;\; 0 \end{aligned} \tag{67}$$

and

$$\theta_1 = 0.1\,\pi \qquad \theta_2 = -0.05\,\pi \tag{68}$$

The results are presented in Figs. 6 and 7 where it can be observed that there are 'islands' of feasible and infeasible regions and that the optimal front is discontinuous. The proposed code seems to also work well in this case.

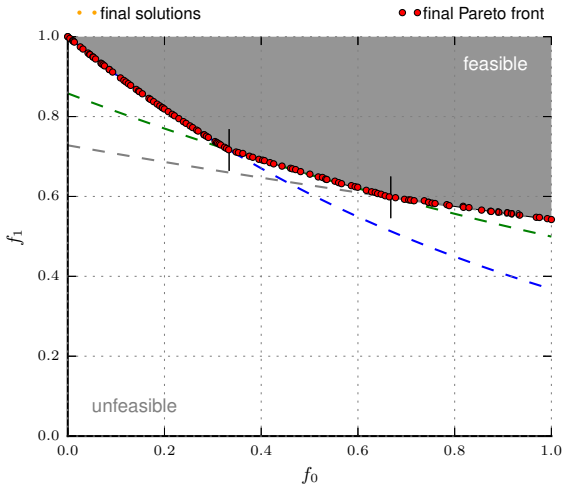
Each problem is run 1000 times (samples) generating data for the statistical analysis. Once again, $T_{sys}$ is the computer (system) time spent in running *all 1000 samples* of one particular problem; hence considering the $1000 \times 500$ loops.

The statistics with respect to error are presented in Table 5. It can be observed that the accuracy is reasonable (small $E$) and the repeatability characteristics are quite good for all tests. The spread has been also observed in several runs—every time TNK or CTPi are run, Figs. 5 and 6a-7b exhibit similar results.

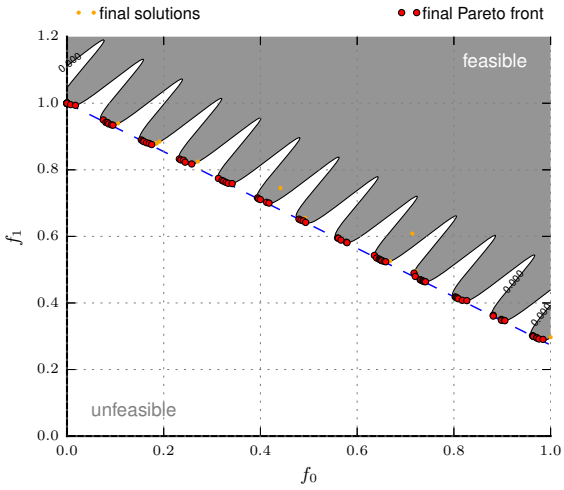
## 9. Test cases: three objective

Nine three-objective optimisation problems are studied in this section. Eight tests are unconstrained (except for the limits on $\boldsymbol{x}$) and one has one constraint. As before, the number of iterations is $t_{max} = 500$. The number of trial solutions is set to $N_{sol} = 200$ for all tests in order to better depict the optimal Pareto front in the 3D space defined by $(f_0, f_1, f_2)$.

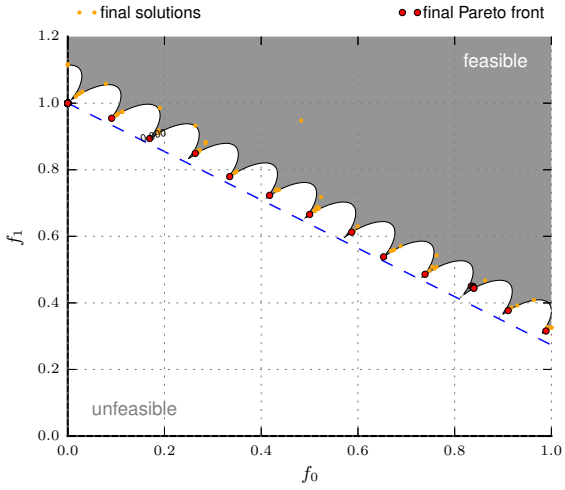
The set of tests introduced in [30] are considered. These tests are known as *DTZLi*. In addition, the convex version of *DTZL2* presented in [54, 55] is solved. This version is named here *DTZL2x*. A constraint expression is added to *DTZL2* and the resulting test named *DTZL2c*. Finally, a modification to



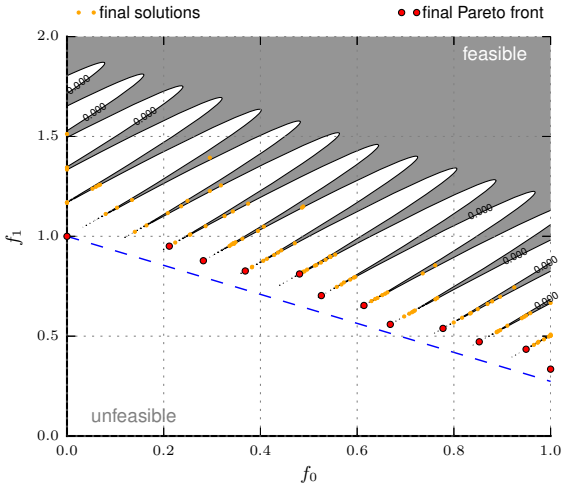

(a) CTP1



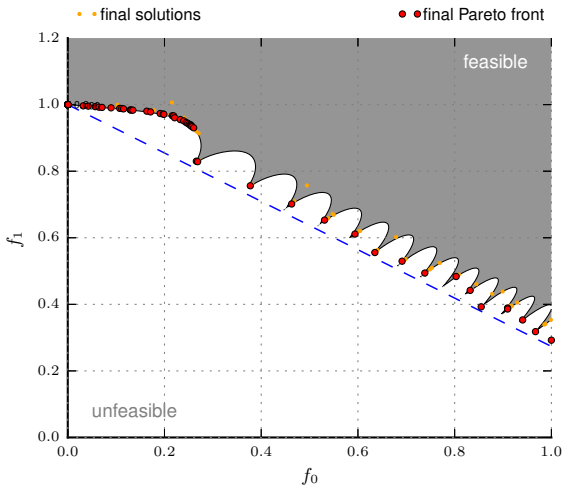

(b) CTP2



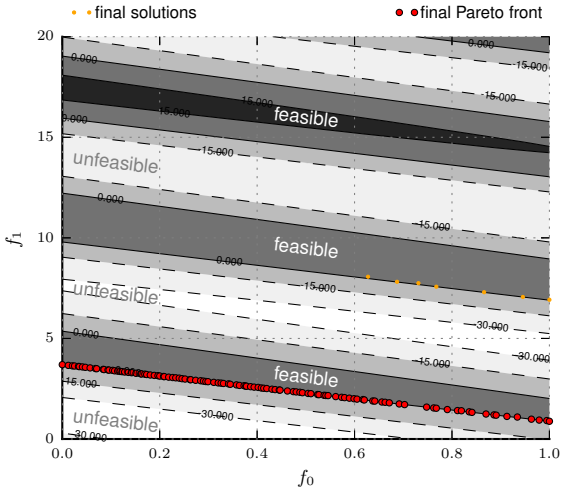

(c) CTP3




(d) CTP4

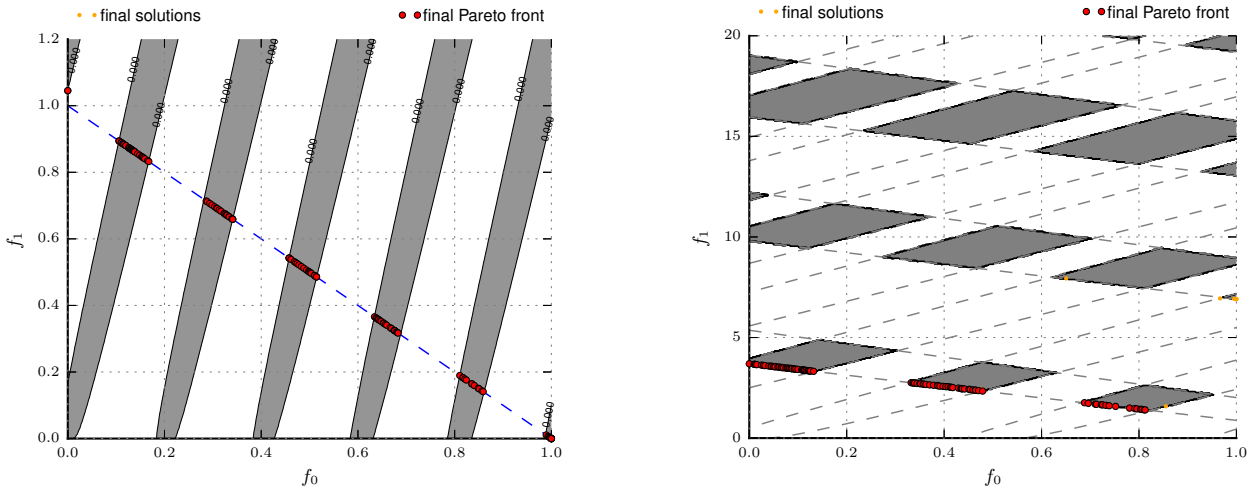



(e) CTP5




(f) CTP6

Figure 6: Constrained two-objective problems.




(a) CTP7

(b) CTP8

Figure 7: Constrained two-objective problems (contd.).

*DTLZi* is applied by using the expressions of a superquadric—the resulting tests are named here as *SUQ1* and *SUQ2*. In all problems, the variables are limited by

$$0 \le x_i \le 1 \quad (i \in [0, N_x - 1]) \tag{69}$$

*Problem DTZL1* has 7 variables and is defined by

$$\begin{aligned} c &= 100 \left\{ 5 + \sum_{i=2}^{6} \left[ (x_i - 0.5)^2 - \cos\left(20\,\pi\,(x_i - 0.5)\right) \right] \right\} \\ f_0 &= x_0\, x_1\, (1 + c)/2 \\ f_1 &= x_0\, (1 - x_1)\, (1 + c)/2 \\ f_2 &= (1 - x_0)\, (1 + c)/2 \end{aligned} \tag{70}$$

The optimal Pareto front is a flat surface described by

$$\psi(f_0, f_1, f_2) = f_0 + f_1 + f_2 - 0.5 \tag{71}$$

The results of one run are illustrated in Figs. 8a and 8b where it can be seen that the trial solutions (spheres) are close to the optimal front indicated by the plane. A table is presented shortly with a statistical analysis.

*Problem DTZL2* has 12 variables and is defined by

$$\begin{aligned} c &= \textstyle\sum_{i=2}^{11} (x_i - 0.5)^2 \\ f_0 &= (1 + c)\, \cos\left(0.5 x_0 \pi\right)\, \cos\left(0.5 x_1 \pi\right) \\ f_1 &= (1 + c)\, \cos\left(0.5 x_0 \pi\right)\, \sin\left(0.5 x_1 \pi\right) \\ f_2 &= (1 + c)\, \sin\left(0.5 x_0 \pi\right) \end{aligned} \tag{72}$$

The optimal Pareto front is the surface of a sphere described by

$$\psi(f_0, f_1, f_2) = f_0^2 + f_1^2 + f_2^2 - 1 \tag{73}$$

The solutions are illustrated in Fig. 8c and 8d and also allow to conclude that the results are reasonable.

*Problem DTLZ3* has 12 variables, is similar to the previous problem; but is more difficult because it has a number of local fronts parallel to the global one. The problem is defined by

$$\begin{aligned} c &= 100 \left\{ 10 + \sum_{i=2}^{11} \left[ (x_i - 0.5))^2 - \cos\left(20\,\pi\,(x_i - 0.5)\right) \right] \right\} \\ f_0 &= (1 + c)\, \cos\left(0.5 x_0 \pi\right)\, \cos\left(0.5 x_1 \pi\right) \\ f_1 &= (1 + c)\, \cos\left(0.5 x_0 \pi\right)\, \sin\left(0.5 x_1 \pi\right) \\ f_2 &= (1 + c)\, \sin\left(0.5 x_0 \pi\right) \end{aligned} \tag{74}$$

where the optimal front is also the surface of a sphere as in DTLZ2. The results are very similar to the ones from DTLZ2.

*Problem DTLZ4* has 12 variables and is similar to DTLZ2; however with a bias in the search space that could cause solutions to be concentrated at planes normal to the coordinates axes. The problem is defined by

$$\begin{aligned} c &= \textstyle\sum_{i=2}^{11} (x_i - 0.5)^2 \\ f_0 &= (1 + c)\, \cos\left(0.5 x_0^{\alpha} \pi\right)\, \cos\left(0.5 x_1^{\alpha} \pi\right) \\ f_1 &= (1 + c)\, \cos\left(0.5 x_0^{\alpha} \pi\right)\, \sin\left(0.5 x_1^{\alpha} \pi\right) \\ f_2 &= (1 + c)\, \sin\left(0.5 x_0^{\alpha} \pi\right) \end{aligned} \tag{75}$$

with $\alpha = 100$ and the optimal front is the surface of a sphere as in DTL2. The results are also similar to the ones from DTLZ2.

