Parallel Strategies for Direct Multisearch

S. Tavares[a], C. P. Brás[b], A. L. Custódio[b]*, V. Duarte[c], and P. Medeiros[c]

[a] FCT NOVA; [b]Department of Mathematics, FCT NOVA – CMA; [c] Department of Computer Science, FCT NOVA – NOVA LINCS. Campus de Caparica, 2829-516 Caparica, Portugal.

ARTICLE HISTORY
Compiled June 20, 2022

ABSTRACT

Direct Multisearch (DMS) is a Derivative-free Optimization class of algorithms, suited for computing approximations to the complete Pareto front of a given Multiobjective Optimization problem. In DMS class, constraints are addressed with an extreme barrier approach, only evaluating feasible points. It has a well-supported convergence analysis and simple implementations present a good numerical performance, both in academic test sets and in real applications. Recently, this numerical performance was improved with the definition of a search step based on the minimization of quadratic polynomial models, corresponding to the algorithm BoostDMS.

In this work, we propose and numerically evaluate strategies to improve the performance of BoostDMS, mainly through parallelization applied to the search and to the poll steps. The final parallelized version not only considerably decreases the computational time required for solving a Multiobjective Optimization problem, but also increases the quality of the computed approximation to the Pareto front. Extensive numerical results will be reported in an academic test set and in a chemical engineering application.

KEYWORDS
Multiobjective optimization; derivative-free optimization; direct search methods; parallel algorithms.

AMS CLASSIFICATION
90C29, 90C56, 68W10, 90C30.

1. Introduction

Direct MultiSearch (DMS) [16] is a well-established class of Multiobjective Derivative-free Optimization algorithms, which has been successfully used in real applications from very different scientific areas. Optimization of composite structures [32], economic problems [23], or data science applications [29] are just a few examples of the practical value of DMS. Additionally, the corresponding solver has been considered in benchmark studies, as a reference code for new algorithms [9, 13, 30].

When the algorithmic class of Direct Multisearch was originally proposed, the goal was to generalize directional direct search [5, 14] to Multiobjective Optimization. Sur-

*Corresponding author. Email: `alcustodio@fct.unl.pt`.

prisingly, the first implementation developed, which did not even comprise a defined search step, showed to be competitive against other state-of-art solvers like evolutionary strategies [18], simulated annealing [8], or direct search algorithms [6]. Recently, exceptional good results were also obtained when applying the solver to derivative-based optimization [1].

However, since its first version, in 2010, new releases corresponded to small improvements of the original implementation, with no major changes. Exception occurred in [11], where the definition of a search step based on the minimization of quadratic polynomial models was proposed. This new version of the code was named as BoostDMS.

This paper considers the use of BoostDMS to solve the Multiobjective Derivative-free Optimization problem

$$\begin{array}{ll} \min\limits_{x \in \mathbb{R}^n} & F(x) \equiv (f_1(x), \ldots, f_q(x))^\top \\ \text{s.t.} & lb \leq x \leq ub, \end{array} \tag{1}$$

where $q \geq 2$, $lb < ub$ represent bounds on the problem variables, and each $f_i : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}, i = 1, 2, \ldots, q$ denotes a component of the objective function, for which derivatives are not available, neither can be numerically approximated. BoostDMS addresses constraints with an extreme barrier approach, only evaluating feasible points. This is a simple strategy that confers to the algorithm the ability to treat other types of constraints, like unrelaxable, non-quantifiable, and hidden constraints [5].

When the different components of the objective function are conflicting between each other, it is impossible to find a point that minimizes every function component. Thus, rather than a single point, the solution of a Multiobjective Optimization problem is a set of points, for which it is not possible to improve the value of one component of the objective function without deteriorating the value of at least another. Points with these characteristics are said to be *efficient points* and the corresponding images form the *Pareto front* of the Multiobjective Optimization problem.

Since we are addressing Derivative-free Multiobjective Optimization problems, like in the single objective case, the computational time required for computing a solution is usually large, due to the expensive process of function evaluation. In fact, since the problem solution is no longer a point but a set of points, there is a strong need for strategies that allow to reduce the computational time without deteriorating the quality of the solution computed.

Distributed computing with multiple processors is often considered in the context of single objective optimization, for solving computationally quite demanding problems. Parallelization of Derivative-free Optimization methods for single objective optimization, particularly direct search ones, has been largely studied in the literature. Dennis and Torczon [20] described a first parallel version of a pattern search method which evaluates the function in parallel and synchronizes at each iteration to compare function values and make updates. A first asynchronous parallel version of pattern search was proposed in [28], dynamically initiating actions in response to events. Asynchronous approaches were also followed in [4], where processes solve problems over subsets of variables. Parallel approaches were also adopted for global single objective direct search algorithms [26, 36], often hybridizing metaheuristics with local direct search methods [25, 35].

