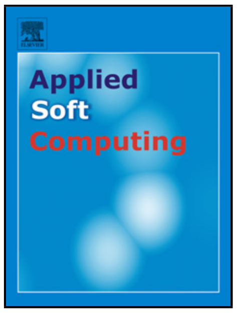


Title: Parallel evolutionary algorithm for single and multi-objective optimisation: differentialevolution and constraints handling

Author: Dorival M. Pedroso Mohammad Reza Bonyadi Marcus Gallagher

PII: S1568-4946(17)30545-8
DOI: http://dx.doi.org/doi:10.1016/j.asoc.2017.09.006
Reference: ASOC 4453

To appear in: *Applied Soft Computing*

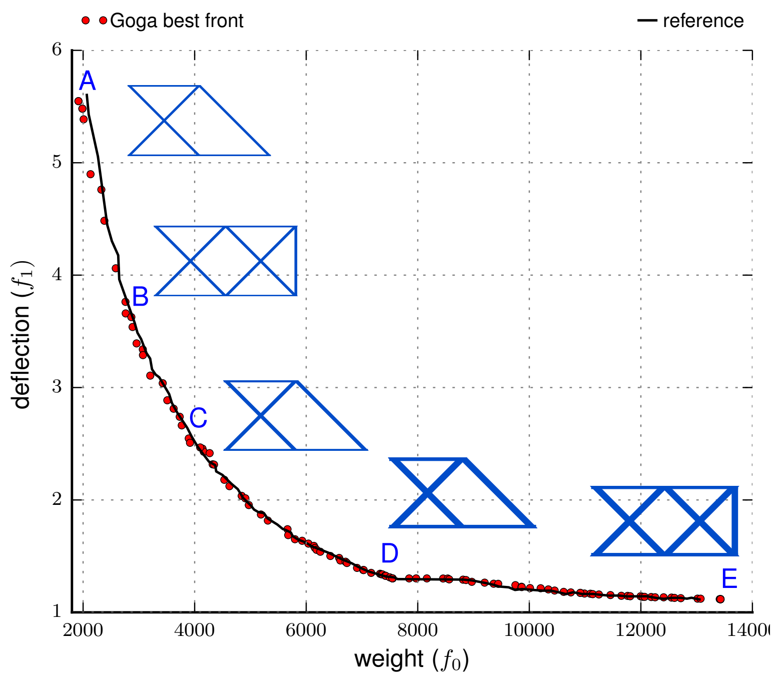
Received date: 11-12-2016
Revised date: 16-5-2017
Accepted date: 3-9-2017

Highlights–Parallel evolutionary algorithm for single and multi-objective optimisation: differential evolution and constraints handling

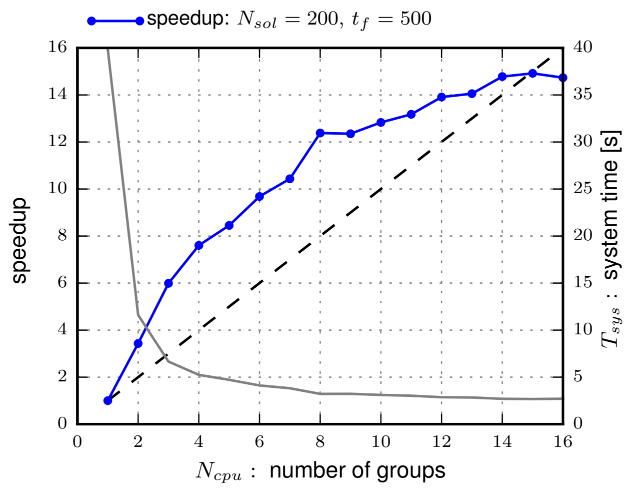
Dorival M. Pedroso, Mohammad Reza Bonyadi, Marcus Gallagher

May 14, 2017

- New Pareto comparison of ranked constraints (out-of-range functions, OORs)
- Differential evolution with random coefficients to accommodate single and multiobjective problems
- New algorithms for constraints handling alongside diversity maintenance and Pareto front spread
- Carefully designed comparison routines for candidate solutions
- Parallel algorithm that uses the differential evolution and preserves diversity
- Applications to real engineering problems

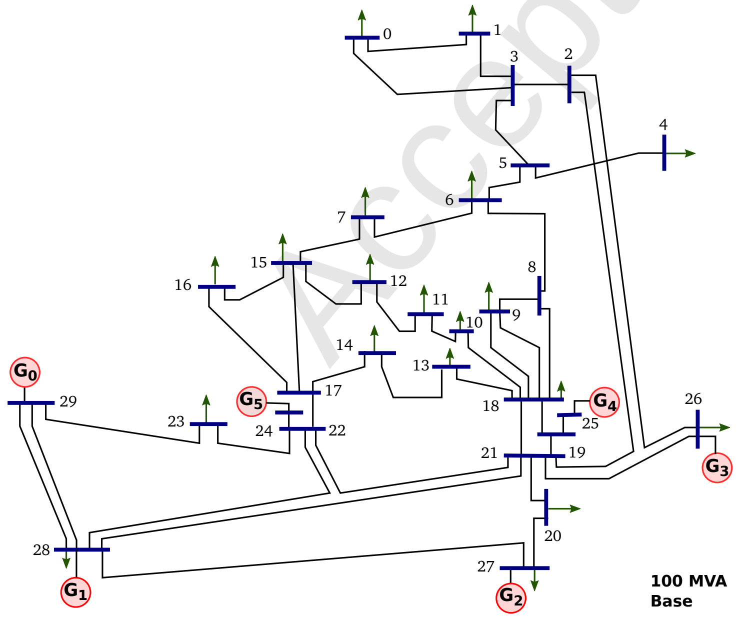


(a)

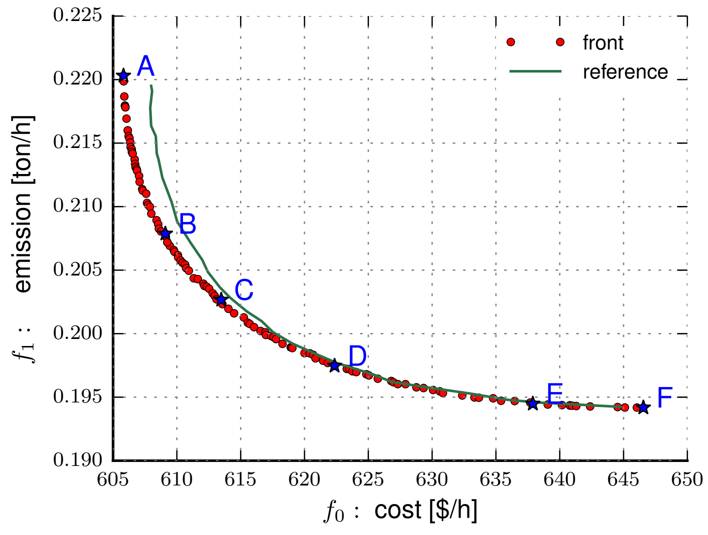


(b)

Shape and topology optimisation. (a) Optimal solutions and Pareto front. (b) Speedup.



(a)



(b)

Economic emission dispatch in power generation. (a) IEEE 30 bus test system. (b) Optimal front: lossy case.

Parallel evolutionary algorithm for single and multi-objective optimisation: differential evolution and constraints handling

Dorival M. Pedroso[a,*], Mohammad Reza Bonyadi[b], Marcus Gallagher[c]

[a]*School of Civil Engineering, The University of Queensland, St Lucia QLD 4072, Australia*
[b]*Centre for Advanced Imaging, The University of Queensland, St Lucia QLD 4072, Australia*
[c]*Information Technology and Electrical Engineering, The University of Queensland, St Lucia QLD 4072, Australia*

Abstract

This paper presents an evolutionary algorithm employing differential evolution to solve nonlinear optimisation problems with (or without) constraints and multiple objectives. New decision strategies to compare candidate solutions are developed that take into account all constraints and objective functions. The new constraint handling strategy uses the concept of Pareto dominance to rank the candidate solutions based on their constraint violation value. In order to improve the performance of the algorithm, a set of genetic operators and differential evolution operators are combined. In addition, the paper proposes an algorithm to perform parallel evolution in a way that the diversity of the final population is preserved after migrations. Another goal of the algorithm is to handle problems with a mix of integers and real-valued variables. Numerical experiments investigate the robustness and the performance of the algorithm through multiple benchmark optimisation problems. Finally, two engineering applications are studied, namely: (1) the topology optimisation of trusses; and (2) the economical dispatch problem in power generation. Results show that the algorithm is capable of handling optimisation problems with a mix of integer and real-valued variables with constraints and multiple objectives.

Keywords:
differential evolution, constraints handling, many objectives, truss optimisation, economic dispatch, parallel computing

1. Introduction

Evolutionary algorithms (EAs), including genetic algorithms (GAs), are optimisation techniques well disseminated in engineering and 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 always achieve the optimal answer and cannot easily tell whether the solution has converged or not. Therefore, EAs must be well developed and tested under several circumstances to be reliable and able to output accurate results after every run.

One essential characteristic of an evolutionary algorithm is the ability to maintain diversity during the optimisation process. Simple GAs, in particular, suffer

[*]Corresponding author
Email addresses: d.pedroso@uq.edu.au (Dorival M. Pedroso), reza@cai.uq.edu.au (Mohammad Reza Bonyadi), marcusg@itee.uq.edu.au (Marcus Gallagher)

from early convergence when, after some iterations, most individuals have very similar genotype and cannot easily produce better solutions. 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 paper considers one solution to this problem that is 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 candidate solutions and only replacing the previous candidate if the new solution wins.

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 in power generation where the capacity of generators is limited, and the available power must compensate the losses.

Methods to handle constraints in evolutionary algorithms range from the use of penalty factors to classification strategies when comparing candidate solutions. Unless an estimate is possible in a specific problem, the penalty method is 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 applied to both the objective values and measures of how close to satisfying constraints a candidate 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. Several algorithms have then been developed along the years to solve multi-objective problems as well [22–38].

Real-coded genetic algorithms have been a challenge in the early days, but good solutions were introduced and studied in, e.g. [39, 40, 2, 41]. A simple attempt that split a float point number into smaller chunks was also successfully proposed and implemented in [42]. Nonetheless, the performance was not necessarily always the best. The differential evolution (DE) [43, 44] 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 [36]. This behaviour has also been observed in the experiments performed. More details on DE can be found in [45, 46].

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 to make better use of the computer hardware. For evolutionary algorithms, concurrency is very natural since candidate solutions can be assigned into different subsets of candidate solutions (groups, islands, etc.) that are run in parallel, hence speeding up computations. Nonetheless, care must be taken to prevent the loss of diversity when exchanging data (e.g. migration) between different populations. Thus, a strategy to implement concurrency is also investigated and proposed.

The algorithms in this paper are implemented in Go language (golang)[1] but

[1]The resulting code is available at https://github.com/cpmech/goga as free/open source

are presented in a general format so they can be easily implemented in any language.

In summary, we develop:

1. Differential evolution combined with classical evolutionary computation.
2. A new constraint handling strategy where the concept of Pareto dominance is extended to functions that measure how much a constraint has been violated.
3. An algorithm to perform parallel evolution, in a way that the diversity of the final population is preserved after migrations.
4. An algorithm that can handle general situations and that can solve problems involving combinations of real numbers and integers.

Finally, this paper presents several numerical experiments to investigate the ability of the algorithm in generating similar results after many runs; i.e. to be reliable. Finally, two engineering applications are studied, namely: (1) the topology optimisation of trusses; and (2) the economical dispatch problem in power generation.

2. Problem definition

An optimisation problem with multiple objectives, multiple constraints, and mixed types can be defined as follows:

$$\begin{aligned}
&\underset{\underset{\sim}{x},\,\underset{\sim}{y}}{\text{minimise}} && \{f_0(\underset{\sim}{x},\underset{\sim}{y}), f_1(\underset{\sim}{x},\underset{\sim}{y}), \ldots\} && && (N_f \text{ objectives})\\
&\text{subject to} && g_i(\underset{\sim}{x},\underset{\sim}{y}) \geq 0 && && (N_g \text{ constraints})\\
& && h_i(\underset{\sim}{x},\underset{\sim}{y}) = 0 && && (N_h \text{ constraints})\\
& && x_i^L \leq x_i \leq x_i^U && (x_i \in \mathbb{R}) && (N_x \text{ box-constr.})\\
& && y_i^L \leq y_i \leq y_i^U && (y_i \in \mathbb{Z}) && (N_y \text{ box-constr.})
\end{aligned} \tag{1}$$

where $f_i(\underset{\sim}{x},\underset{\sim}{y}) \in \mathbb{R}$ are the N_f objective functions, $\underset{\sim}{x} = \{x_0, x_1, \ldots, x_{Nx-1}\}$ are the design variables (unknowns) of real type, $\underset{\sim}{y} = \{y_0, y_1, \ldots, y_{Ny-1}\}$ are the design variables of integer type, and $g_i(\underset{\sim}{x},\underset{\sim}{y}) \in \mathbb{R}$ and $h_i(\underset{\sim}{x},\underset{\sim}{y}) \in \mathbb{R}$ are inequality and equality constraint functions, respectively. In this paper, when counting constraints, the box constraints are not included; hence, if only box constraints are present, the problem is called unconstrained. Also, the objective values are denoted as OVAs.

The inequality constraints in Eq. (1) are converted into equality constraints and the set of all constraints (except 'boxed constraints') is modelled by another function, called *out-of-range* (OOR) and denoted as $u_i(\underset{\sim}{x},\underset{\sim}{y})$. The OOR function returns only positive values (i.e. $u_i \in \mathbb{R}_{\geq 0}$) that are proportional to the constraint violation. See comprehensive discussion on this topic here [47–49].