*Problem DTLZ2x* has 12 variables and is a convex version to DTLZ2—an exact convex counterpart is given in SUQ1 below. The problem is defined by

$$\begin{aligned} c &= \textstyle\sum_{i=2}^{11} (x_i - 0.5)^2 \\ f_0 &= \left[(1 + c)\, \cos\left(0.5 x_0 \pi\right)\, \cos\left(0.5 x_1 \pi\right)\right]^4 \\ f_1 &= \left[(1 + c)\, \cos\left(0.5 x_0 \pi\right)\, \sin\left(0.5 x_1 \pi\right)\right]^4 \\ f_2 &= \left[(1 + c)\, \sin\left(0.5 x_0 \pi\right)\right]^2 \end{aligned} \tag{76}$$

Note the exponents 4 and 2 that basically modify DTLZ2. The optimal Pareto front is described by

$$\psi(f_0, f_1, f_2) = \|f_0\|^{0.5} + \|f_1\|^{0.5} + f_2 - 1 \tag{77}$$

and the results are illustrated in Fig. 8e and 8f where a good convergence can be observed with the points reaching the optimal front.

Table 5: Constrained two objective problems.

| P | settings | settings/info | error | histogram ($N_{samples}$ = 10) |
|---|---|---|---|---|
| TNK | $N_{sol} = 120$ | $N_f = 2$ | $E_{min} = 4.4978 \cdot 10^{-3}$ | [-0.05,-0.03) \| 0<br>[-0.03,-0.00) \| 0<br>[-0.00,0.02) \| 10 ##########<br>[0.02,0.04) \| 0<br>[0.04,0.06) \| 0<br>count = 10 |
| | $N_{cpu} = 3$ | $C_{DE} = 0.1$ | $E_{ave} = 5.4191 \cdot 10^{-3}$ | |
| | $t_{max} = 500$ | $N_{eval} = 60120$ | $E_{max} = 6.5717 \cdot 10^{-3}$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 1.393s$ | $E_{dev} = \mathbf{6.259 \cdot 10^{-4}}$ | |
| CTP1 | $N_{sol} = 120$ | $N_f = 2$ | $E_{min} = 1.0582 \cdot 10^{-3}$ | [-0.05,-0.03) \| 0<br>[-0.03,-0.01) \| 0<br>[-0.01,0.01) \| 10 ##########<br>[0.01,0.03) \| 0<br>[0.03,0.05) \| 0<br>count = 10 |
| | $N_{cpu} = 3$ | $C_{DE} = 0.1$ | $E_{ave} = 1.4322 \cdot 10^{-3}$ | |
| | $t_{max} = 500$ | $N_{eval} = 60120$ | $E_{max} = 1.6912 \cdot 10^{-3}$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 2.723s$ | $E_{dev} = \mathbf{2.243 \cdot 10^{-4}}$ | |
| CTP2 | $N_{sol} = 120$ | $N_f = 2$ | $E_{min} = 1.1665 \cdot 10^{-3}$ | [-0.05,-0.03) \| 0<br>[-0.03,-0.01) \| 0<br>[-0.01,0.01) \| 10 ##########<br>[0.01,0.03) \| 0<br>[0.03,0.05) \| 0<br>count = 10 |
| | $N_{cpu} = 3$ | $C_{DE} = 0.1$ | $E_{ave} = 2.0726 \cdot 10^{-3}$ | |
| | $t_{max} = 500$ | $N_{eval} = 60120$ | $E_{max} = 2.7553 \cdot 10^{-3}$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 2.613s$ | $E_{dev} = \mathbf{5.327 \cdot 10^{-4}}$ | |
| CTP3 | $N_{sol} = 120$ | $N_f = 2$ | $E_{min} = 2.6576 \cdot 10^{-3}$ | [-0.05,-0.02) \| 0<br>[-0.02,-0.00) \| 0<br>[-0.00,0.02) \| 10 ##########<br>[0.02,0.04) \| 0<br>[0.04,0.06) \| 0<br>count = 10 |
| | $N_{cpu} = 3$ | $C_{DE} = 0.1$ | $E_{ave} = 6.3243 \cdot 10^{-3}$ | |
| | $t_{max} = 500$ | $N_{eval} = 60120$ | $E_{max} = 1.3662 \cdot 10^{-2}$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 2.596s$ | $E_{dev} = \mathbf{3.055 \cdot 10^{-3}}$ | |
| CTP4 | $N_{sol} = 120$ | $N_f = 2$ | $E_{min} = 1.6622 \cdot 10^{-2}$ | [-0.03,-0.01) \| 0<br>[-0.01,0.02) \| 0<br>[0.02,0.04) \| 10 ##########<br>[0.04,0.06) \| 0<br>[0.06,0.09) \| 0<br>count = 10 |
| | $N_{cpu} = 3$ | $C_{DE} = 0.1$ | $E_{ave} = 2.9671 \cdot 10^{-2}$ | |
| | $t_{max} = 500$ | $N_{eval} = 60120$ | $E_{max} = 3.8986 \cdot 10^{-2}$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 2.245s$ | $E_{dev} = \mathbf{7.755 \cdot 10^{-3}}$ | |
| CTP5 | $N_{sol} = 120$ | $N_f = 2$ | $E_{min} = 3.1174 \cdot 10^{-3}$ | [-0.05,-0.03) \| 0<br>[-0.03,-0.00) \| 0<br>[-0.00,0.02) \| 10 ##########<br>[0.02,0.04) \| 0<br>[0.04,0.06) \| 0<br>count = 10 |
| | $N_{cpu} = 3$ | $C_{DE} = 0.1$ | $E_{ave} = 5.6907 \cdot 10^{-3}$ | |
| | $t_{max} = 500$ | $N_{eval} = 60120$ | $E_{max} = 7.5488 \cdot 10^{-3}$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 2.826s$ | $E_{dev} = \mathbf{1.625 \cdot 10^{-3}}$ | |
| CTP6 | $N_{sol} = 120$ | $N_f = 2$ | $E_{min} = 8.0480 \cdot 10^{-2}$ | [0.03,0.06) \| 0<br>[0.06,0.08) \| 1 ##<br>[0.08,0.11) \| 8 ##########<br>[0.11,0.13) \| 1 ##<br>[0.13,0.16) \| 0<br>count = 10 |
| | $N_{cpu} = 3$ | $C_{DE} = 0.1$ | $E_{ave} = 9.5980 \cdot 10^{-2}$ | |
| | $t_{max} = 500$ | $N_{eval} = 60120$ | $E_{max} = 1.1097 \cdot 10^{-1}$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 2.552s$ | $E_{dev} = \mathbf{9.168 \cdot 10^{-3}}$ | |
| CTP7 | $N_{sol} = 120$ | $N_f = 2$ | $E_{min} = 4.1257 \cdot 10^{-3}$ | [-0.05,-0.03) \| 0<br>[-0.03,-0.00) \| 0<br>[-0.00,0.02) \| 10 ##########<br>[0.02,0.04) \| 0<br>[0.04,0.06) \| 0<br>count = 10 |
| | $N_{cpu} = 3$ | $C_{DE} = 0.1$ | $E_{ave} = 5.5464 \cdot 10^{-3}$ | |
| | $t_{max} = 500$ | $N_{eval} = 60120$ | $E_{max} = 8.2630 \cdot 10^{-3}$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 2.583s$ | $E_{dev} = \mathbf{1.404 \cdot 10^{-3}}$ | |
| CTP8 | $N_{sol} = 120$ | $N_f = 2$ | $E_{min} = 5.8808 \cdot 10^{-2}$ | [0.01,0.04) \| 0<br>[0.04,0.07) \| 2 #####<br>[0.07,0.10) \| 6 ##########<br>[0.10,0.12) \| 2 #####<br>[0.12,0.15) \| 0<br>count = 10 |
| | $N_{cpu} = 3$ | $C_{DE} = 0.1$ | $E_{ave} = 7.6783 \cdot 10^{-2}$ | |
| | $t_{max} = 500$ | $N_{eval} = 60120$ | $E_{max} = 1.0417 \cdot 10^{-1}$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 2.851s$ | $E_{dev} = \mathbf{1.420 \cdot 10^{-2}}$ | |

*Problem DTLZ2c* is the same as DTLZ2 with the addition of one inequality constraint function. This function ($g_0(\boldsymbol{x})$) defines a cone in the $f_i$ space that is aligned with its diagonal. The constraint is defined by

$$g_0 = \tan\alpha - \frac{\sqrt{(f_0 - f_1)^2 + (f_1 - f_2)^2 + (f_2 - f_0)^2}}{f_0 + f_1 + f_2} \tag{78}$$

where $\alpha$ equals half the cone's opening angle. Here, $\alpha = 15$ has been selected. In this way, the feasible solutions must lie on a circular patch of the optimal front. The results are illustrated in Figs. 9a and 9b.

*Problems SUQi*: A superquadric version of DTLZ2 is now introduced. The expressions in 72 are used, except that the sine and cosine functions are replaced by the following functions

$$\text{sinX}(w; m) = \text{sign}(\sin w)\,|\sin w|^m \tag{79}$$

and

$$\text{cosX}(w; m) = \text{sign}(\cos w)\,|\cos w|^m \tag{80}$$

respectively. In this way, the *SUQi* problems are defined by

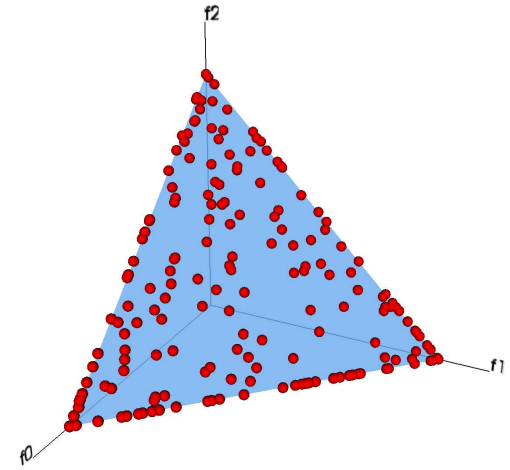
$$\begin{aligned} c &= \textstyle\sum_{i=2}^{11}(x_i - 0.5)^2 \\ f_0 &= (1+c)\ \text{cosX}(0.5x_0\pi; 2/a)\ \text{cosX}(0.5x_1\pi; 2/a) \\ f_1 &= (1+c)\ \text{cosX}(0.5x_0\pi; 2/b)\ \text{sinX}(0.5x_1\pi; 2/b) \\ f_2 &= (1+c)\ \text{sinX}(0.5x_0\pi; 2/c) \end{aligned} \tag{81}$$

where $a$, $b$ and $c$ are the coefficients of the superquadric. The error related to the Pareto-optimal front is then

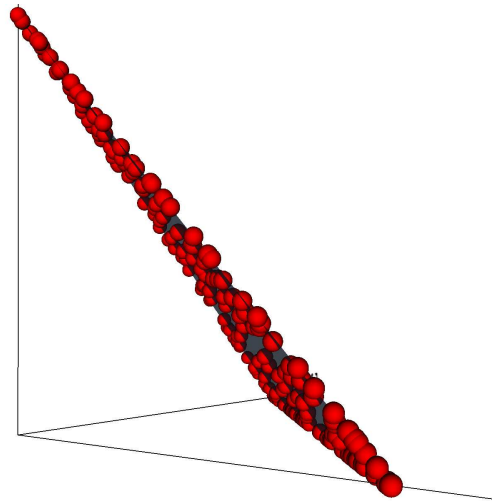
$$\psi(f_0, f_1, f_2) = |f_0|^a + |f_1|^b + |f_2|^c - 1 \tag{82}$$

The above expressions are quite versatile because they can produce concave, convex and many kinds of twisted surfaces in the space of objective values. Views of the Pareto-optimal fronts and numerical results are shown in Figs. 9c and 9d for SUQ1 and in Figs. 9e and 9f for SUQ2 where it can be seen that the points (small spheres) approach the optimal front.