In Multiobjective Optimization, parallelization appears typically associated to metaheuristics, like is the case of evolutionary strategies [10, 12, 19] or particle swarm approaches [22, 37]. A good overview of different parallel models for Multiobjective Optimization can be found in [33]. However, to the best of our knowledge, parallel approaches have not yet been considered for Multiobjective Directional Direct Search algorithms. Thus, the main goal of this paper will be to develop a parallel version of BoostDMS, which allows a considerable decrease of the computational time required for solving a problem, without deteriorating the quality of the computed approximation to the Pareto front.

The paper is organized as follows. Section 2 revises the algorithmic structure of the BoostDMS algorithm, pointing out possible steps for parallelization. Since all the proposed strategies are going to be numerically evaluated, Section 3 introduces tools and metrics that allow an assessment of the corresponding performance. Section 4 details and evaluates the different strategies considered. Numerical experiments on a chemical engineering problem are reported in Section 5, enhancing the benefits of the selected parallel strategy. Finally, conclusions are drawn in Section 6.

2. BoostDMS implementation of Direct Multisearch

BoostDMS [11] is a numerical implementation of a Direct Multisearch [16] algorithm, where the individual or joint minimization of quadratic polynomial models, built for each component of the objective function, is used in the definition of a search step.

The algorithm initializes with a list, L_0, of feasible, nondominated points, corresponding function values and stepsize parameters ($(x, F(x), \alpha_x)$, respectively). In BoostDMS, the initial stepsize is equal to one and, following the initialization proposed in [16], n points are evenly spaced in a line joining the problem bounds. Additionally, the centroid of the feasible region is always considered as initialization. This list represents the current approximation to the Pareto front of the problem and is updated every time that a new feasible nondominated point is found, corresponding to a successful iteration. The new nondominated point is added to the list and all the dominated points are removed from it. Using the strict partial order induced by the cone $\mathbb{R}^q_+$, a point x is said to dominate a point y if $F(x) \prec_F F(y)$, i.e., when $F(y) - F(x) \in \mathbb{R}^q_+ \setminus \{0\}$.

Each iteration of the algorithm starts by selecting an iterate point, $(x_k, F(x_k), \alpha_k)$, from the current list of feasible nondominated points, L_k, corresponding to the largest gap in the current approximation to the Pareto front, measured using the Γ spread metric:

$$\Gamma = \max_{i \in \{1,\ldots,q\}} \left(\max_{j \in \{1,\ldots,N-1\}} \{\delta_{i,j}\} \right). \tag{2}$$

Here $\delta_{i,j} = f_i(y_{j+1}) - f_i(y_j)$, assuming that for each component i of the objective function, $y_1, \ldots, y_N$ are the current points in the list L_k, sorted by increasing order of the corresponding objective function component value. Ties are broken by considering the point with the largest stepsize parameter. This iterate point will be used at the search step, as center of the quadratic polynomial models that will be built for each component, f_i, of the objective function. In case of failure of the search step in computing a new feasible nondominated point, the poll step will be performed, with the iterate point as poll center.

The search step reuses previously feasible evaluated points (not necessarily nondominated), kept in a cache, thus with no additional cost in terms of objective function values required for model computation. Points are selected in a neighborhood of the iterate point x_k, i.e., in the closed ball $B(x_k; \Delta_k)$, centered at x_k, with radius Δ_k, where Δ_k is proportional to the stepsize parameter α_k. In the current implementation of BoostDMS, 80% of the points are selected in $B(x_k; 3\alpha_k)$ and the remaining 20% are the ones closest to the current iterate point x_k, but lying outside this ball, conferring diversity to the quadratic models.

Let m_i define the quadratic polynomial model centered at x_k, corresponding to the objective function component f_i,

$$m_i(x) = f_i(x_k) + {g_k^i}^\top (x - x_k) + \frac{1}{2}(x - x_k)^\top H_k^i (x - x_k),\ i = 1, 2, \ldots, q \tag{3}$$

where the gradient vector g_k^i and the symmetric Hessian matrix H_k^i are computed by solving the linear system corresponding to the interpolation conditions

$$m_i(y_j) = f_i(y_j),\ j = 1, \ldots, p, \tag{4}$$

with y_j representing a feasible point for which the objective function has been previously evaluated, and p denoting the number of points available for computation.

Models are built once that $n + 2$ points have been selected, using a maximum of $(n + 1)(n + 2)$ points. Depending on the number of points available, minimum Frobenius norm models, determined interpolation models or regression approaches can be considered [14].