Therefore, the model problem for the evolutionary algorithm is:

$$\begin{aligned}
&\underset{\underset{\sim}{x},\,\underset{\sim}{y}}{\text{minimise}} && \{f_0(\underset{\sim}{x},\underset{\sim}{y}), f_1(\underset{\sim}{x},\underset{\sim}{y}), \ldots\} && && (N_f \text{ objectives})\\
&\text{subject to} && u_i(\underset{\sim}{x},\underset{\sim}{y}) = 0 && (u_i \in \mathbb{R}_{\geq 0}) && (N_u \text{ constraints})\\
& && x_i^L \leq x_i \leq x_i^U && (x_i \in \mathbb{R}) && (N_x \text{ box-constr.})\\
& && y_i^L \leq y_i \leq y_i^U && (y_i \in \mathbb{Z}) && (N_y \text{ box-constr.})
\end{aligned} \tag{2}$$

and takes advantage of the native concurrency features of Go.

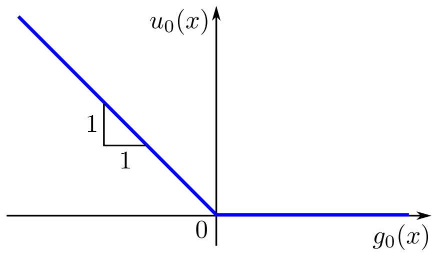


Figure 1: Out-of-range function u_0 corresponding to inequality constraint g_0.

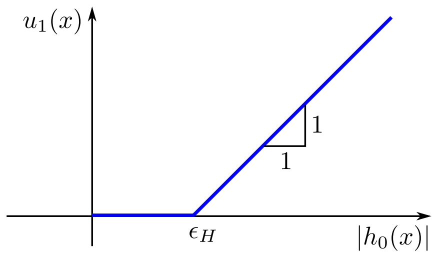


Figure 2: Out-of-range function u_1 corresponding to equality constraint h_0.

The OOR function $u_i(\underset{\cdot}{\boldsymbol{x}}, \underset{\cdot}{\boldsymbol{y}})$ will help with the comparison of candidate solutions as explained shortly. The number of OOR functions is $N_u = N_g + N_h$. Furthermore, the g_i and h_i constraints can be easily converted into the u_i form with the aid of the ramp function as follows (Figs. 1 and 2):

$$u_0(x) = \begin{cases} 0 & \text{if } g_0(x) \geq 0 \\ -g_0(x) & \text{otherwise} \end{cases} \tag{3}$$

and

$$u_1(x) = \begin{cases} 0 & \text{if } |h_0(x)| \leq \epsilon_H \\ |h_0(x)| - \epsilon_H & \text{otherwise} \end{cases} \tag{4}$$

where ϵ_H is a small number, but not very small (i.e. not machine precision). The use of ϵ_H is necessary to search within a small gap close to the equality constraint.

It is worth noting that, 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 handle multiple constraints effectively and, in the proposed evolutionary algorithm, this method will pull infeasible solutions towards the feasibility region.

The solution of the optimisation problem defined in Eq. (2) is obtained in a stochastic manner starting with the definition of a set S of candidate solutions:

$$S = \{(\underset{\cdot}{\boldsymbol{x}}, \underset{\cdot}{\boldsymbol{y}})_0, (\underset{\cdot}{\boldsymbol{x}}, \underset{\cdot}{\boldsymbol{y}})_1, \ldots (\underset{\cdot}{\boldsymbol{x}}, \underset{\cdot}{\boldsymbol{y}})_{Nsol-1}\} \tag{5}$$

where $(\underset{\cdot}{\boldsymbol{x}}, \underset{\cdot}{\boldsymbol{y}})_i$ is a candidate solution and N_{sol} is the number of candidate solutions. Each solution corresponds to a vector of OVAs $\underset{\cdot}{\boldsymbol{f}}_i$ and a vector of OORs $\underset{\cdot}{\boldsymbol{u}}_i$. For example, $\underset{\cdot}{\boldsymbol{f}}_A(\underset{\cdot}{\boldsymbol{x}}_A, \underset{\cdot}{\boldsymbol{y}}_A)$ is the vector of objective values computed with $(\underset{\cdot}{\boldsymbol{x}}_A, \underset{\cdot}{\boldsymbol{y}}_A)$.

The concept of Pareto domination is considered; see e.g. [24], where a vector $\underset{\cdot}{\boldsymbol{v}}_A$ dominates another one $\underset{\cdot}{\boldsymbol{v}}_B$ if, and only if, the following conditions are met at the same time

$$\begin{array}{lrl} v_i^A \leq v_i^B & \text{for all} & i = 0, 1, \ldots N_v - 1 \\ v_i^A < v_i^B & \text{for at least one} & i = 0, 1, \ldots N_v - 1 \end{array} \tag{6}$$

where v_i^j are the components of the vector $\underset{\cdot}{\boldsymbol{v}}_j$ and N_v is the length of the vector. Note that Eq. (6) will be applied to OVAs or OORs as explained in Section 3. In other words, $\underset{\cdot}{\boldsymbol{v}}_i$ will be $\underset{\cdot}{\boldsymbol{f}}_i$ or $\underset{\cdot}{\boldsymbol{u}}_i$ and cases such as when A dominates B but A is infeasible will be properly handled; see also discussion in [49].

For multi-objective optimisation problems, the word minimise in Eq. (2) means to find the Pareto optimal solution satisfying Eq. (6) using the vectors $\underset{\cdot}{\boldsymbol{f}}_i$ of objective values. Single-objective optimisation problems are also included in Eq. (2) with $N_f = 1$.

In this paper, the command `ParetoComparison` means to apply Eq. (6) to any two vectors related to the candidate solutions A and B and to find whether solution A dominates solution B ($A_{dom} = \text{true}$), solution B dominates solution A ($B_{dom} = \text{true}$), or none of the above. Thus, it can be implemented in a routine that takes two vector arguments and returns two Boolean results as follows

$$A_{dom}, B_{dom} \leftarrow \texttt{ParetoComparison}\left(\underset{\cdot}{\boldsymbol{v}}_A, \underset{\cdot}{\boldsymbol{v}}_B\right) \tag{7}$$

3. Constraints handling

The method employed to find solutions satisfying all constraints is inspired by the approach presented in [15]; but avoids adding weights to the objective function (penalty method). The proposed algorithm is designed to deal with multi-objective problems at the same time as single-objective ones. The handling is accomplished by means of a Pareto comparison of OORs in addition to a Pareto comparison of OVAs. It is worth noting that other algorithms exist to avoid penalties as well [29], including the use of Pareto comparisons and a ranking procedure [50, 51]. 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 approaches are available in [52–54].

The motivation for the method is on how some constraints are dealt with in civil engineering. Specifically, a candidate solution not satisfying all the constraints, thus termed infeasible, is completely ignored at design stages—for example, an infeasible structure 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 candidate solutions. This is facilitated by the OOR functions.

It might be the case that a candidate solution is very close to the best possible feasible solution in the search space while violating all constraints. On the other way around, we only find non-optimal solutions that are feasible. This problem with infeasible candidates that are very close to the optimal is by no means an easy one; see comprehensive discussion in [47–49]. In this paper, we have decided upon a hierarchical procedure where satisfying the number of violations comes first. Then, we conduct a Pareto comparison employing the OOR values, and, finally, we carry out a conventional Pareto comparison using the OVAs. We have found that this procedure works well with the problems studied here.

The constraints handling procedure is summarised in Algorithm 1. In essence, the comparison between two candidate solutions A and B is carried out by first checking the number of constraint violations N_{viol}^i due to solution i and by performing a Pareto comparison on the OOR values $\underset{\cdot}{\boldsymbol{u}}_i$. Therefore, in this paper, 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

Algorithm 1. `CompareSolutions`(A,B) performs the comparison between trial solutions considering the OVAs and the OORs.

```
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 $i$ in $[0, N_u)$ 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($\underset{\cdot}{\boldsymbol{u}}_A, \underset{\cdot}{\boldsymbol{u}}_B$)
        if not $A_{dom}$ and not $B_{dom}$ then
            $A_{dom}, B_{dom} \leftarrow$ ParetoComparison($\underset{\cdot}{\boldsymbol{f}}_A, \underset{\cdot}{\boldsymbol{f}}_B$)
        return
    $B_{dom} \leftarrow$ true
    return
if $N^B_{viol} > 0$ then
    $A_{dom} \leftarrow$ true
    return
$A_{dom}, B_{dom} \leftarrow$ ParetoComparison($\underset{\cdot}{\boldsymbol{f}}_A, \underset{\cdot}{\boldsymbol{f}}_B$)
```

the OVA values $\underset{\cdot}{\boldsymbol{f}}_i$. The output of Algorithm 1 includes two flags describing the domination status between A and B; it can happen that yet none dominates the other. This situation will be properly handled during tournaments as presented later on.

4. Metrics

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

- x_i^{min} and x_i^{max} minimum and maximum solution values of real type in Ω corresponding to dimension i
- y_i^{min} and y_i^{max} minimum and maximum solution values of integer type in Ω corresponding to dimension i
- f_i^{min} and f_i^{max} minimum and maximum objective values in Ω corresponding to objective function i
- η_i *neighbour distance*: smallest distance to any neighbour around solution i

- w_i *crowding distance* corresponding to solution i
- ϕ_i rank of the Pareto front of solution i (small is better)

In this paper, we also employ a very simple measure of how close a candidate solution is to its neighbours. This metric is called the neighbour distance η_i. First the distance d_{ij} between solutions i and j is computed as follow

$$d_{ij} = \frac{1}{N_x} \sum_{k}^{N_x} \frac{x_k^i - x_k^j}{x_k^{max} - x_k^{min} + \epsilon} + \frac{1}{N_y} \sum_{k}^{N_y} \frac{y_k^i - y_k^j}{y_k^{max} - y_k^{min} + \epsilon} \tag{8}$$

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

The rank ϕ_i of the Pareto front of the 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 ϕ).

The steps to compute the Pareto fronts are collected in Algorithm 2 where W_i is a subset of solutions such that solution i wins the comparison against any item in this subset when using `CompareSolutions`. The algorithm allocates 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 [36]. 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 ϕ). For each front, if the front has more than two solutions, the crowding distance w_i of a solution i (not at the extremities of the front) is computed by

$$w_i = \sum_{j}^{N_f} \left(\frac{f_j^i - f_j^{i-1}}{\delta} \right) \left(\frac{f_j^{i+1} - f_j^i}{\delta} \right) \tag{9}$$

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 to cover a wider range of values. The steps are summarised in Algorithm 3.

5. Main loop

The main loop of the evolutionary algorithm is presented in Algorithm 4. In summary, a set (population) of candidate solutions is firstly generated and later organised into smaller and distinct groups. Within each group, evolution is simulated in parallel where candidate solutions are combined to explore the search space aiming at finding better solutions. The pairing of solutions is random 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 (e.g. migration).

Algorithm 2. `ParetoFronts`(Ω) computes Pareto fronts.

```
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 $r \in [0, N_{sol})$ do
    if $z_r = 0$ then
        break
    $s \leftarrow r + 1$
    for solution indices $i \in [0, z_r)$ 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
```

Before the evolution proceeds, the set S of N_{sol} candidate solutions is randomly generated. The generation considers the limits of variables; thus the search space is 'boxed'. The Latin Hypercube method is employed to this task. Afterwards, the candidate solutions array is mapped into N_{cpu} smaller groups G that can be independently evolved.

The evolution process is simulated from an initial time $t = 0$ to the final time $t = t_{max}$. This happens in larger increments Δt_{exc} corresponding to the time passed between the exchange of solutions among groups. Four major steps are followed in the main loop of the proposed algorithm. This is explained in Algorithm 4.

After the parallel evolution, the `Metrics` function is called considering all solutions to update the information required before the exchange of solutions. The call of `Metrics` with all solutions is necessary for the execution of tournaments. In this way, although chunks of solutions are grouped, their effect in the whole

Algorithm 3. `CrowdingDistances`(Ω) computes the crowding distances.

```
Input: output data from ParetoFronts($\Omega$)
Output: crowding distances $w_i$
zero all $w_i$ values
for each Pareto front $F_{r,\mathrm{all}}$ 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($F_{r,\mathrm{all}}$, "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
            $w_i \mathrel{+}= \frac{f_j^i - f_j^{i-1}}{\delta} \times \frac{f_j^{i+1} - f_j^i}{\delta}$
```

set is taken into account. For instance, values such as neighbour distance and Pareto fronts will be computed for the whole set. Therefore, the exchange will be biased towards the best candidates of all groups.

The parallel code works by exchanging solutions in time intervals (Δt_{exc}) where each independent group has a chance to search for solutions in different spaces. The best solutions are then transferred from group to group. The algorithm implements the exchange in a cyclic fashion: from groups with lower indices to groups with higher indices, except that, for the last pair, 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 occurs 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 update of candidate solutions to a next generation (evolution) is independently carried out for a group of solutions G. In Algorithm 4, the evolution corresponds to the `EvolveOneGroup` command. The crowding approach to niching explained in [10] is adopted, but small modifications are applied, in particular, with intentions to tackle multi-objective optimisation problems. Algorithm 5 summarises all steps in `EvolveOneGroup`.

First, a bi-dimensional array P (pairs) with the indices of candidate 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, as required by the DE operator that will be presented shortly.