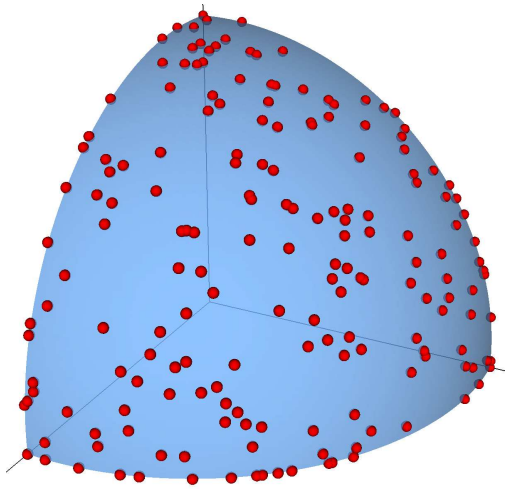
A *starplot* is employed to assist on verifying the ability of the code to cover all objective functions. For each feasible trial solution with $\phi = 0$ (best Pareto-optimal front), a line plot is



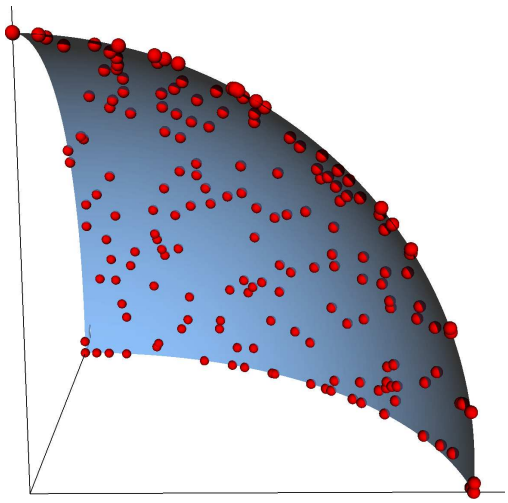

(a) DTLZ1 (view 1)



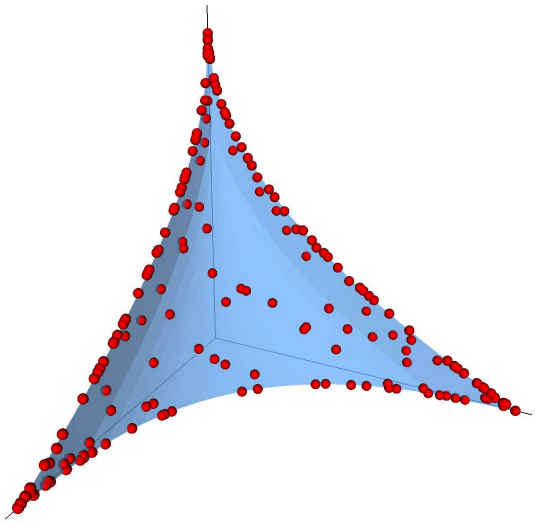
(b) DTLZ1 (view 2)



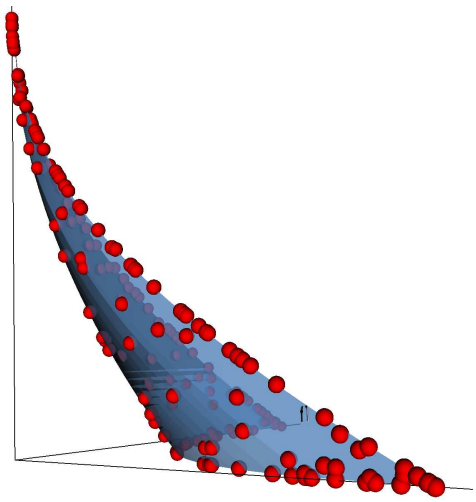
(c) DTLZ2 (view 1)



(d) DTLZ2 (view 2)

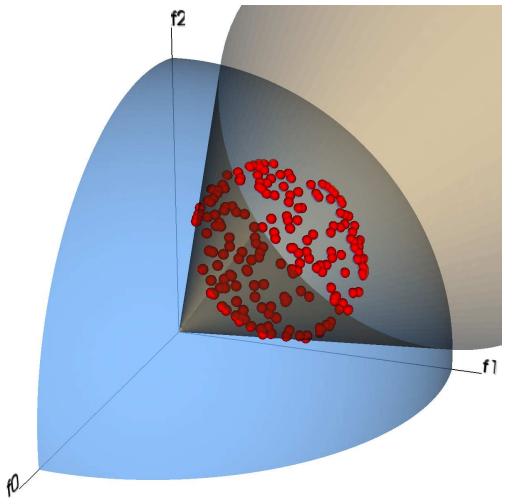


(e) DTLZ2x (view 1)

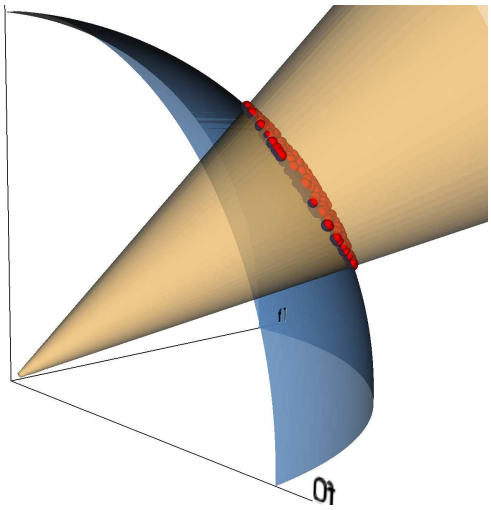


(f) DTLZ2x (view 2)

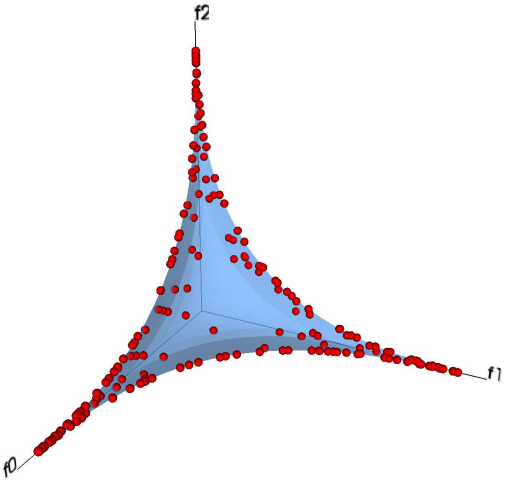
Figure 8: Three-objective problems.



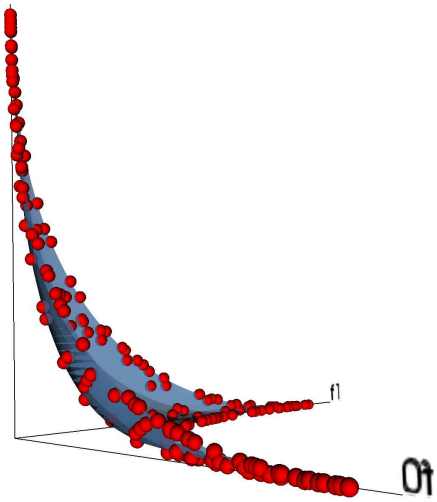

(a) DTLZ2c (view 1)



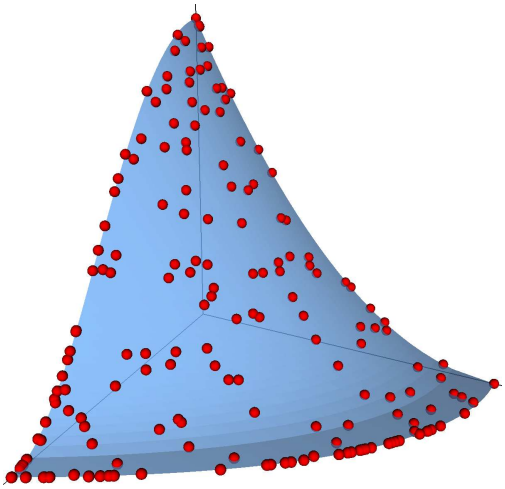

(b) DTLZ2c (view 2)



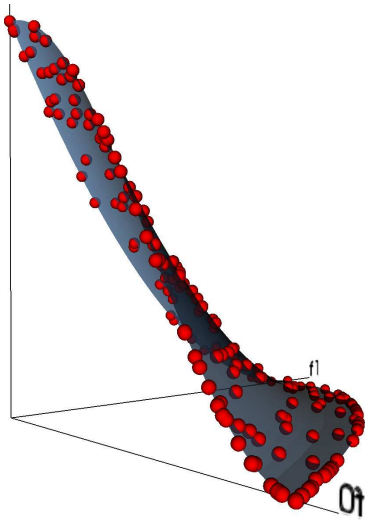

(c) SUQ1 (view 1)




(d) SUQ1 (view 2)



(e) SUQ2 (view 1)




(f) SUQ2 (view 2)

Figure 9: Three-objective problems (contd.).

drawn by connecting the $f_i$ values. The axes for the $f_i$ values are drawn as in a star with equal angles. The angle between axes is hence $\Delta\theta = 2\pi/N_f$. Because the $f_i$ points are connected, an area is created in the diagram. For reference, four circles are drawn; each corresponding to a normalised objective value of 0.25, 0.5, 0.75 and 1.0. The normalisation factor is equal to 1.0 for all tests, except for DTLZ1 where it is equal to 0.5. A good coverage of objective functions is hence indicated by lines reaching the outer circles with high frequency.

The starplots for all DTLZi and SUQi tests are shown in Figs. 10 and 11 where it can be observed that the three objective values are well represented in all tests. Symmetry is observed in DTLZ1-4, DTLZ2c and SUQ1. The superquadric SUQ2 does not have any symmetry in particular. It is interesting to observe the difference between DTLZ2x (convex) and SUQ1 (superquadric) with respect to symmetry. In DTLZ2c, the region of values is restricted because of the constraint. In this case, the combination of $f_i$ values must be over 0.3 and smaller than a value close to 0.75.

To assess the accuracy and repeatability of results, 1000 samples are run for each problem and the statistics with respect to the root mean square error are computed. The results are collected in Table 6. It can be observed that the errors and standard deviations reached machine precision with the exception of DTLZ3 which is a little harder due to the local optima. With more iterations, the error is reduced. The constraint in DTLZ2c does not cause problems to the solver; remember that the algorithm prioritises satisfaction of constraints first hence trial solutions are 'pulled' into the conical space.

## 10. Test cases: many objective

A final set of tests is carried out prior to the applications. Now, problem DTLZ2 is re-analysed with more than 3 objectives. The following numbers of are selected: $N_f \in \{5, 7, 10, 15, 20\}$. The number of variables is also increased according to

$$N_x = N_f + 10 \tag{83}$$

Thus, $N_x \in \{15, 17, 20, 35\}$. The number of trial solutions is $N_{sol} = 300$ and 6 groups are employed. It is clear that the problem with more objectives and variables are more difficult to be solved. The problems are identified as *DTLZ2mI* where *I* means the number of objectives.

The starplots corresponding to the DTLZ2mI tests are illustrated in Fig. 12 where it can be observed that all objective values are found. As an exception, tests with 13 or more objective values have less values reaching 1. Note that the number of trial solutions was kept constant and equal to 300. With a few more iterations ($t_{max}$) and number of variables, the response would be improved. This has not been done here for the sake of showing how the difficulty increases with the number of objective functions.

With regards to the statistics of errors, Table 7 presents the results for 1000 samples. As shown in this table, it is remarkable the ability of the proposed evolutionary algorithm to obtain very small errors every time it is run. We observe in particular that the error increases in a nonlinear trend with the number of objective functions; this is plotted in Fig. 13. It is actually interesting to see that the error rate decreases a little considering that the number of trial solutions and iterations was kept constant. This means that the performance of the code is reasonable for these tests.

## 11. Application: truss shape and topology optimisation with two objectives

The topology optimisation problem is an important one in structural engineering design. This problem is even more difficult to be solved when the cross sectional areas of structural members (rods) have to be found at the same time as the connections between rods. The problem is known as shape and topology optimisation of trusses made up of slender rods (bars). A two objective version of this problem is considered here. The two objectives are: (1) minimum deflection when the structure is under loads; and (2) the overall minimum weight. Better satisfying one goal induces a failure in satisfying the other one. Hence a Pareto optimal front is to be determined. The structure has to be stable as well; therefore limits on maximum stresses must be followed hence defining constraints.

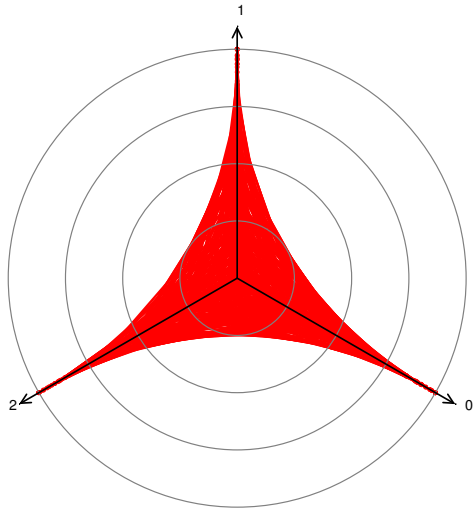
To solve this problem, a combination of real numbers and integers is implemented. The equilibrium of the structure is solved by the finite element method (FEM) and a method to handle failures of the FEM calculations is developed. The speedup that can be achieved with the resulting parallel code is also demonstrated in this section.

The example studied in [56, 3, 57–59] and illustrated in Fig. 14 is considered. The Young's modulus of all bars is $E = 10^4$ [units of pressure] and the density is $\rho = 0.1$ [units of density]. In this figure, the structure is known as *ground* structure because it serves as a basis for the generation of other combinations. It is important to note that a minimum number of joints (nodes) and bars (elements) must be present at all times in order to allow the structure to carry the prescribed load. In this particular problem, the indicated fixed nodes and the nodes where loads are applied must be always present. Therefore, before solving the mechanical problem, the connectivity has to be checked.