These models are then minimized individually inside $B(x_k; \Delta_k)$, in an attempt to improve the ability of the algorithm in generating approximations to the extreme points of the Pareto front (an extreme point corresponds to an individual extremum of a component of the objective function). However, the joint minimization of models is also considered, using a weighted Chebyshev norm scalarization, by solving the following problem:

$$\begin{array}{rl} \min\limits_{(x,\zeta)\in\mathbb{R}^{n+1}} & \zeta \\ \text{s.t.} & m_i(x) \ \leq \ \zeta, \quad i \in I \\ & \|x - x_k\| \ \leq \ \Delta_k \\ & lb \leq x \leq ub, \end{array} \tag{5}$$

where $I \subseteq \{1, 2, \ldots, q\}$. For each cardinality $1 \leq l \leq q$ of a subset I there are C_l^q (l combinations of q elements) possibilities to jointly minimize the models corresponding to the different components of the objective function. The algorithm starts by the individual minimization of each model ($l = 1$). If it fails in finding a new feasible nondominated point for the objective function, combinations of two models are considered ($l = 2$). The process is iteratively repeated, increasing the level l of model combinations until the joint minimization of all models is considered ($l = q$). If no new feasible nondominated point is found for the objective function, the poll step will be performed.

The poll step corresponds to a local search around the selected iterate point (the poll center), testing a set of directions with an adequate geometry, scaled by the stepsize parameter. Typically, positive spanning sets are considered [17] that conform to the geometry of the feasible region around the poll center. For bound constraints, BoostDMS considers coordinate directions, which are evaluated in a complete polling approach.

At the end of each iteration the stepsize is updated, keeping it constant for successful iterations and halving the stepsize of the poll center for unsuccessful ones.

A simplified description of BoostDMS is provided in Algorithm 1. For a detailed description see [11, 16].

The convergence analysis of Direct Multisearch algorithmic class relies on the behavior of the method at unsuccessful poll steps [16], holding independently of the initialization, of the strategy considered for the selection of the iterate point, of the definition of a search step, or of the type of polling strategy considered (opportunistic variants are allowed, in which polling is interrupted once that a new feasible nondominated point is found). Recently, worst-case complexity bounds were also derived for some variants of Direct Multisearch [15]. BoostDMS, as an algorithmic instance of DMS, inherits the convergence properties of this general class of Multiobjective Derivative-free Optimization methods.

From the algorithmic description provided, it is clear that parallelization strategies can be applied to both the search and the poll steps, enhancing the numerical performance of the algorithm. Section 4 will detail possible strategies for parallelization, from the simple distribution of objective function evaluations among the different processors to more sophisticated schemes, including the parallel computation of the quadratic polynomial models, or the simultaneous selection of several iterate points.

3. Metrics and performance assessment

To assess the quality of the strategies that are going to be proposed in Section 4, typical metrics from the Multiobjective Optimization literature are considered: purity and spread, as defined in [16], and the hypervolume indicator [38, 39]. For completeness, we provide a short description of each one of these metrics. A recent revision of Multiobjective Optimization metrics can be found in [3].

Purity measures the capability of a given solver in generating nondominated points. Let $\mathcal{P}$ and $\mathcal{S}$ be the sets of all problems and of all solvers considered, respectively. Let $F_{p,s}$ denote the approximation to the Pareto front computed for problem $p \in \mathcal{P}$ by solver $s \in \mathcal{S}$, and F_p denote the true Pareto front of problem p. The purity metric, for problem $p \in \mathcal{P}$ and solver $s \in \mathcal{S}$, is defined by the ratio

$$Pur_{p,s} = \frac{|F_{p,s} \cap F_p|}{|F_{p,s}|},$$

taking values between zero and one. Higher values of $Pur_{p,s}$ indicate a better Pareto front in terms of percentage of nondominated points generated. In general, F_p is not known, being considered an approximation to it by joining all the final approximations computed by all the solvers tested and removing all the dominated points from it.

Spread metrics assess the quality of the distribution of nondominated points across the final approximation to the Pareto front generated by the solver. Since we are interested in computing the complete Pareto front, spread metrics need to consider the extreme points associated to each objective function component (see [16]). The Γ metric (2), previously used to select the iterate point based on the maximum gap between consecutive points lying on the approximation to the Pareto front, is now computed using a set that also includes these extreme points. Again, since the true