Next, eight candidate solutions are extracted from the group where four solutions corresponds to A and the other four to B. This is easily accomplished by a cyclic permutation of indices in P. Then, the real parts of the solutions are

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 $t_{exc}$
    $done \leftarrow$ communication channel (size=$N_{cpu}$)
    for each group $i$ do
        execute concurrently ("spawn")
            for $time \leftarrow t$ to $time < t_{exc}$ do
                EvolveOneGroup($I$)
            send $done \leftarrow$ true via $channel$
    for each group $i$ do
        collect done flag from channel
    // compute metrics
    Metrics($S$)
    // exchange solutions via tournament
    for each group $i$ do
        $j \leftarrow (i+1) \,\%\, N_{cpu}$
        $A, B \leftarrow$ random solutions from $G_i$
        $a,\ b \leftarrow$ random solutions from $G_j$
        Tournament($S,\ \ A, B, a, b$)
    // exchange one solution randomly
    select non-repeated pairs of groups
    for each random pair $(G_0, G_1)$ do
        $A \leftarrow$ random solution from $G_0$
        $B \leftarrow$ random solution from $G_1$
        swap $A$ by $B$ in $S$
    // update time variables
    $t \mathrel{+}= \Delta t_{exc}$
    $t_{exc} \mathrel{+}= \Delta t_{exc}$
```

given to the differential evolution function producing two new solutions a and b. Furthermore, classical genetic operations on the integer parts are performed.

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 we only were to use the DE operation, it would not matter how the eight solutions were selected, as long as the process is random and involved unique candidates.

The new solutions are stored in a backup set represented by Γ. After that, the `Metrics` function is called with the union of the old (G) **and** the new populations (Γ). In this way, the information required for tournaments such as the Pareto front, neighbour and crowding distances and limits will consider the presence of the new solutions in the space of all solutions. This feature, for instance, will induce new solutions to be less crowded when after all solutions are carried over 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 matching the closest pairs of old

Algorithm 5. `EvolveOneGroup`(G) evolves group.

```
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 $k \leftarrow 0$ to $k < N_p$ do
    $l \leftarrow (k+1)\,\%\,N_p$
    $m \leftarrow (k+2)\,\%\,N_p$
    $n \leftarrow (k+3)\,\%\,N_p$
    $\underset{\sim}{x}_A \leftarrow$ RealPart($G[P_{k,0}]$)
    $\underset{\sim}{x}_{A0} \leftarrow$ RealPart($G[P_{l,0}]$)
    $\underset{\sim}{x}_{A1} \leftarrow$ RealPart($G[P_{m,0}]$)
    $\underset{\sim}{x}_{A2} \leftarrow$ RealPart($G[P_{n,0}]$)
    $\underset{\sim}{x}_B \leftarrow$ RealPart($G[P_{k,1}]$)
    $\underset{\sim}{x}_{B0} \leftarrow$ RealPart($G[P_{l,1}]$)
    $\underset{\sim}{x}_{B1} \leftarrow$ RealPart($G[P_{m,1}]$)
    $\underset{\sim}{x}_{B2} \leftarrow$ RealPart($G[P_{n,1}]$)
    $\underset{\sim}{x}_a \leftarrow$ DiffEvol($\underset{\sim}{x}_A, \underset{\sim}{x}_{A0}, \underset{\sim}{x}_{A1}, \underset{\sim}{x}_{A2}$)
    $\underset{\sim}{x}_b \leftarrow$ DiffEvol($\underset{\sim}{x}_B, \underset{\sim}{x}_{B0}, \underset{\sim}{x}_{B1}, \underset{\sim}{x}_{B2}$)
    $\underset{\sim}{y}_A \leftarrow$ IntPart($G[P_{k,0}]$)
    $\underset{\sim}{y}_B \leftarrow$ IntPart($G[P_{k,1}]$)
    $\underset{\sim}{y}_a, \underset{\sim}{y}_b \leftarrow$ Crossover($\underset{\sim}{y}_A, \underset{\sim}{y}_B$)
    $\underset{\sim}{y}_a \leftarrow$ Mutation($\underset{\sim}{y}_a$)
    $\underset{\sim}{y}_b \leftarrow$ Mutation($\underset{\sim}{y}_b$)
    store $\{\underset{\sim}{x}_a, \underset{\sim}{y}_a, \underset{\sim}{x}_b, \underset{\sim}{y}_b\}$ in $\Gamma$
// metrics
Metrics($G \cup \Gamma$)
// tournaments
for $k \leftarrow 0$ to $k < N_p$ do
    $A \leftarrow G[P_{k,0}]$
    $B \leftarrow G[P_{k,1}]$
    $a \leftarrow \Gamma[P_{k,0}]$
    $b \leftarrow \Gamma[P_{k,1}]$
    Tournament($G$, $A, B, a, b$)
```

(A,B) and new (a,b) solutions. The neighbour distance d_{ij} is employed to this task. The pair of competitors is then

$$\begin{matrix} \{A,\,a\} \text{ and } \{B,\,b\} & \text{if } d_{Aa} + d_{Bb} < d_{Ab} + d_{Ba} \\ \{A,\,b\} \text{ and } \{B,\,a\} & \text{otherwise} \end{matrix} \tag{10}$$

This strategy is essential to the crowding approach to niching [10] and allows for a new solution to compete against its closest match; if it wins, it will replace the old solution in set G; otherwise, the corresponding position in G will remain unchanged.

The `Tournament` function is given in Algorithm 6 and requires a function called `Fight` where two solutions will 'fight' each other. The improvement of diversity is effectively implemented in the `Fight` function. Although the comparison function `CompareSolutions` may return an indecisive response (non-dominance),

Algorithm 6. Tournament(Ω, A, B, a, b) performs the tournament with replacement in Ω.

```
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 $\Omega$
    if not Fight($B, b$) then // $b$ wins over $B$
        replace $B$ with $b$ in $\Omega$
else
    if not Fight($A, b$) then // $b$ wins over $A$
        replace $A$ with $b$ in $\Omega$
    if not Fight($B, a$) then // $a$ wins over $B$
        replace $B$ with $a$ in $\Omega$
```

Algorithm 7. Fight(A, B) performs the decisive selection between trial solutions.