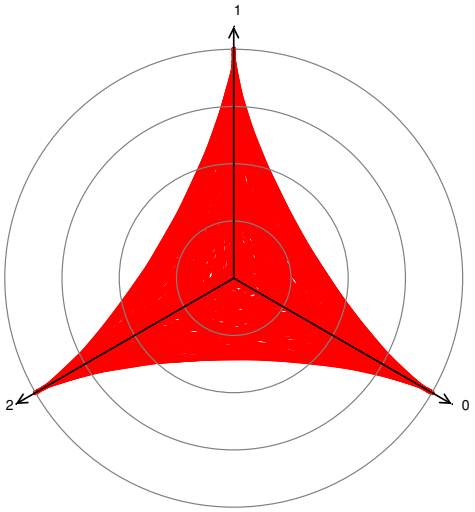
Even if all required nodes are present, the structure may become a mechanism (or mobile). In this situation, the FEM solver will fail due to a singular global stiffness matrix. To minimise the number of unsuccessful calls to the FEM solver, a *mobility* number $M$ can be estimated. In this work, the Chebychev-Grübler-Kutzbach criterion is employed to this task [60, 61]. For the truss system, $M$ is defined as

$$M = 2\,n - m - \chi \tag{84}$$

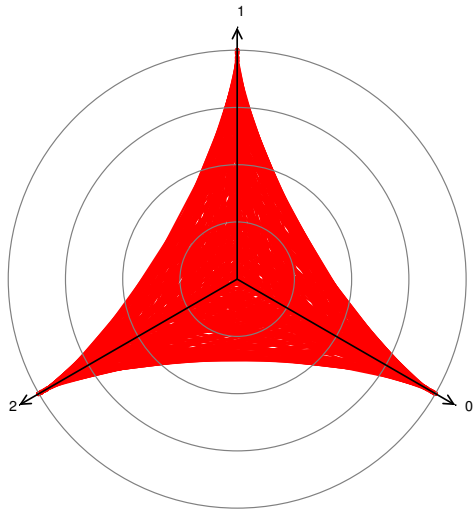
where $n$ is the number of nodes, $m$ is the number of bars and $\chi$ is the number of fixed degrees of freedom. For instance, in the structure of Fig. 14, $n = 6$, $m = 10$ and $\chi = 4$ because $\{u_x, u_y\}$ at $(0.0, 0.0)$ and $(0.0, 360.0)$ are fixed. If $M$ is positive,



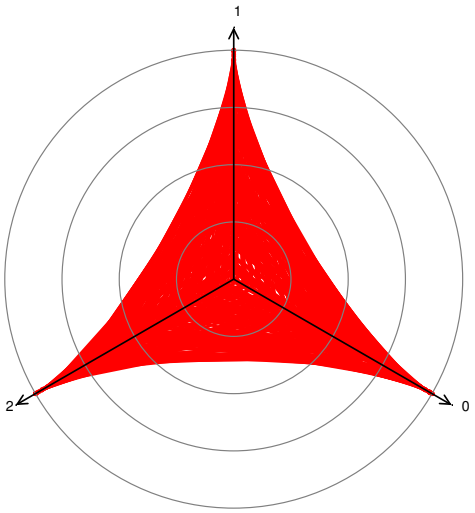

(a) DTLZ1



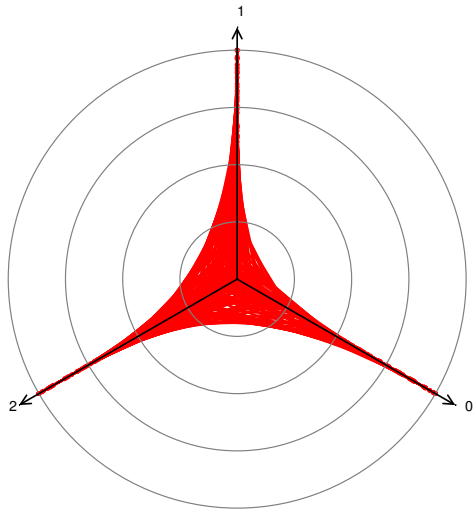

(b) DTLZ2



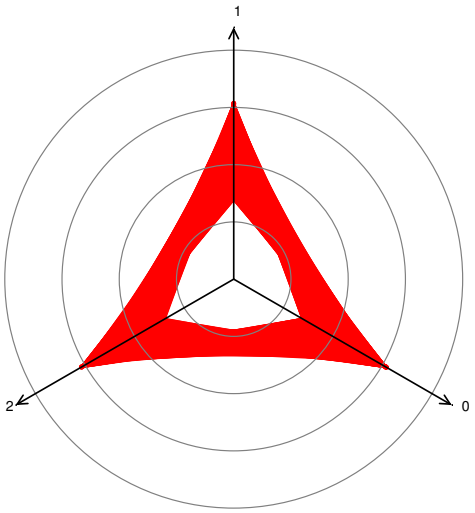

(c) DTLZ3




(d) DTLZ4

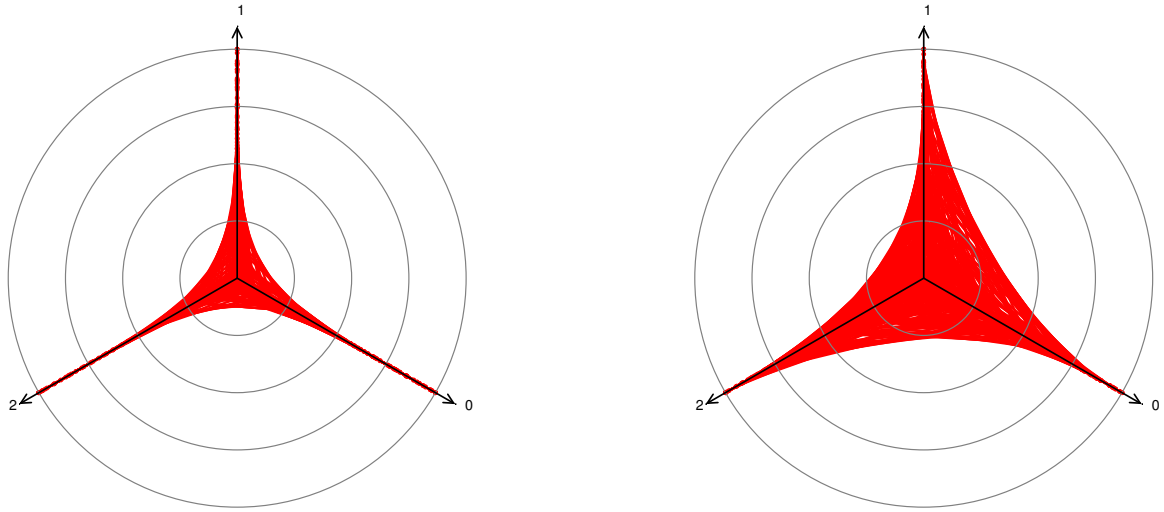



(e) DTLZ2x




(f) DTLZ2c

Figure 10: Three-objective problems. Starplots.



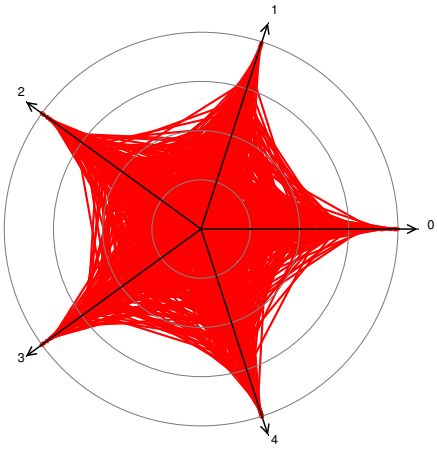

(a) SUQ1 (b) SUQ2

Figure 11: Three-objective problems. Starplots (contd.).

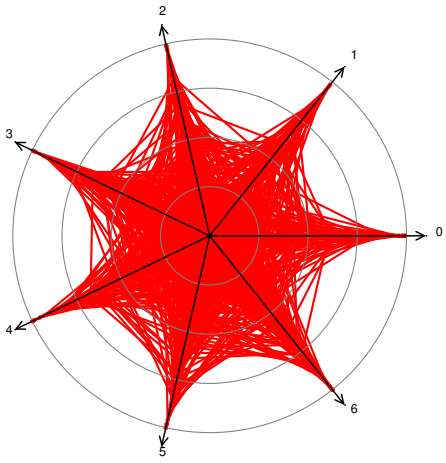
Table 6: Unconstrained and constrained three objective problems.

| P | settings | settings/info | error | histogram ($N_{samples} = 1000$) |
|---|---|---|---|---|
| DTLZ1 | $N_{sol} = 200$ | $N_f = 3$ | $E_{min} = 0.0000$ | [-0.05,-0.03) \| 0<br>[-0.03,-0.01) \| 0<br>[-0.01,0.01) \| 1000 ##############<br>[0.01,0.03) \| 0<br>[0.03,0.05) \| 0<br>count = 1000 |
| | $N_{cpu} = 5$ | $C_{DE} = 0.01$ | $E_{ave} = 1.8491 \cdot 10^{-15}$ | |
| | $t_{max} = 500$ | $N_{eval} = 100200$ | $E_{max} = 4.5561 \cdot 10^{-14}$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 5m57.776s$ | $E_{dev} = \mathbf{2.228 \cdot 10^{-15}}$ | |
| DTLZ2 | $N_{sol} = 200$ | $N_f = 3$ | $E_{min} = 3.6770 \cdot 10^{-16}$ | [-0.05,-0.03) \| 0<br>[-0.03,-0.01) \| 0<br>[-0.01,0.01) \| 1000 ##############<br>[0.01,0.03) \| 0<br>[0.03,0.05) \| 0<br>count = 1000 |
| | $N_{cpu} = 5$ | $C_{DE} = 0.01$ | $E_{ave} = 5.4261 \cdot 10^{-16}$ | |
| | $t_{max} = 500$ | $N_{eval} = 100200$ | $E_{max} = 2.6167 \cdot 10^{-14}$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 7m50.912s$ | $E_{dev} = \mathbf{1.017 \cdot 10^{-15}}$ | |
| DTLZ3 | $N_{sol} = 200$ | $N_f = 3$ | $E_{min} = 6.5402 \cdot 10^{-6}$ | [-0.05,-0.01) \| 0<br>[-0.01,0.03) \| 998 ##############<br>[0.03,0.06) \| 1 #<br>[0.06,0.10) \| 1 #<br>[0.10,0.14) \| 0<br>count = 1000 |
| | $N_{cpu} = 5$ | $C_{DE} = 0.01$ | $E_{ave} = 8.8755 \cdot 10^{-4}$ | |
| | $t_{max} = 500$ | $N_{eval} = 100200$ | $E_{max} = 8.9955 \cdot 10^{-2}$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 8m0.856s$ | $E_{dev} = \mathbf{3.326 \cdot 10^{-3}}$ | |
| DTLZ4 | $N_{sol} = 200$ | $N_f = 3$ | $E_{min} = 3.7136 \cdot 10^{-16}$ | [-0.05,-0.03) \| 0<br>[-0.03,-0.01) \| 0<br>[-0.01,0.01) \| 1000 ##############<br>[0.01,0.03) \| 0<br>[0.03,0.05) \| 0<br>count = 1000 |
| | $N_{cpu} = 5$ | $C_{DE} = 0.01$ | $E_{ave} = 7.5152 \cdot 10^{-16}$ | |
| | $t_{max} = 500$ | $N_{eval} = 100200$ | $E_{max} = 2.3689 \cdot 10^{-13}$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 7m41.966s$ | $E_{dev} = \mathbf{7.522 \cdot 10^{-15}}$ | |
| DTLZ2x | $N_{sol} = 200$ | $N_f = 3$ | $E_{min} = 3.6292 \cdot 10^{-16}$ | [-0.05,-0.03) \| 0<br>[-0.03,-0.01) \| 0<br>[-0.01,0.01) \| 1000 ##############<br>[0.01,0.03) \| 0<br>[0.03,0.05) \| 0<br>count = 1000 |
| | $N_{cpu} = 5$ | $C_{DE} = 0.01$ | $E_{ave} = 5.1407 \cdot 10^{-16}$ | |
| | $t_{max} = 500$ | $N_{eval} = 100200$ | $E_{max} = 1.2601 \cdot 10^{-14}$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 7m37.829s$ | $E_{dev} = \mathbf{5.393 \cdot 10^{-16}}$ | |
| DTLZ2c | $N_{sol} = 200$ | $N_f = 3$ | $E_{min} = 4.3752 \cdot 10^{-16}$ | [-0.05,-0.03) \| 0<br>[-0.03,-0.01) \| 0<br>[-0.01,0.01) \| 1000 ##############<br>[0.01,0.03) \| 0<br>[0.03,0.05) \| 0<br>count = 1000 |
| | $N_{cpu} = 5$ | $C_{DE} = 0.01$ | $E_{ave} = 1.1407 \cdot 10^{-15}$ | |
| | $t_{max} = 500$ | $N_{eval} = 100200$ | $E_{max} = 1.2321 \cdot 10^{-13}$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 7m50.907s$ | $E_{dev} = \mathbf{5.053 \cdot 10^{-15}}$ | |
| SUQ1 | $N_{sol} = 200$ | $N_f = 3$ | $E_{min} = 1.2918 \cdot 10^{-16}$ | [-0.05,-0.03) \| 0<br>[-0.03,-0.01) \| 0<br>[-0.01,0.01) \| 1000 ##############<br>[0.01,0.03) \| 0<br>[0.03,0.05) \| 0<br>count = 1000 |
| | $N_{cpu} = 5$ | $C_{DE} = 0.01$ | $E_{ave} = 1.7526 \cdot 10^{-16}$ | |
| | $t_{max} = 500$ | $N_{eval} = 100200$ | $E_{max} = 6.8901 \cdot 10^{-15}$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 7m46.474s$ | $E_{dev} = \mathbf{2.539 \cdot 10^{-16}}$ | |
| SUQ2 | $N_{sol} = 200$ | $N_f = 3$ | $E_{min} = 2.0724 \cdot 10^{-16}$ | [-0.05,-0.03) \| 0<br>[-0.03,-0.01) \| 0<br>[-0.01,0.01) \| 1000 ##############<br>[0.01,0.03) \| 0<br>[0.03,0.05) \| 0<br>count = 1000 |
| | $N_{cpu} = 5$ | $C_{DE} = 0.01$ | $E_{ave} = 2.9736 \cdot 10^{-16}$ | |
| | $t_{max} = 500$ | $N_{eval} = 100200$ | $E_{max} = 6.7016 \cdot 10^{-15}$ | |
| | $\Delta t_{exc} = 50$ | $T_{sys} = 7m35.755s$ | $E_{dev} = \mathbf{3.274 \cdot 10^{-16}}$ | |

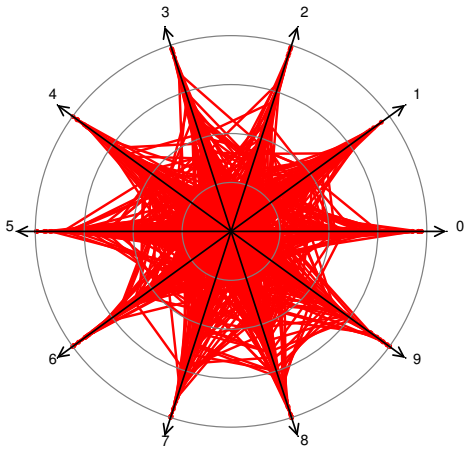
the above expression indicates that the structure is a mechanism/mobile. Nonetheless, the expression may fail at times for instance when there are redundant bars at some locations and some local mechanisms arise at other locations. Therefore, the



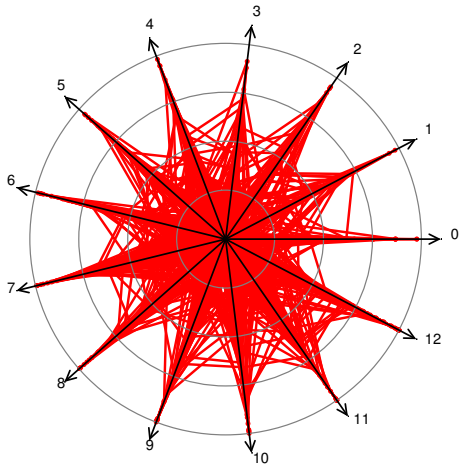

(a) DTLZ2m5



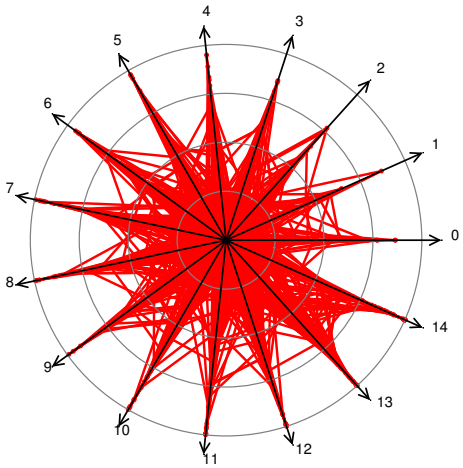

(b) DTLZ2m7



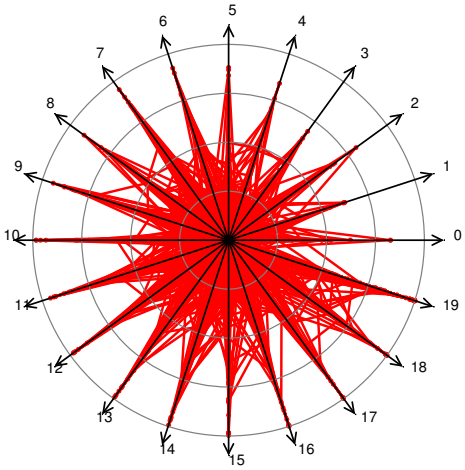

(c) DTLZ2m10




(d) DTLZ2m13




(e) DTLZ2m15




(f) DTLZ2m20

Figure 12: Many-objective problems. Starplots.

Table 7: Unconstrained many objective problems

| P | settings | settings/info | error | histogram ($N_{samples}$ = 1000) |
|---|---|---|---|---|
| DTLZ2m5 | $N_{sol}$ = 300 | $N_f$ = 5 | $E_{min} = 1.5523 \cdot 10^{-13}$ | [-0.05,-0.03) \| 0<br>[-0.03,-0.01) \| 0<br>[-0.01,0.01) \| 1000 ##############<br>[0.01,0.03) \| 0<br>[0.03,0.05) \| 0<br>count = 1000 |
| | $N_{cpu}$ = 6 | $C_{DE}$ = 0.01 | $E_{ave} = 3.5998 \cdot 10^{-12}$ | |
| | $t_{max}$ = 500 | $N_{eval}$ = 150300 | $E_{max} = 9.7436 \cdot 10^{-10}$ | |
| | $\Delta t_{exc}$ = 50 | $T_{sys}$ = 20$m$25.3$s$ | $E_{dev} = \mathbf{3.288 \cdot 10^{-11}}$ | |
| DTLZ2m7 | $N_{sol}$ = 300 | $N_f$ = 7 | $E_{min} = 2.2129 \cdot 10^{-11}$ | [-0.05,-0.03) \| 0<br>[-0.03,-0.01) \| 0<br>[-0.01,0.01) \| 1000 ##############<br>[0.01,0.03) \| 0<br>[0.03,0.05) \| 0<br>count = 1000 |
| | $N_{cpu}$ = 6 | $C_{DE}$ = 0.01 | $E_{ave} = 2.9629 \cdot 10^{-10}$ | |
| | $t_{max}$ = 500 | $N_{eval}$ = 150300 | $E_{max} = 1.6546 \cdot 10^{-8}$ | |
| | $\Delta t_{exc}$ = 50 | $T_{sys}$ = 22$m$42.9$s$ | $E_{dev} = \mathbf{1.142 \cdot 10^{-9}}$ | |
| DTLZ2m10 | $N_{sol}$ = 300 | $N_f$ = 10 | $E_{min} = 4.5378 \cdot 10^{-9}$ | [-0.05,-0.03) \| 0<br>[-0.03,-0.01) \| 0<br>[-0.01,0.01) \| 1000 ##############<br>[0.01,0.03) \| 0<br>[0.03,0.05) \| 0<br>count = 1000 |
| | $N_{cpu}$ = 6 | $C_{DE}$ = 0.01 | $E_{ave} = 6.0300 \cdot 10^{-8}$ | |
| | $t_{max}$ = 500 | $N_{eval}$ = 150300 | $E_{max} = 3.0293 \cdot 10^{-6}$ | |
| | $\Delta t_{exc}$ = 50 | $T_{sys}$ = 26$m$10.4$s$ | $E_{dev} = \mathbf{2.024 \cdot 10^{-7}}$ | |
| DTLZ2m13 | $N_{sol}$ = 300 | $N_f$ = 13 | $E_{min} = 2.6536 \cdot 10^{-7}$ | [-0.05,-0.03) \| 0<br>[-0.03,-0.01) \| 0<br>[-0.01,0.01) \| 1000 ##############<br>[0.01,0.03) \| 0<br>[0.03,0.05) \| 0<br>count = 1000 |
| | $N_{cpu}$ = 6 | $C_{DE}$ = 0.01 | $E_{ave} = 3.3686 \cdot 10^{-6}$ | |
| | $t_{max}$ = 500 | $N_{eval}$ = 150300 | $E_{max} = 4.3098 \cdot 10^{-4}$ | |
| | $\Delta t_{exc}$ = 50 | $T_{sys}$ = 29$m$33.6$s$ | $E_{dev} = \mathbf{1.967 \cdot 10^{-5}}$ | |
| DTLZ2m15 | $N_{sol}$ = 300 | $N_f$ = 15 | $E_{min} = 1.9740 \cdot 10^{-6}$ | [-0.05,-0.03) \| 0<br>[-0.03,-0.01) \| 0<br>[-0.01,0.01) \| 1000 ##############<br>[0.01,0.03) \| 0<br>[0.03,0.05) \| 0<br>count = 1000 |
| | $N_{cpu}$ = 6 | $C_{DE}$ = 0.01 | $E_{ave} = 2.5914 \cdot 10^{-5}$ | |
| | $t_{max}$ = 500 | $N_{eval}$ = 150300 | $E_{max} = 1.8183 \cdot 10^{-3}$ | |
| | $\Delta t_{exc}$ = 50 | $T_{sys}$ = 31$m$35.1$s$ | $E_{dev} = \mathbf{1.035 \cdot 10^{-4}}$ | |
| DTLZ2m20 | $N_{sol}$ = 300 | $N_f$ = 20 | $E_{min} = 1.5243 \cdot 10^{-4}$ | [-0.05,-0.02) \| 0<br>[-0.02,0.00) \| 791 #############<br>[0.00,0.03) \| 207 ####<br>[0.03,0.05) \| 2 #<br>[0.05,0.08) \| 0<br>count = 1000 |
| | $N_{cpu}$ = 6 | $C_{DE}$ = 0.01 | $E_{ave} = 1.1966 \cdot 10^{-3}$ | |
| | $t_{max}$ = 500 | $N_{eval}$ = 150300 | $E_{max} = 2.7725 \cdot 10^{-2}$ | |
| | $\Delta t_{exc}$ = 50 | $T_{sys}$ = 37$m$8.4$s$ | $E_{dev} = \mathbf{2.453 \cdot 10^{-3}}$ | |

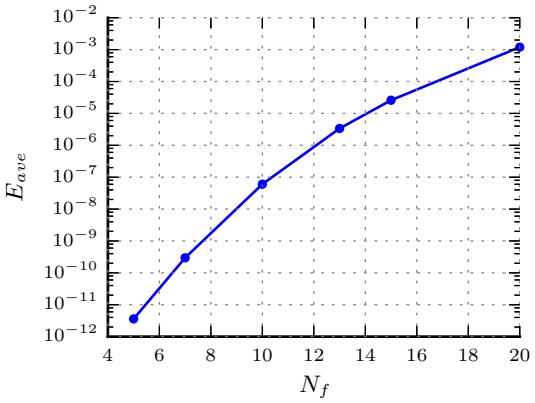



Figure 13: DTLZ2mI: Increase of error with number of objectives.

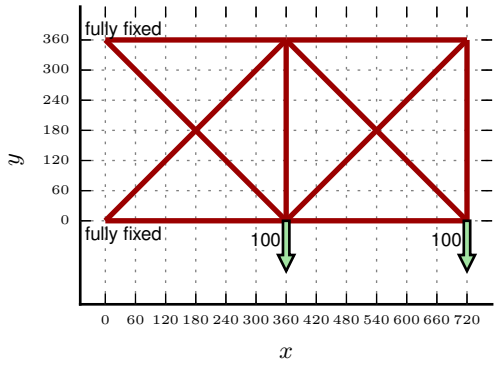



Figure 14: "Ground" mesh with all 10 rods active.

above condition is not sufficient and yet a singular global stiffness matrix (or the failure of the FEM) must be properly handled.

The problem representation is as follows. Integers are used for the topology description (connection of bars) and real numbers for the cross sectional areas. In this case, integers simply serve as Boolean variables where 1 means a connection is active and 0 otherwise. There are 10 bars (10 areas); thus the number of real numbers $\boldsymbol{x}$ is $N_x = 10$ and the number of integers $\boldsymbol{\xi}$ is $N_\xi = 10$. The limits are

$$0.09 \leq x_i \leq 35 \quad i \in [0, 9] \tag{85}$$

indicating that the minimum and maximum areas are 0.09 and 35 [units of area], respectively.

The objective values ($f_0$: total weight; $f_1$: deflection) are computed after running a finite element (FEM) simulation to calculate the maximum deflection $\delta = |u_y^{max}|$ at $x = 720$ and $y = 0$. This value is also limited by a maximum allowance of $\delta_{awd} = 5.6$ [units of displacement]. The maximum allowed tensile or compressive axial stress $\sigma_{max}$ (from FEM) at any bar is $\sigma_{awd} = 35$ [units of pressure]. The bar lengths are $L_i$.