```
Input: trial solutions $A$ and $B$
Output: returns true if $A$ wins
// compare solutions using OOR and OVA
$A_{dom}, B_{dom} \leftarrow$ CompareSolutions($A, B$)
if $A_{dom}$ then return true
if $B_{dom}$ then return false
// tie: single-OVA
if $N_f = 1$ then
    if $\eta_A > \eta_B$ then return true
    if $\eta_B > \eta_A$ then return false
    return FlipCoin(0.5)
// tie: multi-OVA: same Pareto front
if $\phi_A = \phi_B$ 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 $\phi_A < \phi_B$ then return true
if $\phi_B < \phi_A$ then return false
if $\eta_A > \eta_B$ then return true
if $\eta_B > \eta_A$ then return false
// tie: nothing else can be done
return FlipCoin(0.5)
```

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.

The `Fight` function is listed in Algorithm 7 where four additional branches are considered. The branches include the case when there is a tie between solutions A and B. This situation occurs, for instance, when A and B have the same OOR and OVA. Now, the analysis is different between single-objective and multi-objective problems. If there is a tie in a single-objective problem, the solution having the largest neighbour distance η wins; hence, inducing diversity. In multiple objectives problems, two further cases arise:

1. Both solutions are in the same Pareto front—in this case, the one with the largest crowding distance w wins; or
2. The solutions are located at different fronts—in this case, first, the one with the smallest Pareto front rank ϕ wins; second, the one with the largest neighbour distance η wins.

In the end, if the two solutions are equal one with another by comparing all metrics, the decision is random; i.e. by the flip of an unbiased coin. In summary, the proposed algorithm aims to induce diversity and seemed to work quite well.

5.2. Differential evolution

The differential evolution [43, 44] method is selected to combine real numbers. Other methods are available such as the SBX [39]; however, some experiments with the proposed code demonstrated that DE is more efficient. This finding has been partially observed by others as well [36].

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 candidate solution or a combination of the components of the other three auxiliary solutions. The combination is known as the mutation step [44] and is accomplished with

$$x_{\text{new},i} = x_{0,i} + F \cdot (x_{1,i} - x_{2,i}) \tag{11}$$

where F is a scaling factor that induces a displacement of the candidate 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 $\underset{\sim}{x}$ being modified or not.

The steps required for the differential evolution operator are collected in Algorithm 8. Note that, after the mutation step, some components of the solution vector may be outside the limits. In this situation, the values are simply truncated to the limiting values, thus automatically handling the *box constraints*.

6. Validation experiments

This section presents some experiments to study the capabilities and limitations of the proposed code. First, some single-objective optimisation problems with many constraints are studied. Then, unconstrained and constrained multi-objective problems are solved. Note that the limits of variables are also constraints, but not counted like so in the following discussion.

Attention is given to the repeatability of results. Therefore, each problem is run 1000 times (1000 samples), and the statistics of results are analysed. Each run generates one sample. After the execution of one sample is finished, all values are initialised, the population is randomly re-created, and the process is run all over again.

In these experiments, the number of candidate solutions N_{sol} is usually calculated as $10\,N_x$ (unless explicitly defined). Sometimes more solutions are needed and sometimes the problem could be easily solved with less. The number of groups (N_{cpu}) is selected to reduce the number of solutions in the same group to

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

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

be no more than 50. When needed, the exchange time Δt_{exc} is simply set equal to $\Delta t_{exc} = t_{max}/10$.

An important parameter is the differential evolution coefficient C_{DE}. By default, this value is selected as $C_{DE} = 0.8$. Later, the value is changed for some multi- and many-objective tests whenever the performance can be improved.

In the tables with results, the best reference values from the literature are listed as well. The best reference f values are shown as f_{ref} and the reference solutions x_i are listed below the solutions computed here.

In the tables, T_{sys}^{ave} is the average computer time to solve one sample in a 4-core Intel(R) i7-4770 CPU @ 3.40GHz (Debian-Sid/GNU/Linux). The 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.

6.1. Constrained one-objective problems

Nine constrained one-objective problems from [15] are analysed in this section. The first seven problems are somewhat "easy" whereas the last two pose some difficulties. Most problems are solved with $t_{max} = 500$ iterations, although the first ones could be solved with 100-200 iterations. The last three problems are solved with 1000, 5000 and 7000 iterations, respectively.

The last problem (P9) requires longer iterations and is a quite challenging one because it involves multiplications of the decision variable as $x_0\,x_1\,x_2\,x_3\,x_4$ within the exponential function. This means that the sign of pairs $x_i\,x_j$ $(i \neq j)$ can vary freely. Moreover, P9 has some nonlinear equality constraints and the space of feasible solution is "narrow" as defined by $\epsilon_h = 10^{-3}$.

The input data is given in Table 1 and the statistics of results are presented in Table 2. The solutions of each optimisation problem are collected in Table 3.

The code works quite well for problems 1 to 7 with the standard deviation f_{dev} being very small, i.e., every time the code is run, the same solution is obtained. In problems 1 to 7, minimum values (f_{min}) close to the reference ones are always obtained. Nonetheless, the reference f_{min} values of problems 8 and 9 are also found at least by one sample. Therefore, it is worth running an evolutionary algorithm more than once.

Table 1: Constrained single-objective problems. Input data.

P	N_{sol}	N_{cpu}	t_{max}	Δt_{exc}	C_{DE}	N_{eval}
1	20	1	500	1	0.8	10020
2	130	4	500	50	0.8	64130
3	50	1	500	1	0.8	25050
4	50	1	500	1	0.8	25050
5	40	1	500	1	0.8	20040
6	70	2	500	50	0.8	34070
7	100	2	1000	100	0.8	100100
8	80	2	5000	500	0.8	400080
9	50	1	7000	1	0.8	350050

Table 2: Constrained single-objective problems. Results.

P	T_{sys}^{ave}	f_{ref}	f_{min}	f_{ave}	f_{max}	f_{dev}	ref
1	20ms	13.59085	13.59084	13.59084	13.59084	$4.69 \cdot 10^{-10}$	[15]-T1
2	245ms	-15.00000	-14.99970	-14.99905	-14.99790	$2.68 \cdot 10^{-4}$	[15]-T3
3	159ms	-30665.50	-30665.54	-30665.54	-30665.54	$1.45 \cdot 10^{-12}$	[15]-T6
4	163ms	-1.90513	-1.90516	-1.90516	-1.90516	$3.51 \cdot 10^{-10}$	[15]-T2
5	88ms	2.34027	2.34021	2.34021	2.34021	$8.86 \cdot 10^{-10}$	[15]-T9
6	115ms	680.63006	680.63006	680.63032	680.63916	$6.250 \cdot 10^{-4}$	[15]-T5
7	537ms	24.30621	24.30646	24.30867	24.39394	$3.38 \cdot 10^{-3}$	[15]-T8
8	1.244s	7049.33092	7049.24802	7049.78763	7250.96733	9.01	[15]-T4
9	1.946s	0.05395	0.05395	0.27964	1.00000	$2.15 \cdot 10^{-1}$	[15]-T7

Table 3: Constrained single-objective problems. Solutions.

P	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}
1	2.24683	2.38186											
	2.24683	2.38186											
2	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
	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	78.00000	33.00000	29.99526	45.00000	36.77581								
	78.00000	33.00000	29.99500	45.00000	36.77600								
4	705.17454	68.60000	102.90000	282.32493	37.58412								
	705.18030	68.60005	102.90001	282.32500	37.58504								
5	0.25364	7.14155	7.10391	0.25364									
	0.25360	7.14100	7.10440	0.25360									
6	2.33059	1.95134	-0.47726	4.36574	-0.62435	1.03819	1.59432						
	2.33050	1.95137	-0.47754	4.36573	-0.62449	1.03813	1.59423						
7	2.17145	2.36510	8.77428	5.09673	0.99214	1.43342	1.32055	9.82782	8.28007	8.37951			
	2.17200	2.36368	8.77393	5.09598	0.99065	1.43057	1.32164	9.82873	8.28009	8.37593			
8	579.28692	1359.99484	5109.96626	182.01605	295.60135	217.98395	286.41470	395.60135					
	579.31670	1359.94300	5110.07100	182.01740	295.59850	217.97990	286.41620	395.59790					
9	-1.71720	1.59578	-1.82714	0.76349	-0.76379								
	-1.71714	1.59571	1.82725	-0.76364	-0.76365								

6.2. Unconstrained two-objective problems

In multi-objective optimisation problems, we aim to cover the Pareto front as much as we can while finding the optimal front. In this paper, with two objectives, we assess the performance of the evolutionary code with: (1) a measure of how well solution points are located near the optimal front—the error E; and (2) a measure of the spread of points along the optimal front—the spread L.

The first measure (error) is defined by the root mean square error between the numerical and an analytical solution as follows:

$$E = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}\left[f_1^i - f_1^{ana}(f_0^i)\right]^2} \tag{12}$$

where f_0^i is the numerical solution corresponding to $\underset{\cdot}{\boldsymbol{x}}_i$ and n is the number of solutions (points) that are feasible and non-dominated at the best front ($\phi_i = 0$). $f_1^{ana}(f_0)$ is the closed-form solution curve.

The second measure (spread) is calculated by summing the Euclidean distance of 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

$$L = \frac{1}{L_{ref}} \sum_{\substack{1\le j<n \\ 0\le i<n-1}} \sqrt{\left(f_0^j - f_0^i\right)^2 + \left(f_1^j - f_1^i\right)^2} \tag{13}$$

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 the points are.

The problems presented in [26] and a problem from [23, 3] are studied in this section. The first five problems are named ZDTi and the last one is named FON. These problems include concave and convex curves in the (f_0, f_1) space and situations of multiple local minima.

Except for FON, 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; however, ZDT4 would require more iterations due to the local fronts. The input data for all problems are list in Table 4.

Table 5 presents the results where we can observe 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 regarding accuracy (E) and spread (L).

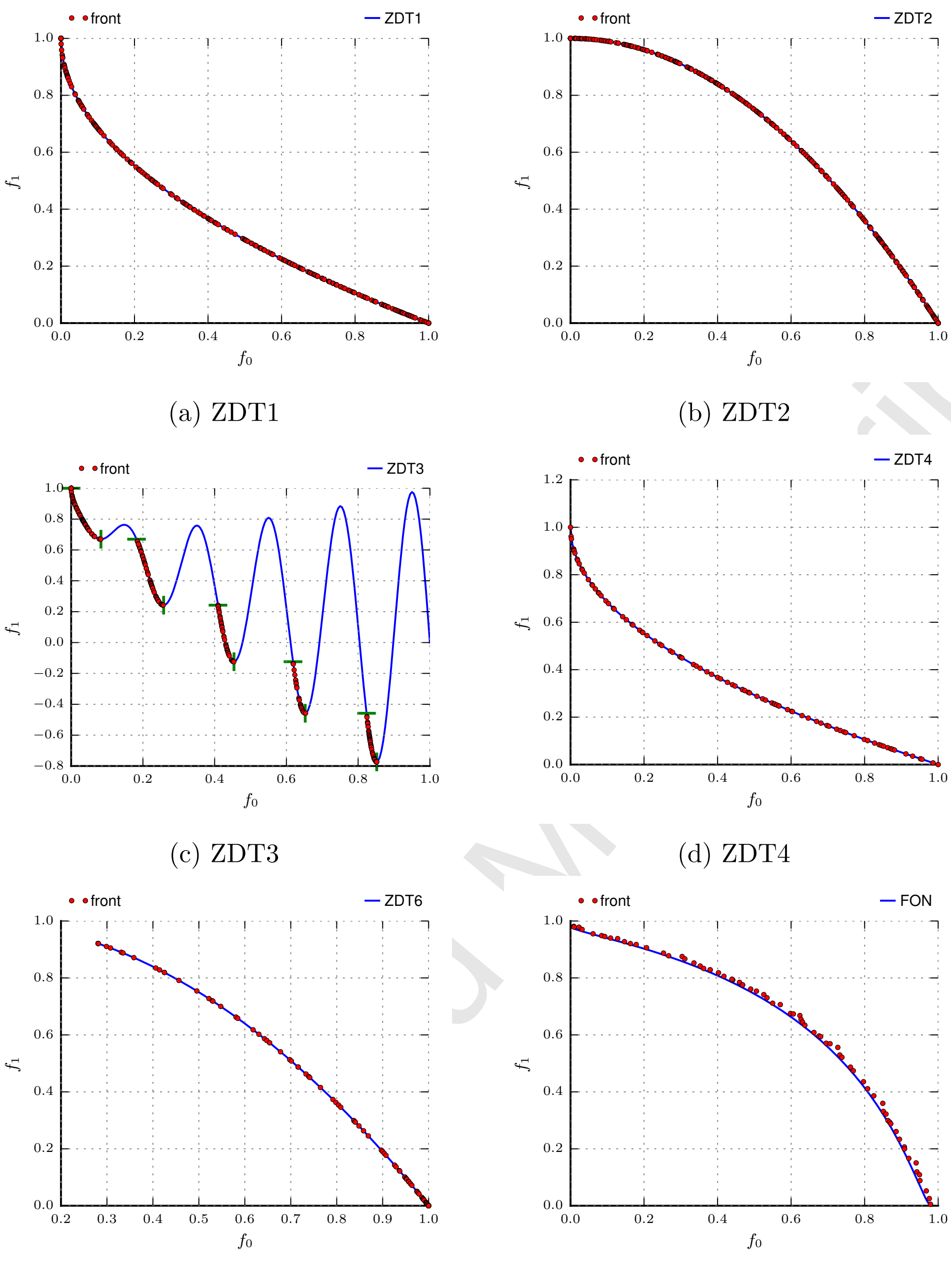
To further illustrate, plots of Pareto-optimal fronts for the six problems are presented in Fig. 3 corresponding to one (any) sample.

Table 4: Unconstrained two-objective problems. Input data.

P	N_{sol}	N_{cpu}	t_{max}	Δt_{exc}	C_{DE}	N_{eval}
ZDT1	300	6	500	50	0.1	150300
ZDT2	300	6	500	50	0.1	150300
ZDT3	300	6	500	50	0.1	150300
ZDT4	100	2	500	50	0.1	50100
FON	100	2	500	50	0.8	50100
ZDT6	100	2	500	50	0.1	50100

Table 5: Unconstrained two-objective problems. Results.

P	T_{sys}^{ave}	E_{min}	E_{ave}	E_{max}	E_{dev}	L_{dev}	L_{ave}	L_{max}	L_{dev}
ZDT1	885ms	0.0000	0.0000	0.0000	0.000	$9.9 \cdot 10^{-1}$	$9.9 \cdot 10^{-1}$	1.0000	$5.2 \cdot 10^{-6}$
ZDT2	915ms	0.0000	0.0000	0.0000	0.000	$9.9 \cdot 10^{-1}$	1.0000	1.0000	$9.1 \cdot 10^{-7}$
ZDT3	911ms	0.0000	0.0000	0.0000	0.000	$9.2 \cdot 10^{-1}$	$9.7 \cdot 10^{-1}$	1.4189	$1.8 \cdot 10^{-2}$
ZDT4	278ms	$5.3 \cdot 10^{-11}$	$1.8 \cdot 10^{-9}$	$2.8 \cdot 10^{-8}$	$2.6 \cdot 10^{-9}$	$9.7 \cdot 10^{-1}$	$9.9 \cdot 10^{-1}$	$9.9 \cdot 10^{-1}$	$9.2 \cdot 10^{-4}$
FON	265ms	$1.2 \cdot 10^{-2}$	$1.6 \cdot 10^{-2}$	$2.9 \cdot 10^{-2}$	$1.9 \cdot 10^{-3}$	1.0043	1.0371	1.2898	$2.0 \cdot 10^{-2}$
ZDT6	204ms	0.0000	0.0000	0.0000	0.000	$9.9 \cdot 10^{-1}$	$9.9 \cdot 10^{-1}$	1.0000	$1.4 \cdot 10^{-4}$



(a) ZDT1 (b) ZDT2

(c) ZDT3 (d) ZDT4

(e) ZDT6 (f) FON

Figure 3: Unconstrained two-objective problems. Numerical (dots) and analytical Pareto fronts.

6.3. Constrained two-objective problems

The series of constrained two-objective problems presented in [3] are studied in this section. 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. The input data are presented in Table 6.

The results are presented in Table 7 and illustrated in Figs. 4 and 5. We can observe that the accuracy is reasonable (small E) and the repeatability characteristics is good for all tests. The spread has been also observed in several runs—every time TNK or CTPi are run, Figs. 4 and 5 exhibit similar results.

Table 6: Constrained two-objective problems. Input data.

P	N_{sol}	N_{cpu}	t_{max}	Δt_{exc}	C_{DE}	N_{eval}
TNK	120	3	500	50	0.1	60120
CTP1	120	3	500	50	0.1	60120
CTP2	120	3	500	50	0.1	60120
CTP3	120	3	500	50	0.1	60120
CTP4	120	3	500	50	0.1	60120
CTP5	120	3	500	50	0.1	60120
CTP6	120	3	500	50	0.1	60120
CTP7	120	3	500	50	0.1	60120
CTP8	120	3	500	50	0.1	60120

Table 7: Constrained two-objective problems. Results.

P	T_{sys}^{ave}	E_{min}	E_{ave}	E_{max}	E_{dev}
TNK	119ms	$3.34 \cdot 10^{-3}$	$5.49 \cdot 10^{-3}$	$1.36 \cdot 10^{-2}$	$8.61 \cdot 10^{-4}$
CTP1	196ms	$6.94 \cdot 10^{-4}$	$1.50 \cdot 10^{-3}$	$6.06 \cdot 10^{-3}$	$3.82 \cdot 10^{-4}$
CTP2	219ms	$6.97 \cdot 10^{-4}$	$1.95 \cdot 10^{-3}$	$1.90 \cdot 10^{-2}$	$9.67 \cdot 10^{-4}$
CTP3	222ms	$1.69 \cdot 10^{-3}$	$5.13 \cdot 10^{-3}$	$2.26 \cdot 10^{-2}$	$1.72 \cdot 10^{-3}$
CTP4	234ms	$8.09 \cdot 10^{-3}$	$3.04 \cdot 10^{-2}$	$1.15 \cdot 10^{-1}$	$9.30 \cdot 10^{-3}$
CTP5	222ms	$1.96 \cdot 10^{-3}$	$5.19 \cdot 10^{-3}$	$2.34 \cdot 10^{-2}$	$1.58 \cdot 10^{-3}$
CTP6	232ms	$5.59 \cdot 10^{-2}$	$1.05 \cdot 10^{-1}$	$2.81 \cdot 10^{-1}$	$2.20 \cdot 10^{-2}$
CTP7	203ms	$4.08 \cdot 10^{-3}$	$5.70 \cdot 10^{-3}$	$1.38 \cdot 10^{-2}$	$1.75 \cdot 10^{-3}$
CTP8	233ms	$4.18 \cdot 10^{-2}$	$8.43 \cdot 10^{-2}$	$4.24 \cdot 10^{-1}$	$2.90 \cdot 10^{-2}$

6.4. Constrained and unconstrained three-objective problems

For problems with three or more objectives, the accuracy of the evolutionary algorithm is assessed by a scalar function $\psi(f_0, f_1, f_2, \ldots) = 0$ representing the optimal front. A root mean square error is then computed using

$$E = \sqrt{\frac{1}{n}\left[\sum_{i=0}^{n-1} \psi(f_0^i, f_1^i, f_2^i, \ldots)^2\right]} \tag{14}$$

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 $\phi_i = 0$.

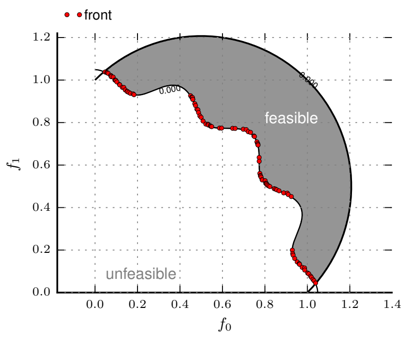
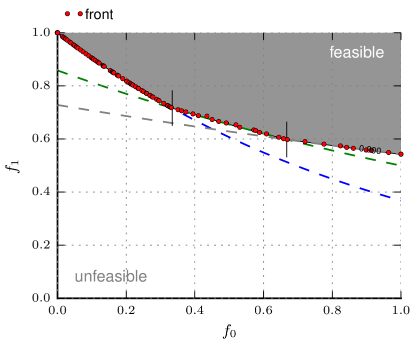
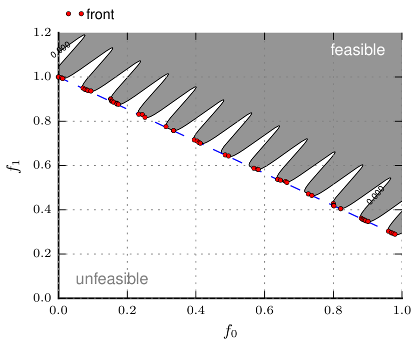


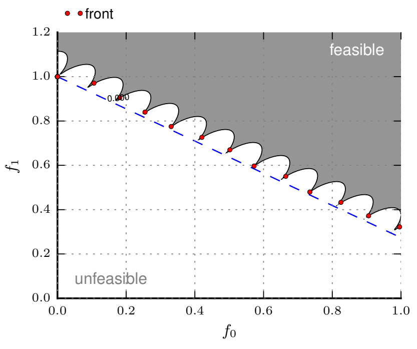
Figure 4: Constrained two objective test: TNK.



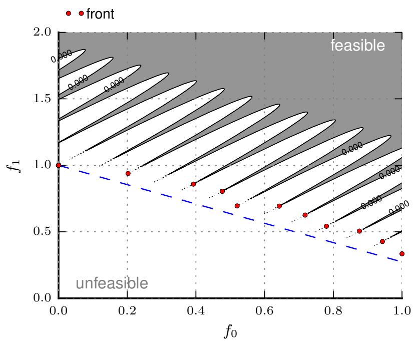
(a) CTP1



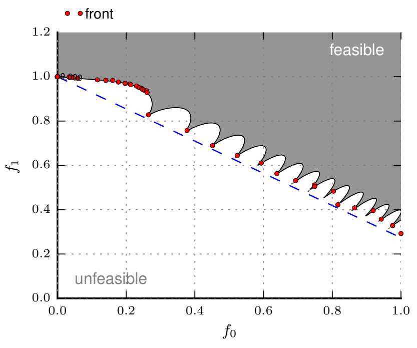
(b) CTP2



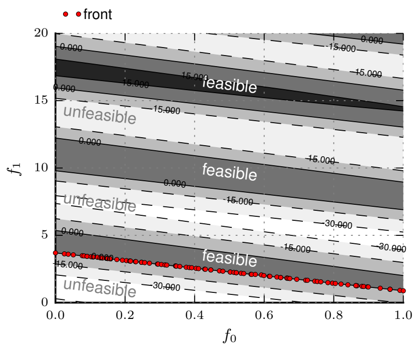
(c) CTP3



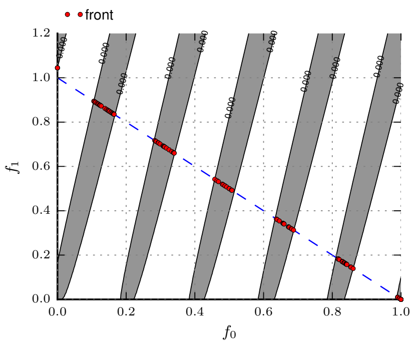
(d) CTP4



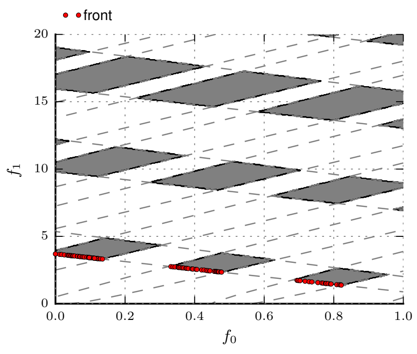
(e) CTP5



(f) CTP6



(g) CTP7



(h) CTP8

Figure 5: Constrained two-objective problems. Numerical (dots) Pareto front.

The tests known as DTZLi and discussed in [30] are studied. The convex version of DTZL2, named here as DTZL2x, and presented in [55, 56] is solved as well. Eight tests are unconstrained (except for the limits on $\boldsymbol{x}$). A new test (DTZL2c) is introduced with a constraint representing a cone aligned with the diagonal of the (f_0, f_1, f_2)-space. Thus, DTZL2c has the following constraint

$$g_0 = \tan\alpha - \frac{\sqrt{(f_0 - f_1)^2 + (f_1 - f_2)^2 + (f_2 - f_0)^2}}{f_0 + f_1 + f_2} \tag{15}$$

where α is equal to half of the cone's opening angle ($\alpha = 15$ is selected). In this way, the feasible solutions must lie on a circular patch of the optimal front.

Furthermore, a new family of tests is developed by changing DTLZi using the expression of a superquadric—the resulting tests are named as SUQi. The superquadric problems are defined by

$$\begin{aligned} c &= \textstyle\sum_{i=2}^{11}(x_i - 0.5)^2 \\ f_0 &= (1 + c)\ \mathrm{suqcos}\,(0.5x_0\pi; 2/a)\ \mathrm{suqcos}\,(0.5x_1\pi; 2/a) \\ f_1 &= (1 + c)\ \mathrm{suqcos}\,(0.5x_0\pi; 2/b)\ \mathrm{suqsin}\,(0.5x_1\pi; 2/b) \\ f_2 &= (1 + c)\ \mathrm{suqsin}\,(0.5x_0\pi; 2/c) \end{aligned} \tag{16}$$

where a, b and c are the coefficients of the superquadric and

$$\mathrm{suqsin}\,(w; m) = \mathrm{sign}\,(\sin w)\,|\sin w|^m \tag{17}$$

and

$$\mathrm{suqcos}\,(w; m) = \mathrm{sign}\,(\cos w)\,|\cos w|^m \tag{18}$$

Therefore, the function to compute the error corresponding the SUQi tests is

$$\psi(f_0, f_1, f_2) = |f_0|^a + |f_1|^b + |f_2|^c - 1 \tag{19}$$

The above expressions are quite versatile because they can produce concave, convex and many kinds of twisted surfaces in the space of objective values. Two tests are considered: (a) SUQ1 with $a = 0.5$, $b = 0.5$, and $c = 0.5$; and SUQ2 with $a = 2.0$, $b = 1.0$, and $c = 0.5$.

The results are collected in Table 9 where we can observe that the errors and standard deviations reached machine precision, except in DTLZ3 which is a little more difficult due to the local optima. With more iterations, the error is reduced. The constraint in DTLZ2c does not cause problems to the solver—we recall that the algorithm prioritises satisfaction of constraints and, hence, candidate solutions are "pulled" into the conical space.

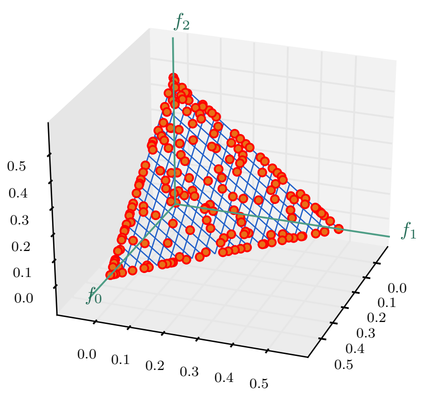
The results of one sample are illustrated in Fig. 6 where we observe that the candidate solutions (dots) are close to the optimal front indicated by the surface.

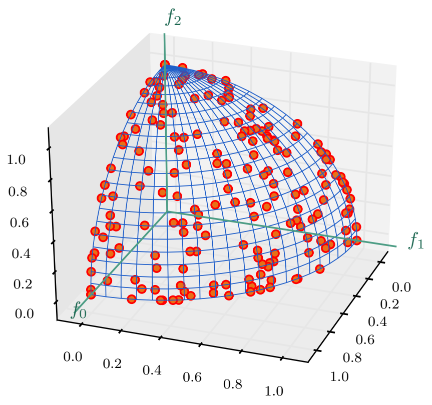
Table 8: Constrained and unconstrained three-objective problems. Input data.

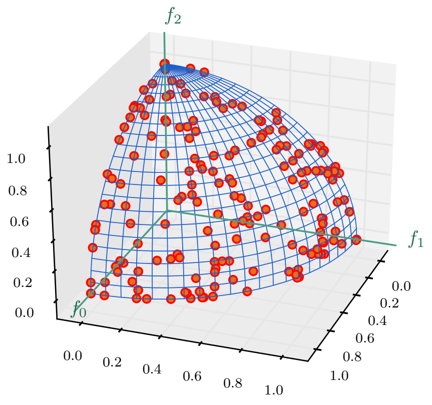
P	N_{sol}	N_{cpu}	t_{max}	Δt_{exc}	C_{DE}	N_{eval}
DTLZ1	200	5	500	50	0.01	100200
DTLZ2	200	5	500	50	0.01	100200
DTLZ3	200	5	500	50	0.01	100200
DTLZ4	200	5	500	50	0.01	100200
DTLZ2x	200	5	500	50	0.01	100200
DTLZ2c	200	5	500	50	0.01	100200
SUQ1	200	5	500	50	0.01	100200
SUQ2	200	5	500	50	0.01	100200

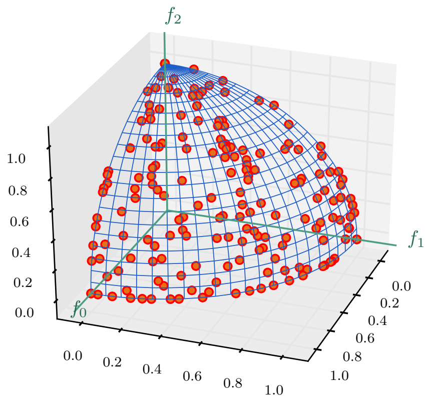
Table 9: Constrained and unconstrained three-objective problems. Results.

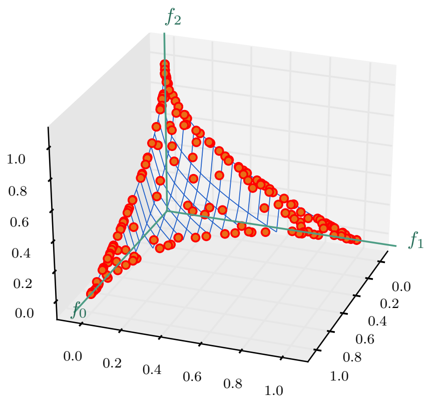
P	T_{sys}^{ave}	E_{min}	E_{ave}	E_{max}	E_{dev}
DTLZ1	249ms	0.0000	$1.75 \cdot 10^{-15}$	$2.48 \cdot 10^{-14}$	$1.97 \cdot 10^{-15}$
DTLZ2	362ms	$3.78 \cdot 10^{-16}$	$5.28 \cdot 10^{-16}$	$4.77 \cdot 10^{-14}$	$1.51 \cdot 10^{-15}$
DTLZ3	371ms	$4.98 \cdot 10^{-6}$	$9.66 \cdot 10^{-4}$	$9.62 \cdot 10^{-2}$	$3.59 \cdot 10^{-3}$
DTLZ4	358ms	$3.66 \cdot 10^{-16}$	$5.04 \cdot 10^{-16}$	$1.29 \cdot 10^{-14}$	$5.72 \cdot 10^{-16}$