Considering the general optimisation problem Eq. (1), the

required expressions are (with $f_i = f_i(\underset{\cdot}{x}, \underset{\cdot}{\xi})$ and $u_i = u_i(\underset{\cdot}{x}, \underset{\cdot}{\xi})$)

$$
\begin{aligned}
f_0 &= \sum_{i=0}^{N_{active}} \rho\, x_i\, L_i \\
f_1 &= \delta \\
u_0 &= \langle M \rangle \\
u_1 &= 1 \text{ if FEM failed; } 0 \text{ otherwise} \\
u_2 &= \langle \delta - \delta_{awd} \rangle \\
u_3 &= \langle |\sigma_{max}| - \sigma_{awd} \rangle
\end{aligned} \tag{86}
$$

There are four constraints (out-of-range/OOR) functions $u_i$. In the above equation, $\langle x \rangle$ is the ramp function defined by

$$
\langle x \rangle = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise} \end{cases} \tag{87}
$$

The use of the OOR functions is quite convenient because, since the comparison between trial solutions first considers the number of constraint violations $N_{viol}$, the OOR functions can be set in a way such that the worst case gets the highest $N_{viol}$ value by assigning a greater than zero $u_i$ values (e.g. 1.0). Note also that the OORs are compared using the `ParetoComparison` which effectively ranks trial solutions even if the FEM solver is not called at all. The proposed strategy is implemented as detailed in Algorithm 9.

In Algorithm 9, if not all vertices are present because connections cannot be made, the OOR $u_0$ gets $1 + 2\,n$ which is a number greater than the maximum mobility $M$ (when there are no bars and no fixities). In this situation, the other OORs all get 1.0. Therefore, the first exit condition in the algorithm will define a candidate that is worse than another one which could be at least used to compute its mobility number. Note that the third exit condition is closer to a situation needed to calculate the displacements and stresses; but still the FEM fails.

The solution is obtained with Goga where its good repeatability characteristics is verified again. The other configuration parameters are as follow. Simple crossover and mutation rules for integers are employed [1] with probability $P_c = 0.5$ and $P_m = 0.01$, respectively. The number of cuts during crossover is 1 and the number of bit changes during mutation is 1. $N_{sol} = 200$, $t_{max} = 500$ and $\Delta t_{exc} = 50$ are selected. The differential evolution coefficient value $C_{DE} = 0.8$ is selected. The number of groups $N_{cpu}$ is varied from 1 to 16 in order to study the speedup characteristic.

A typical Pareto-optimal front obtained with Goga is shown in Fig. 15 where a reference line from [3] is also drawn (shown in blue). Five combinations of weight-deflection values are indicated in the figure using labels 'A' to 'E' ranging from the lightest structure to the heaviest one. The resulting structure corresponding to each label is drawn in the same figure near the weight-deflection point (marked with a green star). Some examples of trusses are drawn near the weight-deflection point (marked with a green star). In each drawing, the thickness of the line is proportional to the optimal cross sectional area. We observe that the results match the reference ones reasonably well.

**Algorithm 9.** `RunFEA`$(\underset{\cdot}{x}, \underset{\cdot}{\xi})$ runs finite element analysis and returns OVA and OOR values.

**Input:** trial solution: Areas=$\underset{\cdot}{x}$, EnabledConnections=$\underset{\cdot}{\xi}$
**Output:** $f_i$ and $u_i$
compute $n$ = number of nodes
set connectivity based on $\underset{\cdot}{\xi}$
initialise FEA stage
check for required vertices
**if** *not all required vertices are present* **then**
  $u_0, u_1, u_2, u_3 \leftarrow 1 + 2\,n,\ 1,\ 1,\ 1$
  **return**
compute mobility $M$
**if** $M > 0$ **then**
  $u_0, u_1, u_2, u_3 \leftarrow M,\ 1,\ 1,\ 1$
  **return**
set elements' cross-sectional areas
compute total weight $W$
run FE analysis (FEA)
**if** *FEA failed* **then**
  $u_0, u_1, u_2, u_3 \leftarrow 0,\ 1,\ 1,\ 1$
  **return**
extract maximum deflection from FEA
extract maximum magnitude of stress from FEA
$f_0, f_1 \leftarrow W, \delta$
$u_0, u_1, u_2, u_3 \leftarrow 0,\ 0,\ \langle \delta - \delta_{awd} \rangle, \langle |\sigma_{max}| - \sigma_{awd} \rangle$

The computed areas corresponding to each selected point are collected in Table 8.

The efficiency of the parallel strategy is now assessed by running the topology optimisation problem sixteen times with different number of groups (CPUs). Simulations are carried out in a 16-core Intel(R) Xeon(R) CPU E5-2687W @ 3.10GHz (Debian-Sid/GNU/Linux). The speedup is then computed by means of

$$
S_{up}(N_{cpu}) = \frac{T^1_{sys}}{T^{N_{cpu}}_{sys}} \tag{88}
$$

where $T^1_{sys}$ is the computer time using one group (CPU) and $T^{N_{cpu}}_{sys}$ is the computer time using $N_{cpu}$ groups. The results are plotted Fig. 16 where we can observe a better than ideal speedup up to 14 CPUs. In the figure, the real computer time is also shown in using a gray line with values given at the right-hand-side scale; the maximum time is around 40 seconds, for instance.

The reason for the excellent speedup behaviour is in part due to a faster execution of the non-optimised implementation of the `Metrics` routine. This routine needs to perform several comparisons between trial solutions in order to find the closest neighbours in addition to determine the Pareto front indices. For example, by disabling the FEM computations, a slightly similar behaviour is observed where a very high speedup happens as well.

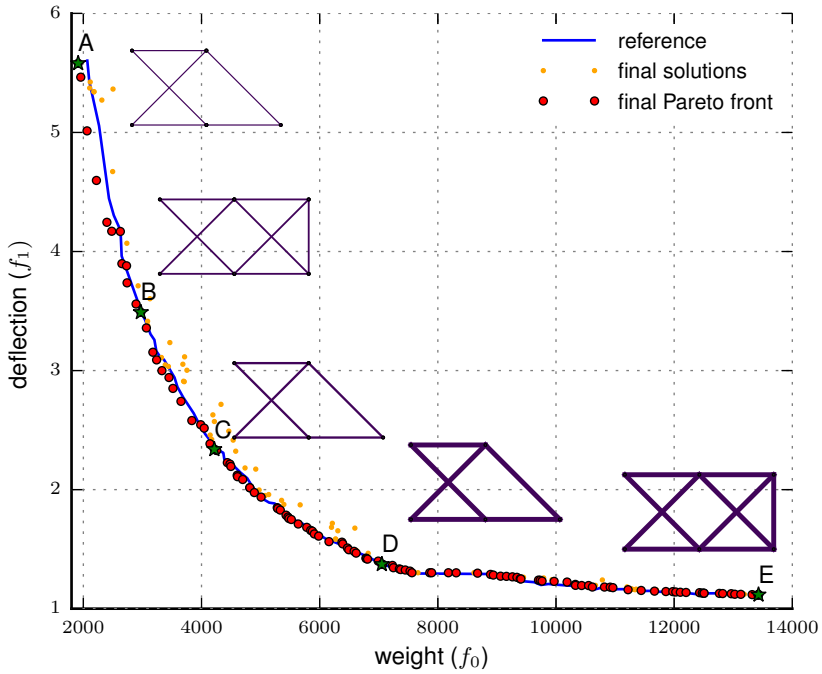



Figure 15: Shape and topology optimisation. Results.

Table 8: Shape and topology optimisation. Results.

| point | weight | deflection | $A_0$ | $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ | $A_7$ | $A_8$ | $A_9$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 1912.13 | 5.580622 | 8.827966 | 3.533588 | 12.495180 | 22.669679 | 8.884148 | 4.063610 | 12.333181 | 3.957120 | 2.173979 | 7.033611 |
| B | 2972.83 | 3.489310 | 15.541437 | 7.134360 | 14.981606 | 0.090000 | 14.618174 | 5.156832 | 0.090000 | 0.090000 | 0.090000 | 11.771825 |
| C | 4217.13 | 2.338615 | 22.326627 | 15.035148 | 25.818657 | 3.976012 | 18.164431 | 4.527009 | 2.816995 | 0.090000 | 0.090000 | 15.465500 |
| D | 7051.61 | 1.373101 | 35.000000 | 20.676599 | 35.000000 | 5.683623 | 35.000000 | 4.388616 | 4.996590 | 18.192946 | 28.873534 | 35.000000 |
| E | 13427.64 | 1.117026 | 35.000000 | 35.000000 | 35.000000 | 0.090000 | 35.000000 | 35.000000 | 35.000000 | 35.000000 | 35.000000 | 35.000000 |

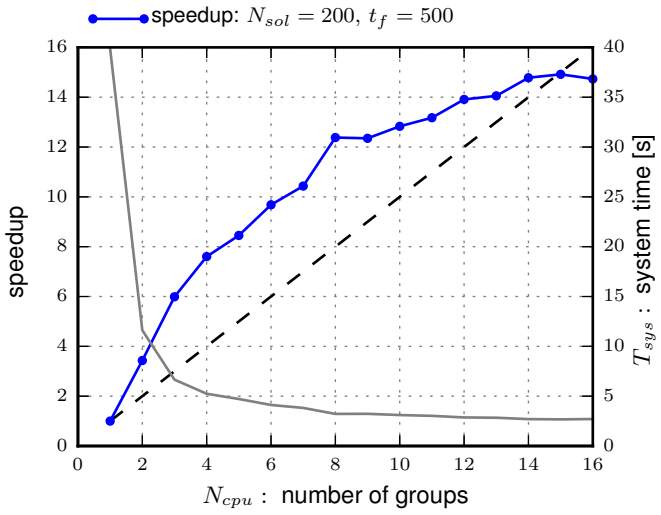



Figure 16: Shape and topology optimisation. Speedup.

## 12. Application: economic emission load dispatch

Economic emission load dispatch (or environmental/economic dispatch EED) is an interesting multi objective optimisation problem of great importance in power generation. The problem involves the design of optimal power flow observing the cost of operation and the emission of pollutants. These two objectives are contradictory and a number of constraints such as the maximum capacity of generators and the total power balance must be satisfied. The problem has been studied over several years; for example, some papers from 1987 to 2015 are: [62–78].

In the EED problem, the fuel cost $f_0$ of a thermal unit is approximated by the following quadratic function

$$f_0 = \sum_{i=0}^{N-1} \left( a_i + b_i P_i + c_i P_i^2 \right) \tag{89}$$

where $N$ is the number of generators and $x_i := P_i$ is the (unknown) active output power of generator $i$. Therein, $a_i$, $b_i$ and $c_i$ are fitting parameters. The corresponding emission $f_1$ is also modelled as a function of the power output by means of

$$f_1 = \sum_{i=0}^{N-1} \left[ \alpha_i + \beta_i P_i + \gamma_i P_i^2 + \zeta_i \exp\left(\lambda_i P_i\right) \right] \tag{90}$$

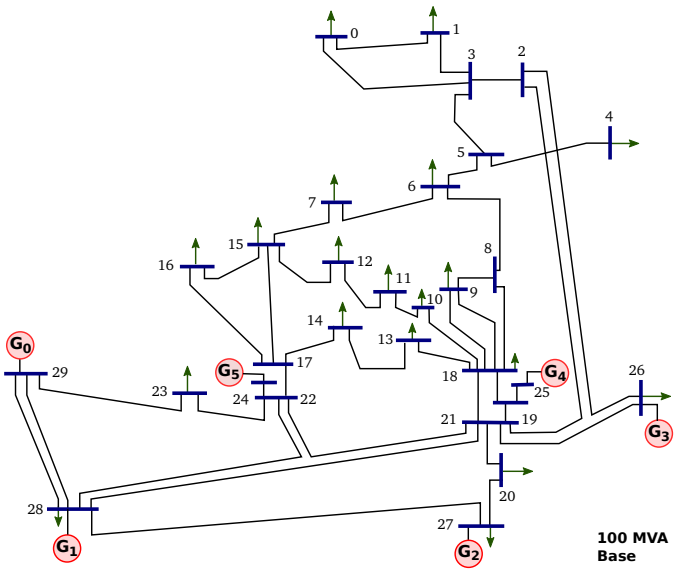



Figure 17: IEEE 30 bus test system.

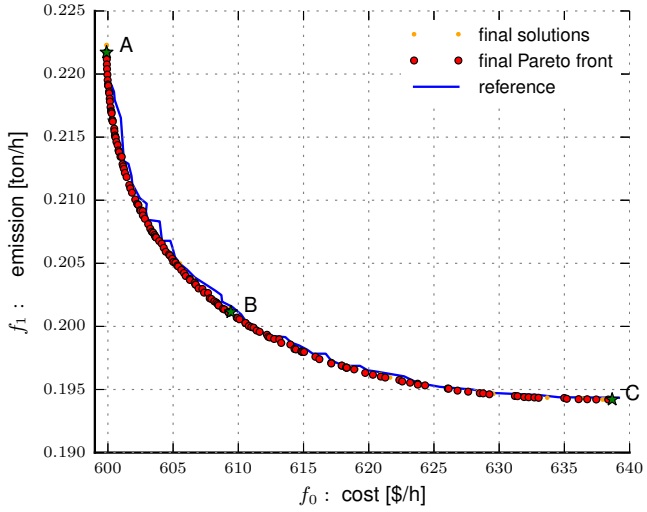



Figure 18: Economic emission load dispatch. Lossless.

where $\alpha_i$, $\beta_i$, $\gamma_i$, $\zeta_i$ and $\lambda_i$ are fitting parameters.

The power output $P_i$ is limited by the generation capacity. Moreover, an equality constraint $h_0$ must be taken into account in order to satisfy the power balance involving the total power generation $\sum P_i$ of the system, the total load demand $P_{demand}$ and the total transmission loss $P_{loss}$. The balance constraint is thus expressed by

$$h_0 = \left(\sum_{i=0}^{N-1} P_i\right) - P_{demand} - P_{loss} \equiv 0 \tag{91}$$

The transmission loss $P_{loss}$ is a nonlinear function of the power outputs $P_i$ and requires the solution of the power flow equations. It depends on other variables such as the bus voltage magnitudes and angles. Nonetheless, a simple expression is available to approximate $P_{loss}$ as follows

$$P_{loss} = B_{00} + \sum_{i=0}^{N-1} B_{0i} P_i + \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} P_i B_{ij} P_j \tag{92}$$

where the $B_{ij}$ coefficients are derived for each particular bus case.

The specific economic-emission dispatch of the *IEEE 30 Bus Test Case* is studied here. Its sketch is presented in Fig. 17 and is based on [62]. The required fitting parameters are listed in Table 9 including the capacity limits. These values are based on data from [62] and [64] (note that there is a typo error in [64] with regards to the $\alpha$ coefficient for $G_4$ in Table 1 of that paper).

The demand load considered in the EED problem is $P_{demand} = 2.834$ and the $B$ coefficients for the IEEE 30 Bus Test case are collected in Table 10 [79–81].

Two cases are considered: a lossless and a lossy situation. In the first one, $P_{loss}$ is set to zero making the problem a little easier. The solution is obtained with $N_{sol} = 200$, $t_{max} = 500$, $N_{cpu} = 4$ and $\Delta t_{exc} = 50$. In addition, the control for the equality constraint $h_0$ is $\epsilon_h = 10^{-3}$ and the differential evolution coefficient is $C_{DE} = 0.8$.

The Pareto-optimal front for the first case (lossless) is illustrated in Fig. 18 where it can be observed that the results match well the reference solution from [65]. The optimal front for the second case (lossy) is illustrated in Fig. 19 where the reference solution from [65] is slightly less optimal than the one computed here.

In both figures, some points corresponding to the minimum cost, a compromise between cost and emission, and the minimum emission are indicated using labels 'A', 'B' and 'C', respectively. The generators power corresponding to these selected points are listed in Table 11 for the lossless case and in Table 12 for the lossy case. The balance errors are also listed in these tables under the $h_0$ column. We observer that the errors fall within the range specified by $\epsilon_h$ as required. It is also interesting to observe that the cost is higher for the lossy case than for the lossless case; a reasonable fact.

## 13. Conclusions

This paper presented an evolutionary algorithm using differential evolution capable of solving a range of optimisation problems, including some with many constraints and more than one objective. Several tests are studied including optimisation problems with up to 20 objective functions. The Pareto optimality is employed for these functions. 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 of the code.

Two applications are studied: topology optimisation of trusses and the economic emission dispatch. For the first one, it is shown that the use of the out-of-range functions helps with the handling of constraints and failures due to calls to the finite element solver that are required to compute the objective

Table 9: IEEE 30 bus: generators parameters.

| G | cost | | | emission | | | | | output limits | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $a$ | $b$ | $c$ | $\alpha/10^{-2}$ | $\beta/10^{-2}$ | $\gamma/10^{-2}$ | $\zeta$ | $\lambda$ | $P_i^{min}$ | $P_i^{max}$ |
| 0 | 10 | 200 | 100 | 4.091 | −5.554 | 6.490 | $2 \cdot 10^{-4}$ | 2.857 | 0.05 | 0.5 |
| 1 | 10 | 150 | 120 | 2.543 | −6.047 | 5.638 | $5 \cdot 10^{-4}$ | 3.333 | 0.05 | 0.6 |
| 2 | 20 | 180 | 40 | 4.258 | −5.094 | 4.586 | $1 \cdot 10^{-6}$ | 8.000 | 0.05 | 1.0 |
| 3 | 10 | 100 | 60 | 5.326 | −3.550 | 3.380 | $2 \cdot 10^{-3}$ | 2.000 | 0.05 | 1.2 |
| 4 | 20 | 180 | 40 | 4.258 | −5.094 | 4.586 | $1 \cdot 10^{-6}$ | 8.000 | 0.05 | 1.0 |
| 5 | 10 | 150 | 100 | 6.131 | −5.555 | 5.151 | $1 \cdot 10^{-5}$ | 6.667 | 0.05 | 0.6 |

Table 10: IEEE 30 bus: $B$ coefficients.

| | | | | | |
|---|---|---|---|---|---|
| $B_{00}$ | | | | | |
| 0.00098573 | | | | | |
| $B_{0i}$ | | | | | |
| −0.0107 | 0.0060 | −0.0017 | 0.0009 | 0.0002 | 0.0030 |
| $B_{ij}$ | | | | | |
| 0.1382 | −0.0299 | 0.0044 | −0.0022 | −0.0010 | −0.0008 |
| −0.0299 | 0.0487 | −0.0025 | 0.0004 | 0.0016 | 0.0041 |
| 0.0044 | −0.0025 | 0.0182 | −0.0070 | −0.0066 | −0.0066 |
| −0.0022 | 0.0004 | −0.0070 | 0.0137 | 0.0050 | 0.0033 |
| −0.0010 | 0.0016 | −0.0066 | 0.0050 | 0.0109 | 0.0005 |
| −0.0008 | 0.0041 | −0.0066 | 0.0033 | 0.0005 | 0.0244 |

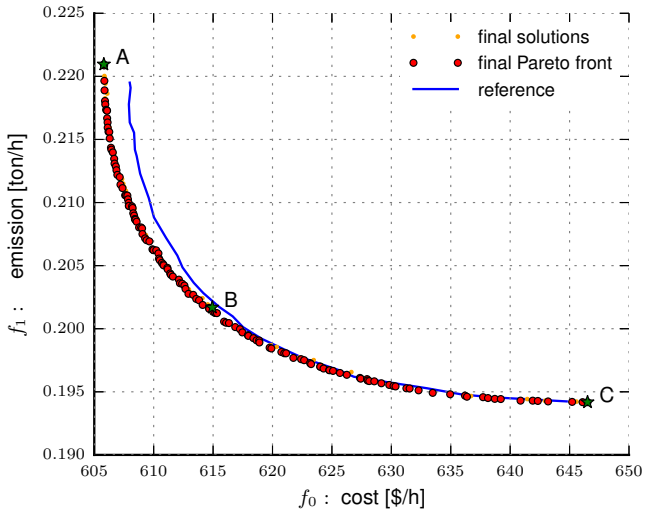



Figure 19: Economic emission load dispatch. Lossy case.

values. The second application involves an equality constraint (the power balance) that makes the problem a little more challenging. Nonetheless, it is demonstrated that the proposed code works well with both applications.

The algorithm is designed to be used in multiple-core 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 to making computations faster. Evidence of the good characteristics of the proposed code including the ideal speedup properties are also shown.

Table 11: Economic emission dispatch. Results: Lossless case.

| point | cost | emission | $h_0$ | $P_0$ | $P_1$ | $P_2$ |
|---|---|---|---|---|---|---|
| | | | | $P_3$ | $P_4$ | $P_5$ |
| A | 599.8973 | 0.221718 | $9.98 \cdot 10^{-4}$ | 0.111932 | 0.304991 | 0.524464 |
| | | | | 1.011370 | 0.525248 | 0.354997 |
| B | 609.4271 | 0.201143 | $4.81 \cdot 10^{-4}$ | 0.278885 | 0.364368 | 0.517304 |
| | | | | 0.714126 | 0.522754 | 0.436083 |
| C | 638.6639 | 0.194210 | $5.56 \cdot 10^{-4}$ | 0.411630 | 0.461994 | 0.528660 |
| | | | | 0.384119 | 0.541038 | 0.507115 |

Table 12: Economic emission dispatch. Results: Lossy case.

| point | cost | emission | $h_0$ | $P_0$ | $P_1$ | $P_2$ |
|---|---|---|---|---|---|---|
| | | | | $P_3$ | $P_4$ | $P_5$ |
| A | 605.8038 | 0.220952 | $9.54 \cdot 10^{-4}$ | 0.111852 | 0.283341 | 0.580172 |
| | | | | 0.991301 | 0.535617 | 0.356460 |
| B | 614.9333 | 0.201684 | $5.37 \cdot 10^{-4}$ | 0.221250 | 0.350637 | 0.592062 |
| | | | | 0.684504 | 0.561183 | 0.449679 |
| C | 646.5103 | 0.194179 | $5.87 \cdot 10^{-4}$ | 0.410754 | 0.467116 | 0.543748 |
| | | | | 0.389823 | 0.543290 | 0.515255 |

## Acknowledgment

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

## References

[1] Goldberg DE. Genetic Algorithms in Search, Optimization and Machine Learning. 1st ed.; Addison-Wesley; 1989. ISBN 0201157675.

[2] Michalewicz Z. Genetic Algorithms + Data Structures = Evolution Programs. Springer; 1996. ISBN 978-3-662-03315-9. doi:10.1007/978-3-662-03315-9.

[3] Deb K. Multi-Objective Optimization using Evolutionary Algorithms. Wiley; 2001. ISBN 978-0-471-87339-6.

[4] Tan KC, Khor EF, Lee TH. Multiobjective Evolutionary Algorithms and Applications. Springer-Verlag London; 2005. ISBN 978-1-84628-132-7. doi:10.1007/1-84628-132-6.

[5] Coello CC, Lamont GB, van Veldhuizen DA. Evolutionary Algorithms for Solving Multi-Objective Problems. Springer; 2007. ISBN 978-0-387-36797-2. doi:10.1007/978-0-387-36797-2.

[6] Goldberg DE, Richardson J. 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. ISBN 0-8058-0158-8; 1987, p. 41–9.

[7] Mahfoud SW. Niching methods for genetic algorithms. Ph.D. thesis; Illinois Genetic Algorithms Laboratory; University of Illinois at Urbana-Champaign; 1995.

[8] Chen CH, Liu TK, Chou JH. A novel crowding genetic algorithm and its applications to manufacturing robots. IEEE Transactions on Industrial Informatics 2014;10(3):1705–16.

[9] Elsayed SM, Sarker Ra, Essam DL. A new genetic algorithm for solving optimization problems. Engineering Applications of Artificial Intelligence 2014;27:57–69. doi:10.1016/j.engappai.2013.09.013.

[10] Mengshoel OJ, Goldberg DE. The crowding approach to niching in genetic algorithms. Evolutionary Computation 2008;16(3):315–54. doi:10.1162/evco.2008.16.3.315.

[11] Mengshoel OJ, Galn SF, de Dios A. Adaptive generalized crowding for genetic algorithms. Information Sciences 2014;258:140–59. doi:10.1016/j.ins.2013.08.056.

[12] Michalewicz Z, Attia N. Evolutionary optimization of constrained problems. In: Sebald A, Fogel L, editors. Proc. of the 3rd Annual Conf. on Evolutionary Programming. World Scientific Publishing, River Edge, NJ; 1994, p. 98–108.

[13] Michalewicz Z. Genetic algorithms, numerical optimization and constraints. In: Proc. of the 6th Int. Conf. on Genetic Algorithms. 1995, p. 151–8.

[14] Michalewicz Z, Schoenauer M. Evolutionary algorithms for constrained parameter optimization problems. Evolutionary Computation 1996;4(1):1–32.

[15] Deb K. An efficient constraint handling method for genetic algorithms. Computer Methods in Applied Mechanics and Engineering 2000;186(2-4):311–38. doi:10.1016/S0045-7825(99)00389-8.

[16] Coello CAC. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Computer Methods in Applied Mechanics and Engineering 2002;191(11–12):1245–87. doi:10.1016/S0045-7825(01)00323-1.

[17] Deb K, Reddy AR, Singh G. Optimal scheduling of casting sequence using genetic algorithms. Materials and Manufacturing Processes 2003;18(3):409–32. doi:10.1081/AMP-120022019.

[18] Vieira D, Adriano R, Vasconcelos J, Krahenbuhl L. Treating constraints as objectives in multiobjective optimization problems using niched pareto genetic algorithm. IEEE Transactions on Magnetics 2004;40(2):1188–91. doi:10.1109/TMAG.2004.825006.

[19] Galán SF, Mengshoel OJ. Constraint handling using tournament selection: Abductive inference in partly deterministic bayesian networks. Evolutionary Computation 2016;17(1):55–88. doi:10.1162/evco.2009.17.1.55.

[20] Long Q. A constraint handling technique for constrained multi-objective genetic algorithm. Swarm and Evolutionary Computation 2014;15:66–79. doi:10.1016/j.swevo.2013.12.002.

[21] Hellwig M, Arnold DV. Comparison of constraint-handling mechanisms for the (1,$\lambda$)-es on a simple constrained problem. Evolutionary Computation 2016;24(1):1–23. doi:10.1162/EVCO_a_00139.