DTLZ2x	351ms	$3.72 \cdot 10^{-16}$	$4.92 \cdot 10^{-16}$	$1.18 \cdot 10^{-14}$	$4.66 \cdot 10^{-16}$
DTLZ2c	352ms	$4.30 \cdot 10^{-16}$	$1.01 \cdot 10^{-15}$	$7.89 \cdot 10^{-14}$	$3.78 \cdot 10^{-15}$
SUQ1	352ms	$1.30 \cdot 10^{-16}$	$1.72 \cdot 10^{-16}$	$2.71 \cdot 10^{-15}$	$1.19 \cdot 10^{-16}$
SUQ2	350ms	$2.09 \cdot 10^{-16}$	$2.84 \cdot 10^{-16}$	$3.86 \cdot 10^{-15}$	$1.70 \cdot 10^{-16}$

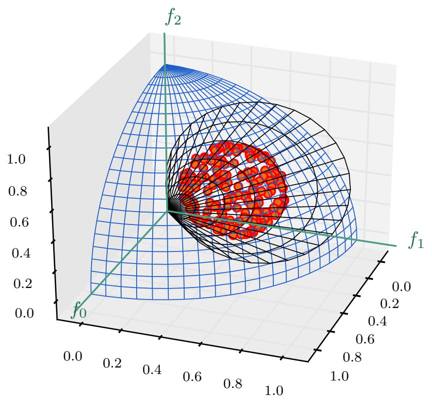

(a) DTLZ1

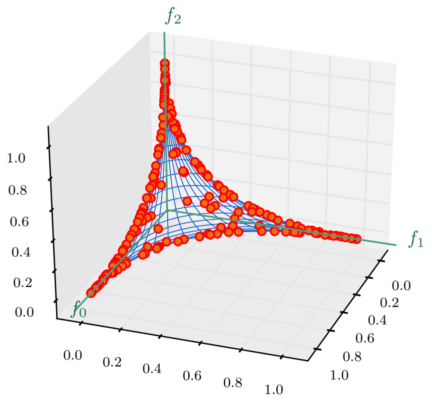

(b) DTLZ2


(c) DTLZ3


(d) DTLZ4


(e) DTLZ2x


(f) DTLZ2c


(g) SUQ1

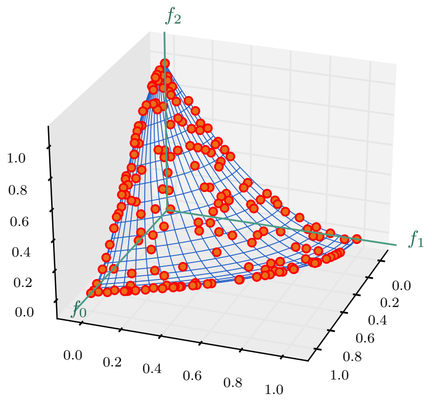

(h) SUQ2

Figure 6: Three-objective problems. Surfaces representing the optimal Pareto fronts and solutions (dots).

6.5. Unconstrained many-objectives problems

In this section, problem DTLZ2 is re-analysed with more than three objectives. The following numbers are selected: $N_f \in \{5, 7, 10, 15, 20\}$. The number of variables is also increased according to

$$N_x = N_f + 10 \tag{20}$$

Thus, $N_x \in \{15, 17, 20, 35\}$. It is clear that the problems with more objectives and variables are more difficult. The problems are identified as DTLZ2mI where I stands for the number of objectives. Table 10 collects the input data.

The results are presented in Table 11 showing that small errors are obtained every time the code is run. We observe, in particular, that the error increases in a nonlinear trend with the number of objective functions. This behaviour is illustrated in Fig. 7. In this figure, we can also observe that, despite keeping the number of solutions constant, the error rate decreases somewhat with the number of objectives.

Table 10: Unconstrained many objective problems. Input data.

P	N_{sol}	N_{cpu}	t_{max}	Δt_{exc}	C_{DE}	N_{eval}
DTLZ2m5	300	6	500	50	0.01	150300
DTLZ2m7	300	6	500	50	0.01	150300
DTLZ2m10	300	6	500	50	0.01	150300
DTLZ2m13	300	6	500	50	0.01	150300
DTLZ2m15	300	6	500	50	0.01	150300
DTLZ2m20	300	6	500	50	0.01	150300

Table 11: Unconstrained many objective problems. Results.

P	T_{sys}^{ave}	E_{min}	E_{ave}	E_{max}	E_{dev}
DTLZ2m5	946ms	$1.35 \cdot 10^{-13}$	$2.33 \cdot 10^{-12}$	$2.22 \cdot 10^{-10}$	$1.01 \cdot 10^{-11}$
DTLZ2m7	1.116s	$2.11 \cdot 10^{-11}$	$2.39 \cdot 10^{-10}$	$1.40 \cdot 10^{-8}$	$8.34 \cdot 10^{-10}$
DTLZ2m10	1.303s	$4.36 \cdot 10^{-9}$	$5.76 \cdot 10^{-8}$	$6.82 \cdot 10^{-6}$	$2.75 \cdot 10^{-7}$
DTLZ2m13	1.507s	$2.92 \cdot 10^{-7}$	$2.39 \cdot 10^{-6}$	$2.37 \cdot 10^{-4}$	$8.61 \cdot 10^{-6}$
DTLZ2m15	1.629s	$2.29 \cdot 10^{-6}$	$2.58 \cdot 10^{-5}$	$2.17 \cdot 10^{-3}$	$1.08 \cdot 10^{-4}$
DTLZ2m20	2.071s	$1.37 \cdot 10^{-4}$	$1.12 \cdot 10^{-3}$	$2.85 \cdot 10^{-2}$	$1.95 \cdot 10^{-3}$

7. Truss shape and topology optimisation

Topology optimisation of trusses (slender rods or bars) is an important problem in structural engineering. The problem is difficult when the cross-sectional areas of structural members have to be found at the same time as the connections between rods.

A two-objective version of this problem is considered here. The problem seeks to minimise: (1) the overall minimum weight (f_0); and (2) the deflection when the structure is under loads (f_1). Better satisfying one goal induces a failure in satisfying the other goal. Hence we seek a Pareto optimal front. The structure has to be stable as well; therefore limits on maximum stresses must be followed and some constraints must be satisfied. The equilibrium problem is solved by the Finite Element Method (FEM).

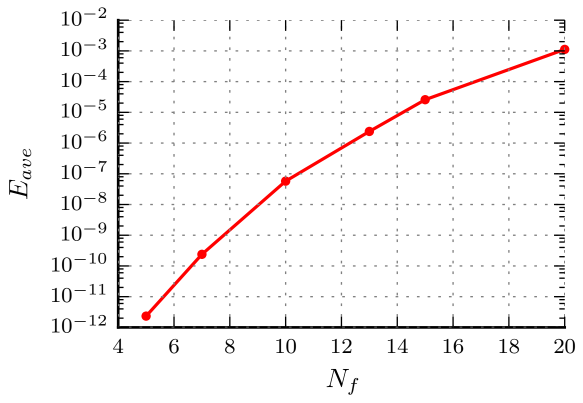


Figure 7: DTLZ2mI: Error behaviour with number of objective functions.

Furthermore, a minimum number of joints (nodes) and bars (elements) must be present at all times to allow the structure to carry the prescribed load. Also, even if the minimal number of joints is satisfied, the structure may become a mechanism (or mobile). In this situation, the FEM solver will fail due to a singular global stiffness matrix.

To reduce the number of unsuccessful calls to the FEM solver, a *mobility* number M can be estimated. The Chebychev-Grübler-Kutzbach criterion is employed to this task [57, 58] allowing the definition of M as

$$M = 2\,n - m - \chi \tag{21}$$

where n is the number of nodes, m is the number of bars, and χ is the number of fixed degrees of freedom. If M is positive, the above expression indicates that the structure is a mechanism.

As an additional complication, the expression for M may fail at times. For instance, M is incorrect when there are redundant bars at some locations, and some local mechanisms arise at other locations. Therefore, the above condition is not sufficient to prevent the FEM in computing a singular global stiffness matrix. Here, this problem is handled by using an additional out-of-range (OOR) variable (u_1 as explained shortly).

The problem representation is as follows. Real numbers are used for the cross-sectional areas (x_i) and integers are used for the topology description (connection of bars, y_i). In this case, integers simply serve as Boolean variables where 1 means a connection is active and 0 means that the connection is inactive.

The maximum deflection $\delta = |u_y^{max}|$ at a selected node is computed by the FEM. This value is limited by a maximum allowance δ_{awd}. The maximum tensile or compressive axial stress σ_{max} at any bar is also limited by σ_{awd}. The bar lengths and densities are L_i and ρ, respectively.

In summary, there are four constraints (OORs): (1) lack of mobility (u_0); (2) successful FEM run (u_1); (3) satisfaction of maximum deflection (u_2); and (4) satisfaction of maximum stresses (u_3).

The problem is modelled by Eq. (2) considering:

$$
\begin{aligned}
f_0 &= \sum_{i=0}^{N_{active}} \rho\, x_i\, L_i \\
f_1 &= \delta \\
u_0 &= \langle M \rangle \\
u_1 &= 1 \text{ if FEM failed; } 0 \text{ otherwise} \\
u_2 &= \langle \delta - \delta_{awd} \rangle \\
u_3 &= \langle |\sigma_{max}| - \sigma_{awd} \rangle
\end{aligned} \tag{22}
$$

where x_i are the cross-sectional areas (design variables), N_{active} is the number of active bars, and $\langle z \rangle$ is the ramp function defined as

$$
\langle z \rangle = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise} \end{cases} \tag{23}
$$

Note that, in Eq. (22), $f_i = f_i(\underset{\cdot}{\boldsymbol{x}}, \underset{\cdot}{\boldsymbol{y}})$ and $u_i = u_i(\underset{\cdot}{\boldsymbol{x}}, \underset{\cdot}{\boldsymbol{y}})$; i.e. all functions depend on the cross-sectional areas $(\underset{\cdot}{\boldsymbol{x}})$ and the connectivity $(\underset{\cdot}{\boldsymbol{y}})$.

The above expressions are implemented in Algorithm 9 where three intermediate exit points are considered: (1) assign the highest OOR values and exit if the required vertices are missing in $\underset{\cdot}{\boldsymbol{y}}$; (2) assign OOR values with the mobility coefficient and other nonzero values and exit if the structure is mobile; and (3) assign some nonzero OORs and exit if the FEM failed.

The first exit point sets u_0 with $1 + 2\,n$ which is a number greater than the maximum mobility M (when there are no bars and no fixities). At the same time, the other OORs all get 1.0. Therefore, the first exit condition defines a candidate that is worse than any another candidate that could at least be used to compute the mobility number.

In Algorithm 9, the OORs decrease as we approach a condition that the FEM is successful and the deflection and stresses can be computed. Therefore, the use of the OOR functions is quite convenient and effective. Generally, since the comparison between candidate solutions first considers the number of constraint violations N_{viol}, the OOR functions can always be set such that the worst case gets the highest N_{viol} value by assigning greater-than-zero u_i values; see Algorithm 1. Also, since the OORs are compared using the `ParetoComparison`, candidate solutions are effectively ranked even if the FEM solver is not called at all; e.g. when we know that the structure is mobile and the FEM will certainly fail.

7.1. Application

We study the algorithm with the example presented in [59, 3, 60–62] and illustrated in Fig. 8. The structure in this figure is known as *ground* structure because it serves as a basis for the generation of other combinations. In this particular structure, the indicated fixed nodes and the nodes with applied loads must always be present. In Fig. 8, $n = 6$, $m = 10$ and $\chi = 4$ because $\{u_x, u_y\}$ at $(0.0, 0.0)$ and $(0.0, 360.0)$ are fixed.

There are 10 bars (10 areas); thus the number of real numbers $\underset{\cdot}{\boldsymbol{x}}$ is $N_x = 10$ and the number of integers $\underset{\cdot}{\boldsymbol{y}}$ is $N_y = 10$. The Young's modulus of all bars is $E = 10^4$ [units of pressure], and the density is $\rho = 0.1$ [units of density]. The minimum and maximum areas are 0.09 and 35 [units of area], respectively, hence:

$$
0.09 \le x_i \le 35 \quad i \in [0, 9] \tag{24}
$$

Algorithm 9. `RunFEA`($\underset{\cdot}{x}, \underset{\cdot}{y}$) runs the finite element analysis and computes objective values (OVAs) and out-of-range values (OORs) for the topology optimisation problem.

```
Input: trial solution: Areas=$\underset{\cdot}{x}$, EnabledConnections=$\underset{\cdot}{y}$
Output: $f_i$ (OVAs) and $u_i$ (OORs)
compute $n$ = number of nodes
set connectivity based on $\underset{\cdot}{y}$
initialise FEA stage
check for required vertices
if not all required vertices are present then
    $u_0, u_1, u_2, u_3 \leftarrow 1.0 + 2.0\,n$, 1.0, 1.0, 1.0
    return
compute mobility $M$
if $M > 0$ then
    $u_0, u_1, u_2, u_3 \leftarrow M$, 1.0, 1.0, 1.0
    return
set cross-sectional areas of elements
compute total weight $W$
run FE analysis (FEA)
if FEA failed then
    $u_0, u_1, u_2, u_3 \leftarrow$ 0.0, 1.0, 1.0, 1.0
    return
find maximum deflection from FEA data
find maximum magnitude of stress from FEA data
$f_0, f_1 \leftarrow W, \delta$
$u_0, u_1, u_2, u_3 \leftarrow 0.0, 0.0, \langle \delta - \delta_{awd} \rangle, \langle |\sigma_{max}| - \sigma_{awd} \rangle$
```

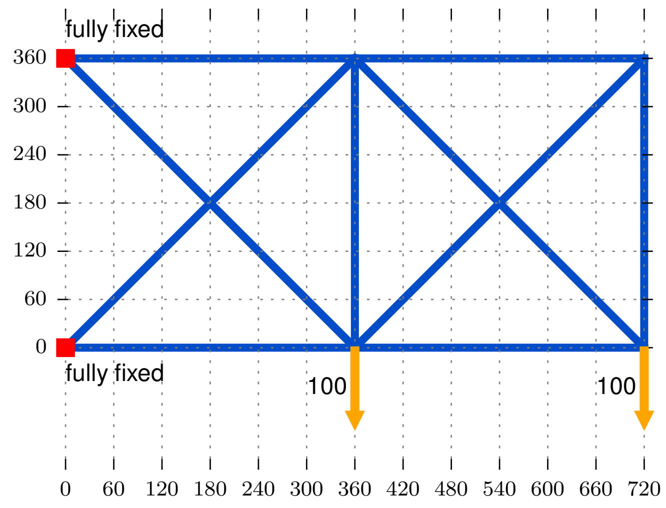


Figure 8: "Ground" mesh with all 10 rods active.

The maximum allowed deflection at $(720, 0)$ is $\delta_{awd} = 5.6$ [units of length] and the maximum allowed tensile or compressive axial stress at any bar is $\sigma_{awd} = 35$ [units of pressure].

As for the handling of integers in $\underset{\cdot}{y}$, simple crossover and mutation rules for integers are employed [1] with probabilities $P_c = 0.5$ and $P_m = 0.01$, respectively. The number of cuts during crossover is 1, and the number of bit changes during mutation is 1.

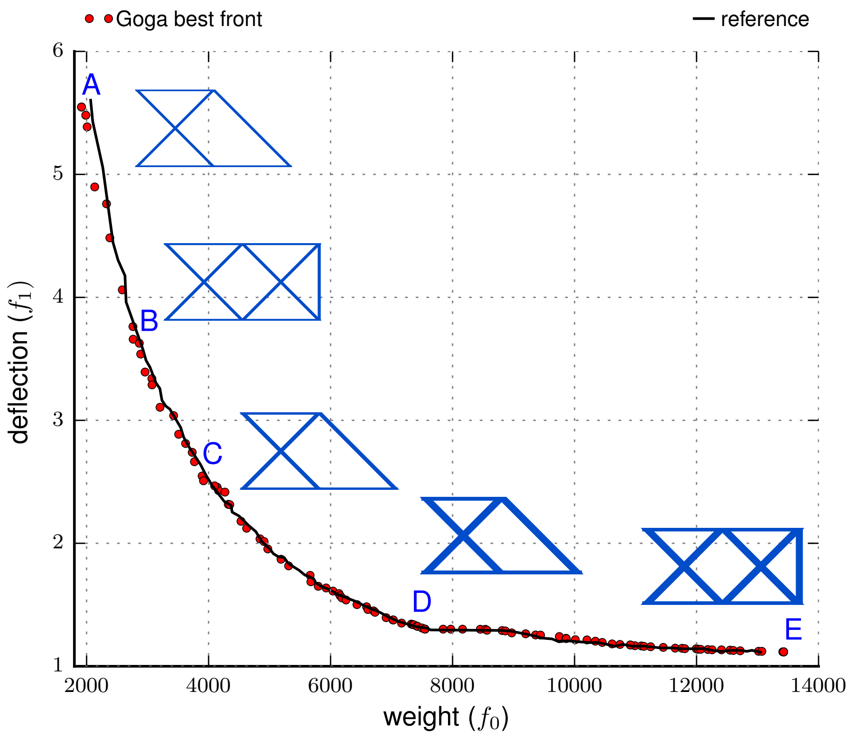


Figure 9: Shape and topology optimisation. Results.

The problem is solved with $N_{sol} = 200$, $N_{cpu} = 4$, $t_{max} = 500$, $\Delta t_{exc} = 50$, and $C_{DE} = 0.8$.

The results are illustrated in Fig. 9 where the optimal Pareto front obtained by Goga is represented by dots and a reference solution from [3] is represented by a continuous like. In this figure, some combinations of weight-deflection are highlighted with the characters ‘A’ to ‘F’ (keys). Near each key, a small sketch of the truss is also drawn where the thickness of bars is proportional to the corresponding optimal cross-sectional area.

The numeric results corresponding to each highlighted weight-deflection combination (key) are presented in Table 12. We have run Goga several times and the results always matched the reference solution fairly well.

The number of groups N_{cpu} is now varied from 1 to 16 to study the speedup behaviour. In this case, the code is run in a 16-core Intel(R) Xeon(R) CPU E5-2687W @ 3.10GHz (Debian-Sid/GNU/Linux). The speedup is then computed using

$$S_{up}(N_{cpu}) = \frac{T^1_{sys}}{T^{N_{cpu}}_{sys}} \tag{25}$$

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. 10 where we can observe a better-than-ideal speedup up to 14 CPUs. In the figure, the real computer time is also shown using a gray line with values given at the right-hand-side scale; the maximum time is around 40 seconds, for instance.

The excellent speedup behaviour is in part due to overcoming the non-optimised implementation of the `Metrics` routine, observing that this routine needs to perform several comparisons between candidate solutions in order to find the closest neighbours in addition to determine the Pareto front indices.

Table 12: Shape and topology optimisation. Results.

key	weight	deflection	A_0 / A_5	A_1 / A_6	A_2 / A_7	A_3 / A_8	A_4 / A_9
A	1916.57	5.548027	7.199095	5.218173	10.811224	0.090000	6.536961
			4.168413	35.000000	32.914802	9.640983	10.514602
B	2864.76	3.626781	12.216014	7.722140	14.233473	2.305935	13.535455
			5.771311	0.090000	0.090000	0.090000	12.582181
C	3894.11	2.546967	13.107062	13.527088	26.832486	18.856008	16.066290
			4.466947	14.912698	0.090000	1.181018	18.147772
D	7323.79	1.343003	34.357683	25.683676	35.000000	1.923554	35.000000
			6.648335	12.222177	0.090000	2.045773	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

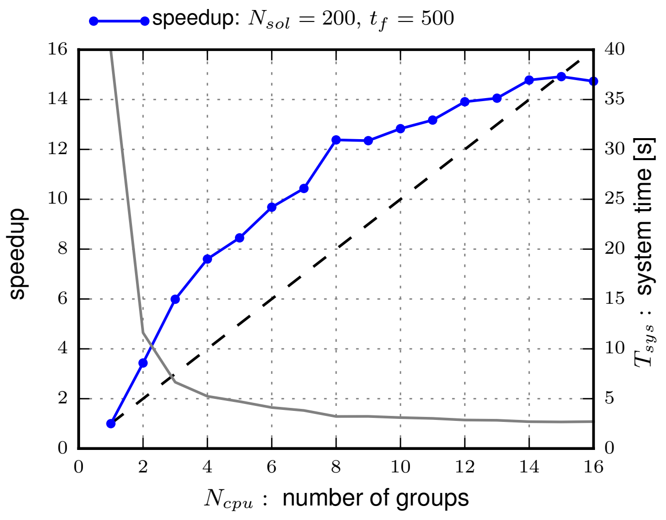


Figure 10: Shape and topology optimisation. Goga speedup behaviour.

8. Economic emission load dispatch

Economic emission load dispatch (or environmental/economic dispatch, EED) is a 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 opposing. Moreover, some constraints such as the maximum capacity of generators and the total power balance must be satisfied. The problem has been studied over several years, with some papers ranging from 1987 to 2015 [63–79].

In the EED problem, the fuel cost $f_0 = f_0(\underset{\cdot}{\boldsymbol{x}})$ of a thermal unit is approximated by the following quadratic function

$$f_0(\underset{\cdot}{\boldsymbol{x}}) = \sum_{i=0}^{N-1} \left(a_i + b_i\, P_i + c_i\, P_i^2\right) \tag{26}$$

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 $f_1 = f_1(\underset{\cdot}{\boldsymbol{x}})$ is also modelled as a function of the power output using

$$f_1(\underset{\cdot}{\boldsymbol{x}}) = \sum_{i=0}^{N-1} \left[\alpha_i + \beta_i\, P_i + \gamma_i\, P_i^2 + \zeta_i\, \exp\left(\lambda_i\, P_i\right)\right] \tag{27}$$

where α_i, β_i, γ_i, ζ_i and λ_i are fitting parameters.

The power output $x_i := P_i$ is limited by the generation capacity. Furthermore, an equality constraint $h_0(\underset{\cdot}{\boldsymbol{x}})$ must be taken into account 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

$$h_0(\underset{\cdot}{\boldsymbol{x}}) = \left(\sum_{i=0}^{N-1} P_i\right) - P_{demand} - P_{loss} \equiv 0 \tag{28}$$

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

$$P_{loss} = B_{00} + \sum_{i=0}^{N-1} B_{0i}\, P_i + \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} P_i\, B_{ij}\, P_j \tag{29}$$

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

The EED problem represented by Eqs. (26), (27), and (28) can be directly solved by Goga, considering the general problem in Eq. (2) and the suggested transformation of constraints in Eq. (4). No additional OORs are required as in the topology optimisation problem.

8.1. Application

The specific economic-emission dispatch of the *IEEE 30 Bus Test Case* is now studied. The circuit sketch is shown in Fig. 11 and is based on an earlier paper from 1987 [63]. The required fitting parameters are listed in Table 13 including the capacity limits. These values are based on data from [63] and [65] (note that

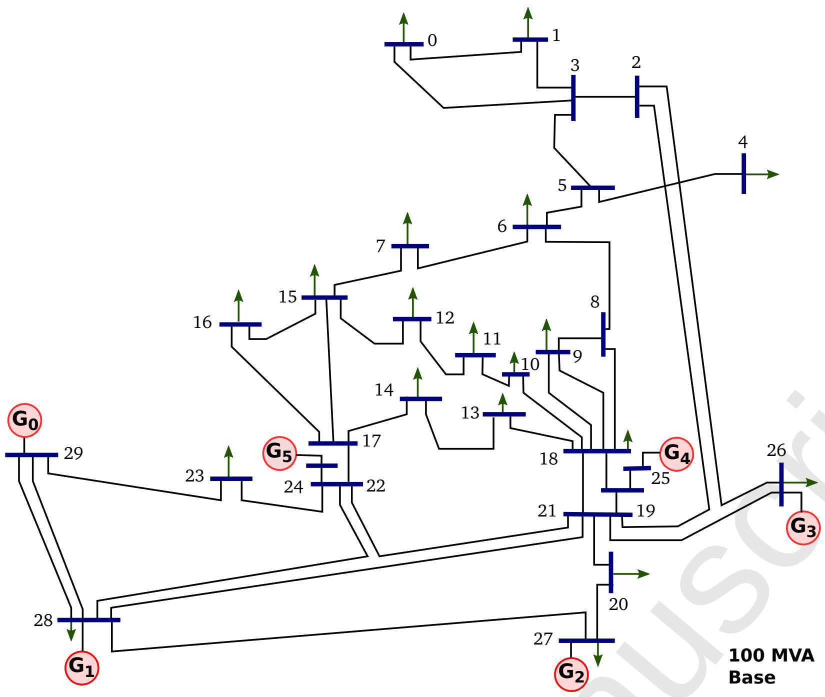


Figure 11: IEEE 30 bus test system.

Table 13: IEEE 30 bus: generators parameters.

G	cost			emission					output limits	
	a	b	c	$\alpha/10^{-2}$	$\beta/10^{-2}$	$\gamma/10^{-2}$	ζ	λ	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 14: 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

there is a typo error in [65] with regards to the α coefficient for G_4 in Table 1 of that paper).

The demand load considered in this EED problem is $P_{demand} = 2.834$ and the B coefficients for the IEEE 30 Bus Test case are collected in Table 14 [80–82].

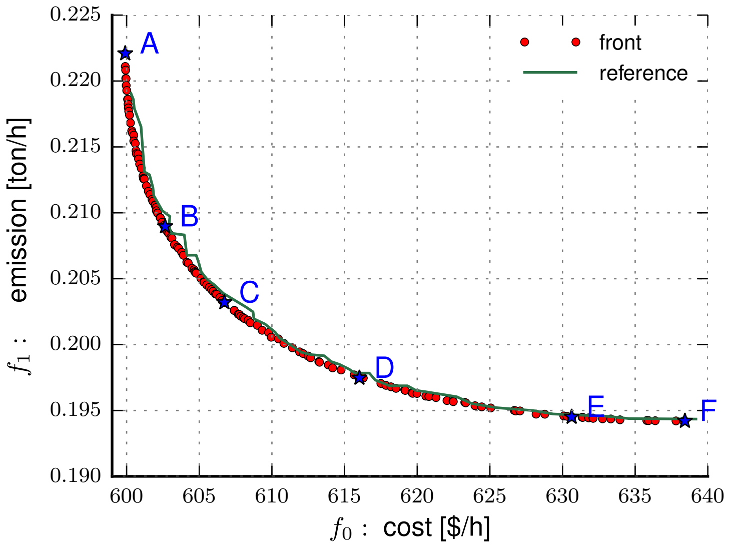


Figure 12: Economic emission load dispatch. Lossless.

Two cases are considered: (1) a lossless situation; and (2) a lossy situation. In the first case, P_{loss} is set to zero making the problem a little easier. In both cases, the solutions are 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. 12 where we observe that the results match well the reference solution from [66]. The optimal front for the second case (lossy) is illustrated in Fig. 13 where the reference solution from [66] is slightly less optimal than the optimal front computed here.

In Figs. 12 and 13, some cost-emission combinations are highlighted with the characters from 'A' to 'F' (keys). The key 'A' corresponds, for example, to the minimum emission whereas keys such as 'C' or 'D' to a compromise between cost and emission. The numeric results corresponding to each key are collected in Table 15 for the lossless case, and in Table 16 for the lossy case. The balance errors are also listed in these tables. We observe that the errors fall within the range specified by ϵ_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.

9. Conclusions

This paper presented an evolutionary algorithm (EA) using differential evolution (DE) capable of solving a range of optimisation problems, including some with many constraints or many objectives. New routines to compare candidate solutions, including a Pareto comparison of transformed constraints via the out-of-range (OOR) functions are proposed. The algorithm is designed to be used in multiple-core machines and be run in parallel. This is accomplished by splitting the space of candidate solutions into smaller groups, each one running in parallel until a time for exchange of solutions is reached. Two strategies for exchanging solutions are developed and combined: by tournament and randomly. These two strategies help with widening the search space in addition to making computations faster.

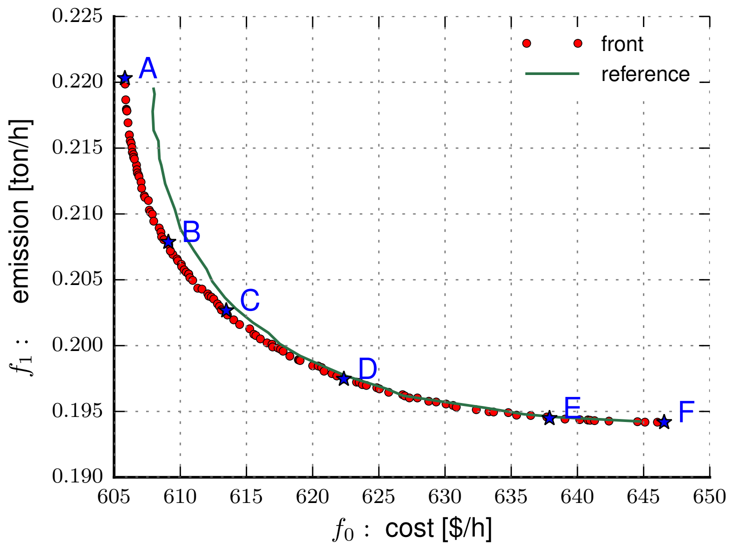


Figure 13: Economic emission load dispatch. Lossy case.

Table 15: Economic emission dispatch. Results: Lossless case.

point	cost	emission	h_0	P_0 P_3	P_1 P_4	P_2 P_5
A	599.9026	0.222090	$9.91 \cdot 10^{-4}$	0.105173	0.301731	0.519581
				1.012547	0.536716	0.357262
B	602.6466	0.208968	$8.43 \cdot 10^{-4}$	0.176092	0.339997	0.536580
				0.835182	0.560449	0.384856
C	606.7280	0.203199	$7.92 \cdot 10^{-4}$	0.226306	0.364127	0.537425
				0.740597	0.545566	0.419187
D	616.0211	0.197476	$7.09 \cdot 10^{-4}$	0.300036	0.396732	0.551286
				0.598921	0.535180	0.451135
E	630.6286	0.194512	$3.51 \cdot 10^{-4}$	0.376040	0.433171	0.536037
				0.450967	0.535909	0.502228
F	638.4239	0.194203	$6.24 \cdot 10^{-4}$	0.406427	0.457721	0.540046
				0.382793	0.536646	0.510990

Table 16: Economic emission dispatch. Results: Lossy case.

point	cost	emission	h_0	P_0 P_3	P_1 P_4	P_2 P_5
A	605.8149	0.220304	$9.27 \cdot 10^{-4}$	0.126417	0.292892	0.580853
				0.988574	0.529353	0.340504
B	609.0984	0.207868	$3.04 \cdot 10^{-4}$	0.202928	0.326906	0.577888
				0.825439	0.529474	0.396573
C	613.4727	0.202666	$3.96 \cdot 10^{-4}$	0.258756	0.363476	0.541359
				0.736984	0.564672	0.394775
D	622.3559	0.197466	$7.37 \cdot 10^{-4}$	0.292819	0.407606	0.540704
				0.600284	0.561182	0.458759
E	637.8918	0.194484	$2.62 \cdot 10^{-4}$	0.373063	0.454400	0.540847
				0.454534	0.548551	0.494973
F	646.5502	0.194181	$6.56 \cdot 10^{-4}$	0.412990	0.466577	0.540581
				0.387875	0.544981	0.515906

Due to the stochastic nature of EAs, the execution of any EA code should be repeated several times in order to assess its repeatability characteristics. Several tests are studied including optimisation problems with up to 20 objective functions. The repeatability of computations is demonstrated by observing a small standard deviation from several runs with the same problem but different initial candidate solutions. This characteristic is essential to obtain a level of reliability of the code. Evidence of the good characteristics of the proposed code including the ideal speedup property is also shown.

Several experiments have been carried out to find the best values for the differential evolution coefficient C_{DE}. For most problems, the value of 0.8 is ideal. Throughout the paper, the following values are used nonetheless: 0.8, 0.1, and 0.01. Apparently, for problems with more than one objective, smaller C_{DE} values make the resulting code to perform better.

Finally, two interesting and useful 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 or failures of the finite element solver. The second application involves an equality constraint (the power balance) that makes the problem a little more challenging. Nonetheless, the proposed code performs well in both applications and is able to match some reference results.

Acknowledgment

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

References

[1] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, 1st ed., Addison-Wesley, 1989.

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

[3] K. Deb, Multi-Objective Optimization using Evolutionary Algorithms, Wiley, 2001.

[4] K. C. Tan, E. F. Khor, T. H. Lee, Multiobjective Evolutionary Algorithms and Applications, Springer-Verlag London, 2005. doi:10.1007/1-84628-132-6.