[22] Tanaka M, Watanabe H, Furukawa Y, Tanino T. Ga-based decision support system for multicriteria optimization. In: IEEE Int. Conf. on Systems, Man and Cybernetics; vol. 2. 1995, p. 1556–1561 vol.2. doi:10.1109/ICSMC.1995.537993.

[23] Fonseca CM, Fleming PJ. An overview of evolutionary algorithms in multiobjective optimization. Evolutionary Computation 1995;3(1):1–16. doi:10.1162/evco.1995.3.1.1.

[24] Fonseca CM, Fleming PJ. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. I. a unified formulation. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 1998;28(1):26–37. doi:10.1109/3468.650319.

[25] Fonseca CM, Fleming PJ. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. II. application example. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans 1998;28(1):1–13. doi:10.1109/3468.650320.

[26] Zitzler E, Deb K, Thiele L. Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary Computation 2000;8(2):173–95. doi:10.1162/106365600568202.

[27] Tan K, Lee T, Khor E. Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization. IEEE Transactions on Evolutionary Computation 2001;5(6):565–88. doi:10.1109/4235.974840.

[28] Deb K. Nonlinear goal programming using multi-objective genetic algorithms. Journal of the Operational Research Society 2001;52(3):291–302.

[29] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 2002;6(2):182–97. doi:10.1109/4235.996017.

[30] Deb K, Thiele L, Laumanns M, Zitzler E. Scalable test problems for evolutionary multiobjective optimization. In: Abraham A, Jain L, Goldberg R, editors. Evolutionary Multiobjective Optimization. Springer. ISBN 978-1-85233-787-2; 2005, p. 105–45. doi:10.1007/1-84628-137-7_6.

[31] Kukkonen S, Deb K. A fast and effective method for pruning of non-dominated solutions in many-objective problems. Parallel Problem Solving from Nature 2006;4193:553–62. doi:10.1007/11844297{_}56.

[32] Deb K, Tiwari S. Omni-optimizer: A generic evolutionary algorithm for single and multi-objective optimization. European Journal of Operational Research 2008;185(3):1062–87. doi:10.1016/j.ejor.2006.06.042.

[33] Tiwari S, Koch P, Fadel G, Deb K. AMGA: An archive-based micro genetic algorithm for multi-objective optimization. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation. GECCO '08; New York, NY, USA: ACM. ISBN 978-1-60558-130-9; 2008, p. 729–36. doi:10.1145/1389095.1389235.

[34] Deb K, Miettinen K, Chaudhuri S. Toward an estimation of nadir objective vector using a hybrid of evolutionary and local search approaches. IEEE Transactions on Evolutionary Computation 2010;14(6):821–41. doi:10.1109/TEVC.2010.2041667.

[35] Tiwari S, Fadel G, Deb K. AMGA2: improving the performance of the archive-based micro-genetic algorithm for multi-objective optimization. Engineering Optimization 2011;43(4):377–401. doi:10.1080/0305215X.2010.491549.

[36] Qiu X, Xu JX, Tan KC, Abbass HA. Adaptive cross-generation differential evolution operators for multiobjective optimization. IEEE Transactions on Evolutionary Computation 2016;20(2):232–44. doi:10.1109/TEVC.2015.2433672.

[37] Deb K, Agrawal RB. Simulated binary crossover for continuous search space. Complex Systems 1995;9:115–48.

[38] Deb K, Kumar A. Real-coded genetic algorithms with simulated binary crossover: Studies on multimodal and multiobjective problems. Complex Systems 1995;9:431–54.

[39] Herrera F, Lozano M, Verdegay J. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. Artificial Intelligence Review 1998;12(4):265–319. doi:10.1023/A:1006504901164.

[40] Pedroso DM, Williams DJ. Automatic calibration of soil–water characteristic curves using genetic algorithms. Computers and Geotechnics 2011;38(3):330–40. doi:10.1016/j.compgeo.2010.12.004.

[41] Storn R, Price KV. Differential evolution–a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, ICSI 1995;.

[42] Price K, Storn RM, Lampinen JA. Differential Evolution: A Practical Approach to Global Optimization. Springer-Verlag; 2005. doi:10.1007/3-540-31306-0.

[43] Kukkonen S, Lampinen J. GDE3: the third evolution step of generalized differential evolution. In: The 2005 IEEE Congress on Evolutionary Computation; vol. 1. 2005, p. 443–50. doi:10.1109/CEC.2005.1554717.

[44] Kukkonen S, Lampinen J. An empirical study of control parameters for the third version of generalized differential evolution (GDE3). In: IEEE Int. Conf. on Evolutionary Computation. ISBN 0-7803-9487-9; 2006, p. 2002–9. doi:10.1109/CEC.2006.1688553.

[45] The Go Authors . The go programming language. https://golangorg 2016;URL: https://golang.org.

[46] Ray T, Tai K, Seow KC. Multiobjective design optimization by an evolutionary algorithm. Engineering Optimization 2001;33(4):399–424. doi:10.1080/03052150108940926.

[47] Ray T, Singh HK, Isaacs A, Smith W. Constraint-Handling in Evolutionary Optimization; chap. Infeasibility Driven Evolutionary Algorithm for Constrained Optimization. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-00619-7; 2009, p. 145–65. doi:10.1007/978-3-642-00619-7_7.

[48] Runarsson TP, Yao X. Stochastic ranking for constrained evolutionary optimization. IEEE Transactions on Evolutionary Computation 2000;4(3):284–94. doi:10.1109/4235.873238.

[49] Isaacs A, Ray T, Smith W. Blessings of maintaining infeasible solutions for constrained multi-objective optimization problems. In: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence). 2008, p. 2780–7. doi:10.1109/CEC.2008.4631171.

[50] Datta R, Deb K. Individual penalty based constraint handling using a hybrid bi-objective and penalty function approach. In: 2013 IEEE Congress on Evolutionary Computation. 2013, p. 2720–7. doi:10.1109/CEC.2013.6557898.

[51] Hock W, Schittkowski K. Test Examples for Nonlinear Programming Codes. Springer; 1981. ISBN 978-3-642-48320-2. doi:10.1007/978-3-642-48320-2.

[52] Ragsdell KM, Phillips DT. Optimal design of a class of welded structures using geometric programming. Journal of Engineering for Industry 1976;98:1021–5. doi:10.1115/1.3438995.

[53] Ravindran A, Ragsdell KM, Reklaitis GV. Engineering Optimization: Methods and Applications, Second Edition. Wiley; 2007. ISBN 9780470117811. doi:10.1002/9780470117811.

[54] Deb K, Jain H. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints. IEEE Transactions on Evolutionary Computation 2014;18(4):577–601. doi:10.1109/TEVC.2013.2281535.

[55] Jain H, Deb K. An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part II: Handling constraints and extending to an adaptive approach. IEEE Transactions on Evolutionary Computation 2014;18(4):602–22. doi:10.1109/TEVC.2013.2281534.

[56] Ruy WS, Yang YS, Kim GH, Yeun YS. Topology design of truss structures in a multicriteria environment. Computer-Aided Civil and Infrastructure Engineering 2001;16(4):246–58. doi:10.1111/0885-9507.00230.

[57] Wu CY, Tseng KY. Truss structure optimization using adaptive multi-population differential evolution. Structural and Multidisciplinary Optimization 2010;42(4):575–90. doi:10.1007/s00158-010-0507-9.

[58] Noilublao N, Bureerat S. Simultaneous topology, shape, and sizing optimisation of plane trusses with adaptive ground finite elements using moeas. Mathematical Problems in Engineering 2013;2013:1–9. doi:10.1155/2013/838102.

[59] Cazacu R, Grama L. Steel truss optimization using genetic algorithms and fea. Procedia Technology 2014;12:339–46. doi:10.1016/j.protcy.2013.12.496.

[60] Gogu G. Mobility of mechanisms: a critical review. Mechanism and Machine Theory 2005;40(9):1068 –97. doi:http://dx.doi.org/10.1016/j.mechmachtheory.2004.12.014.

[61] Li JP. Truss topology optimization using an improved species-conserving genetic algorithm. Engineering Optimization 2015;47(1):107–28. doi:10.1080/0305215X.2013.875165.

[62] Yokoyama R, Bae SH, Morita T, Sasaki H. Multiobjective optimal generation dispatch based on probability security criteria. IEEE Transactions on Power Systems 1987;3(1):317–24. doi:10.1109/59.43217.

[63] Farag A, Al-Baiyat S, Cheng TC. Economic load dispatch multiobjective optimization procedures using linear programming techniques. IEEE Transactions on Power Systems 1995;10(2):731–8.

[64] Abido Ma. A niched pareto genetic algorithm for multiobjective environmental/economic dispatch. Int Journal of Electrical Power and Energy Systems 2003;25(2):97–105. doi:10.1016/S0142-0615(02)00027-3.

[65] Abido M. Multiobjective evolutionary algorithms for electric power dispatch problem. IEEE Transactions on Evolutionary Computation 2006;10(3):315–29. doi:10.1109/TEVC.2005.857073.

[66] AlRashidi M, El-hawary M. Economic dispatch with environmental considerations using particle swarm optimization. Large Engineering Systems Conf on Power Engineering 2006;:41–6doi:10.1109/LESCPE.2006.280357.

[67] Kumar S, Naresh R. Nonconvex economic load dispatch using an efficient real-coded genetic algorithm. Applied Soft Computing 2009;9(1):321–9. doi:10.1016/j.asoc.2008.04.009.

[68] Dhanalakshmi S, Kannan S, Mahadevan K, Baskar S. Application of modified nsga-ii algorithm to combined economic and emission dispatch problem. Int Journal of Electrical Power and Energy Systems 2011;33(4):992–1002. doi:10.1016/j.ijepes.2011.01.014.

[69] Bayón L, Grau JM, Ruiz MM, Suárez PM. The exact solution of the environmental / economic dispatch problem. IEEE Transactions on Power Systems 2012;27(2):723–31.

[70] Rahmat NA, Musirin I, Abidin AF. Differential evolution immunized ant colony optimization (DEIANT) technique in solving economic emission dispatch. Int Conf on Techn, Informatics, Management, Engrg, and Environment (TIME-E) 2013;:198–202doi:10.1109/TIME-E.2013.6611991.

[71] Li YF, Pedroni N, Zio E. A memetic evolutionary multi-objective optimization method for environmental power unit commitment. IEEE Transactions on Power Systems 2013;28(3):2660–9. doi:10.1109/TPWRS.2013.2241795.

[72] Jubril A, Komolafe O, Alawode K. Solving multi-objective economic dispatch problem via semidefinite programming. IEEE Transactions on Power Systems 2013;28(3):2056–64. doi:10.1109/TPWRS.2013.2245688.

[73] Bilil H, Aniba G, Maaroufi M. Probabilistic economic/environmental power dispatch of power system integrating renewable energy sources. Sustainable Energy Technologies and Assessments 2014;8:181–90. doi:10.1016/j.seta.2014.09.002.

[74] Jeddi B, Vahidinasab V. A modified harmony search method for environmental/economic load dispatch of real-world power systems. Energy Conversion and Management 2014;78:661–75. doi:10.1016/j.enconman.2013.11.027.

[75] Jubril A, Olaniyan O, Komolafe O, Ogunbona P. Economic-emission dispatch problem: A semi-definite programming approach. Applied Energy 2014;134:446–55. doi:10.1016/j.apenergy.2014.08.024.

[76] Sayah S, Hamouda A, Bekrar A. Efficient hybrid optimization approach for emission constrained economic dispatch with nonsmooth cost curves. Int Journal of Electrical Power & Energy Systems 2014;56(0):127–39. doi:http://dx.doi.org/10.1016/j.ijepes.2013.11.001.

[77] Bhattacharjee K, Bhattacharya A, Halder S. Electrical power and energy systems backtracking search optimization based economic environmental power dispatch problems. Int Journal of Electrical Power and Energy Systems 2015;73:830–42. doi:10.1016/j.ijepes.2015.06.018.

[78] Zhang H, Yue D, Xie X, Hu S, Weng S. Multi-elite guide hybrid differential evolution with simulated annealing technique for dynamic economic emission dispatch. Applied Soft Computing 2015;34:312–23. doi:10.1016/j.asoc.2015.05.012.

[79] Wang L, Singh C. Balancing risk and cost in fuzzy economic dispatch including wind power penetration based on particle swarm optimization. Electric Power Systems Research 2008;78(8):1361–8. doi:10.1016/j.epsr.2007.12.005.

[80] Özyön S, TemurtaÅ H, DurmuÅ B, Kuvat G. Charged system search algorithm for emission constrained economic power dispatch problem. Energy 2012;46(1):420–30. doi:10.1016/j.energy.2012.08.008.

[81] Pal B, Biswas P, Mukhopadhyay A. Using genetic algorithm to goal programming model of solving economic-environmental electric power generation problem with interval-valued target goals. In: Mathematical Modelling and Scientific Computation; vol. 283. Springer. ISBN 978-3-642-28925-5; 2012, p. 156–69. doi:10.1007/978-3-642-28926-2_17.