[5] C. C. Coello, G. B. Lamont, D. A. van Veldhuizen, Evolutionary Algorithms for Solving Multi-Objective Problems, Springer, 2007. doi:10.1007/978-0-387-36797-2.

[6] D. E. Goldberg, J. Richardson, Genetic algorithms with sharing for multimodal function optimization, in: Proc. of the 2nd Int. Conf. on Genetic Algorithms and Their Application, L. Erlbaum Associates Inc., 1987, pp. 41–9.

[7] S. W. Mahfoud, Niching Methods for Genetic Algorithms, Ph.D. thesis, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 1995.

[8] C. H. Chen, T. K. Liu, J. H. Chou, A novel crowding genetic algorithm and its applications to manufacturing robots, IEEE Transactions on Industrial Informatics 10 (2014) 1705–16.

[9] S. M. Elsayed, R. a. Sarker, D. L. Essam, A new genetic algorithm for solving optimization problems, Engineering Applications of Artificial Intelligence 27 (2014) 57–69.

[10] O. J. Mengshoel, D. E. Goldberg, The crowding approach to niching in genetic algorithms, Evolutionary Computation 16 (2008) 315–54.

[11] O. J. Mengshoel, S. F. Galn, A. de Dios, Adaptive generalized crowding for genetic algorithms, Information Sciences 258 (2014) 140–59.

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

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

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

[15] K. Deb, An efficient constraint handling method for genetic algorithms, Computer Methods in Applied Mechanics and Engineering 186 (2000) 311–38.

[16] C. A. C. Coello, 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 191 (2002) 1245–87.

[17] K. Deb, A. R. Reddy, G. Singh, Optimal scheduling of casting sequence using genetic algorithms, Materials and Manufacturing Processes 18 (2003) 409–32.

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

[19] S. F. Galán, O. J. Mengshoel, Constraint handling using tournament selection: Abductive inference in partly deterministic bayesian networks, Evolutionary Computation 17 (2016) 55–88.

[20] Q. Long, A constraint handling technique for constrained multi-objective genetic algorithm, Swarm and Evolutionary Computation 15 (2014) 66–79.

[21] M. Hellwig, D. V. Arnold, Comparison of constraint-handling mechanisms for the (1,λ)-es on a simple constrained problem, Evolutionary Computation 24 (2016) 1–23.

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

[23] C. M. Fonseca, P. J. Fleming, An overview of evolutionary algorithms in multiobjective optimization, Evolutionary Computation 3 (1995) 1–16.

[24] C. M. Fonseca, P. J. Fleming, 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 28 (1998) 26–37.

[25] C. M. Fonseca, P. J. Fleming, Multiobjective optimization and multiple constraint handling with evolutionary algorithms. II. application example, IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans 28 (1998) 1–13.

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

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

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

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

[30] K. Deb, L. Thiele, M. Laumanns, E. Zitzler, Scalable test problems for evolutionary multiobjective optimization, in: A. Abraham, L. Jain, R. Goldberg (Eds.), Evolutionary Multiobjective Optimization, Springer, 2005, pp. 105–45. doi:10.1007/1-84628-137-7_6.

[31] S. Kukkonen, K. Deb, A fast and effective method for pruning of non-dominated solutions in many-objective problems, Parallel Problem Solving from Nature 4193 (2006) 553–62.

[32] Q. Zhang, H. Li, MOEA/D: A multiobjective evolutionary algorithm based on decomposition, IEEE Transactions on Evolutionary Computation 11 (2007) 712–31.

[33] K. Deb, S. Tiwari, Omni-optimizer: A generic evolutionary algorithm for single and multi-objective optimization, European Journal of Operational Research 185 (2008) 1062–87.

[34] S. Tiwari, P. Koch, G. Fadel, K. Deb, 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, ACM, New York, NY, USA, 2008, pp. 729–36. doi:10.1145/1389095.1389235.

[35] K. Deb, K. Miettinen, S. Chaudhuri, Toward an estimation of nadir objective vector using a hybrid of evolutionary and local search approaches, IEEE Transactions on Evolutionary Computation 14 (2010) 821–41.

[36] S. Tiwari, G. Fadel, K. Deb, AMGA2: improving the performance of the archive-based micro-genetic algorithm for multi-objective optimization, Engineering Optimization 43 (2011) 377–401.

[37] K. Li, . Fialho, S. Kwong, Q. Zhang, Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition, IEEE Transactions on Evolutionary Computation 18 (2014) 114–30.

[38] X. Qiu, J. X. Xu, K. C. Tan, H. A. Abbass, Adaptive cross-generation differential evolution operators for multiobjective optimization, IEEE Transactions on Evolutionary Computation 20 (2016) 232–44.

[39] K. Deb, R. B. Agrawal, Simulated binary crossover for continuous search space, Complex

Systems 9 (1995) 115–48.
[40] K. Deb, A. Kumar, Real-coded genetic algorithms with simulated binary crossover: Studies on multimodal and multiobjective problems, Complex Systems 9 (1995) 431–54.
[41] F. Herrera, M. Lozano, J. Verdegay, Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis, Artificial Intelligence Review 12 (1998) 265–319.
[42] D. M. Pedroso, D. J. Williams, Automatic calibration of soil–water characteristic curves using genetic algorithms, Computers and Geotechnics 38 (2011) 330–40.
[43] R. Storn, K. V. Price, Differential evolution–a simple and efficient adaptive scheme for global optimization over continuous spaces, Technical Report TR-95-012, ICSI (1995).
[44] K. Price, R. M. Storn, J. A. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, Springer-Verlag, 2005. doi:10.1007/3-540-31306-0.
[45] S. Kukkonen, J. Lampinen, GDE3: the third evolution step of generalized differential evolution, in: The 2005 IEEE Congress on Evolutionary Computation, volume 1, 2005, pp. 443–50. doi:10.1109/CEC.2005.1554717.
[46] S. Kukkonen, J. Lampinen, An empirical study of control parameters for the third version of generalized differential evolution (GDE3), in: IEEE Int. Conf. on Evolutionary Computation, 2006, pp. 2002–9. doi:10.1109/CEC.2006.1688553.
[47] M. R. Bonyadi, Z. Michalewicz, On the edge of feasibility: A case study of the particle swarm optimizer, in: 2014 IEEE Congress on Evolutionary Computation (CEC), 2014, pp. 3059–66. doi:10.1109/CEC.2014.6900343.
[48] M. R. Bonyadi, X. Li, Z. Michalewicz, A hybrid particle swarm with a time-adaptive topology for constrained optimization, Swarm and Evolutionary Computation 18 (2014) 22–37.
[49] M. R. Bonyadi, Z. Michalewicz, Locating Potentially Disjoint Feasible Regions of a Search Space with a Particle Swarm Optimizer, Springer India, New Delhi, 2015, pp. 205–30. doi:10.1007/978-81-322-2184-5_8.
[50] T. Ray, K. Tai, K. C. Seow, Multiobjective design optimization by an evolutionary algorithm, Engineering Optimization 33 (2001) 399–424.
[51] T. Ray, H. K. Singh, A. Isaacs, W. Smith, Constraint-Handling in Evolutionary Optimization, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 145–65. doi:10.1007/978-3-642-00619-7_7.
[52] T. P. Runarsson, X. Yao, Stochastic ranking for constrained evolutionary optimization, IEEE Transactions on Evolutionary Computation 4 (2000) 284–94.
[53] A. Isaacs, T. Ray, W. Smith, 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, pp. 2780–7. doi:10.1109/CEC.2008.4631171.
[54] R. Datta, K. Deb, Individual penalty based constraint handling using a hybrid bi-objective and penalty function approach, in: 2013 IEEE Congress on Evolutionary Computation, 2013, pp. 2720–7. doi:10.1109/CEC.2013.6557898.
[55] K. Deb, H. Jain, 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 18 (2014) 577–601.
[56] H. Jain, K. Deb, 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 18 (2014) 602–22.
[57] G. Gogu, Mobility of mechanisms: a critical review, Mechanism and Machine Theory 40 (2005) 1068 –97.
[58] J.-P. Li, Truss topology optimization using an improved species-conserving genetic algorithm, Engineering Optimization 47 (2015) 107–28.
[59] W. S. Ruy, Y. S. Yang, G. H. Kim, Y. S. Yeun, Topology design of truss structures in a multicriteria environment, Computer-Aided Civil and Infrastructure Engineering 16 (2001) 246–58.
[60] C.-Y. Wu, K.-Y. Tseng, Truss structure optimization using adaptive multi-population differential evolution, Structural and Multidisciplinary Optimization 42 (2010) 575–90.
[61] N. Noilublao, S. Bureerat, Simultaneous topology, shape, and sizing optimisation of plane trusses with adaptive ground finite elements using MOEAs, Mathematical Problems in Engineering 2013 (2013) 1–9.
[62] R. Cazacu, L. Grama, Steel truss optimization using genetic algorithms and fea, Procedia Technology 12 (2014) 339–46.
[63] R. Yokoyama, S. H. Bae, T. Morita, H. Sasaki, Multiobjective optimal generation dispatch based on probability security criteria., IEEE Transactions on Power Systems 3 (1987) 317–24.

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

[65] M. a. Abido, A niched pareto genetic algorithm for multiobjective environmental/economic dispatch, Int. Journal of Electrical Power and Energy Systems 25 (2003) 97–105.

[66] M. Abido, Multiobjective evolutionary algorithms for electric power dispatch problem, IEEE Transactions on Evolutionary Computation 10 (2006) 315–29.

[67] M. AlRashidi, M. El-hawary, Economic dispatch with environmental considerations using particle swarm optimization, Large Engineering Systems Conf. on Power Engineering (2006) 41–6.

[68] S. Kumar, R. Naresh, Nonconvex economic load dispatch using an efficient real-coded genetic algorithm, Applied Soft Computing 9 (2009) 321–9.

[69] S. Dhanalakshmi, S. Kannan, K. Mahadevan, S. Baskar, Application of modified nsga-ii algorithm to combined economic and emission dispatch problem, Int. Journal of Electrical Power and Energy Systems 33 (2011) 992–1002.

[70] L. Bayón, J. M. Grau, M. M. Ruiz, P. M. Suárez, The exact solution of the environmental / economic dispatch problem, IEEE Transactions on Power Systems 27 (2012) 723–31.

[71] N. A. Rahmat, I. Musirin, A. F. Abidin, 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) 198–202.

[72] Y.-F. Li, N. Pedroni, E. Zio, A memetic evolutionary multi-objective optimization method for environmental power unit commitment, IEEE Transactions on Power Systems 28 (2013) 2660–9.

[73] A. Jubril, O. Komolafe, K. Alawode, Solving multi-objective economic dispatch problem via semidefinite programming, IEEE Transactions on Power Systems 28 (2013) 2056–64.

[74] H. Bilil, G. Aniba, M. Maaroufi, Probabilistic economic/environmental power dispatch of power system integrating renewable energy sources, Sustainable Energy Technologies and Assessments 8 (2014) 181–90.

[75] B. Jeddi, V. Vahidinasab, A modified harmony search method for environmental/economic load dispatch of real-world power systems, Energy Conversion and Management 78 (2014) 661–75.

[76] A. Jubril, O. Olaniyan, O. Komolafe, P. Ogunbona, Economic-emission dispatch problem: A semi-definite programming approach, Applied Energy 134 (2014) 446–55.

[77] S. Sayah, A. Hamouda, A. Bekrar, Efficient hybrid optimization approach for emission constrained economic dispatch with nonsmooth cost curves, Int. Journal of Electrical Power & Energy Systems 56 (2014) 127–39.

[78] K. Bhattacharjee, A. Bhattacharya, S. Halder, Electrical power and energy systems backtracking search optimization based economic environmental power dispatch problems, Int. Journal of Electrical Power and Energy Systems 73 (2015) 830–42.

[79] H. Zhang, D. Yue, X. Xie, S. Hu, S. Weng, Multi-elite guide hybrid differential evolution with simulated annealing technique for dynamic economic emission dispatch, Applied Soft Computing 34 (2015) 312–23.

[80] L. Wang, C. Singh, Balancing risk and cost in fuzzy economic dispatch including wind power penetration based on particle swarm optimization, Electric Power Systems Research 78 (2008) 1361–8.

[81] S. Özyön, H. Temurta, B. Durmu, G. Kuvat, Charged system search algorithm for emission constrained economic power dispatch problem, Energy 46 (2012) 420–30.

[82] B. Pal, P. Biswas, A. Mukhopadhyay, 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, volume 283, Springer, 2012, pp. 156–69. doi:10.1007/978-3-642-28926-2_17.