Algorithm 1: A simplified description of BoostDMS.

```
Initialization:
  Choose a set of feasible points $\{lb \leq x^j_{ini} \leq ub,\ j \in \Upsilon\}$ with $f_i(x^j_{ini}) < +\infty$,
   $\forall i \in \{1, \ldots, q\}, \forall j \in \Upsilon$, and $\alpha^j_{ini} > 0$, $j \in \Upsilon$ initial stepsizes. Let $\mathcal{D}$ be a
   set of positive spanning sets. Initialize the cache of previously feasible
   evaluated points and corresponding function values
        $L_{cache} = \{(x^j_{ini}, F(x^j_{ini})), j \in \Upsilon\}$.
  Retrieve all nondominated points from $L_{cache}$ and initialize the list of
   feasible nondominated points, corresponding function values and stepsize
   parameters
        $L_0 = \{(x^j_{ini}, F(x^j_{ini}), \alpha^j_{ini}), j \in \bar{\Upsilon} \subseteq \Upsilon\}$.
for $k = 0, 1, 2, \ldots$ do
    Selection of an iterate point: Order the list $L_k$ according to the largest gap
    measured with the $\Gamma$ metric and select the first item $(x, F(x), \alpha) \in L_k$ as the
    current iterate, function value, and stepsize parameter (thus setting
    $(x_k, F(x_k), \alpha_k) = (x, F(x), \alpha)$).
    Search step: Select a subset of points in $L_{cache}$ to build the quadratic
    polynomial models. If the cardinality of the subset is smaller than $n + 2$ go to
    the poll step.
    for $i = 1, 2, \ldots, q$ do
        Build the quadratic polynomial model $m_i$, corresponding to the
        objective function component $f_i$.
    Set $l = 0$.
1   while $l < q$ do
        Set $l := l + 1$, define $J$ as the set of all combinations of $l$
        quadratic polynomial models taken from the total set of $q$
        models, and set $S = \emptyset$.
        for $j = 1, 2, \ldots, |J|$ do
            Compute the point $s_j$, solution of Problem (5), considering $I$
            as the set of indexes corresponding to the polynomial models
            associated to combination $j$. Update $S = S \cup \{s_j\}$.
2       Check for success: Evaluate $F$ at each point in $S$ and update
        $L_{cache}$. Compute $L_{trial}$ by removing all dominated points from
        $L_k \cup \{(x_s, F(x_s), \alpha_k) : x_s \in S\}$. If $L_{trial} \neq L_k$, set $L_{k+1} = L_{trial}$,
        stop the cycle loop while, declare the iteration as successful and
        skip the poll step.
    Poll step: Choose a positive spanning set $D_k$ from the set $\mathcal{D}$.
3   Evaluate $F$ at the feasible poll points belonging to $\{x_k + \alpha_k d : d \in D_k\}$ and
    update $L_{cache}$. Compute $L_{trial}$ by removing all dominated points from
    $L_k \cup \{(x_k + \alpha_k d, F(x_k + \alpha_k d), \alpha_k) : d \in D_k \wedge lb \leq x_k + \alpha_k d \leq ub\}$. If
    $L_{trial} \neq L_k$, declare the iteration as successful and set $L_{k+1} = L_{trial}$. Otherwise,
    declare the iteration unsuccessful and set $L_{k+1} = L_k$.
    Stepsize parameter update: If the iteration was unsuccessful halve the
    stepsize parameter corresponding to the poll center, replacing
    $(x_k, F(x_k), \alpha_k) \in L_{k+1}$ by $(x_k, F(x_k), \frac{\alpha_k}{2})$.
```

Pareto front is not known, an approximation to these extreme points is considered by joining the final approximations to the Pareto front computed by all the solvers tested and selecting the points corresponding to the minimum and maximum values for each component of the objective function (two points for each component).

Let us assume that solver $s \in \mathcal{S}$ has computed, for problem $p \in \mathcal{P}$, an approximated Pareto front with points $y_1, y_2, \ldots, y_N$, to which we add the extreme points mentioned above (y_0 and y_{N+1}). The metric $\Gamma_{p,s} > 0$, for problem $p \in \mathcal{P}$ and solver $s \in \mathcal{S}$, is given by

$$\Gamma_{p,s} \;=\; \max_{i \in \{1,\ldots,q\}} \left(\max_{j \in \{0,\ldots,N\}} \{\delta_{i,j}\} \right), \tag{6}$$

where $\delta_{i,j} = f_i(y_{j+1}) - f_i(y_j)$ (assuming that the objective function values have been sorted by increasing order for each objective function component i).

The second spread metric [18] measures the uniformity of the gap distribution across the computed Pareto front:

$$\Delta_{p,s} \;=\; \max_{i \in \{1,\ldots,q\}} \left(\frac{\delta_{i,0} + \delta_{i,N} + \sum_{j=1}^{N-1} |\delta_{i,j} - \bar{\delta}_i|}{\delta_{i,0} + \delta_{i,N} + (N-1)\bar{\delta}_i} \right), \tag{7}$$

where $\bar{\delta}_i$, for $i = 1, \ldots, q$, represents the average of the distances $\delta_{i,j}$, $j = 1, \ldots, N-1$.

The hypervolume indicator is a good compromise between spread and purity, since it measures the volume of the portion of the objective function space that is dominated by the computed approximation to the Pareto front and is limited by an upper corner $U_p \in \mathbb{R}^q$ (a point that is dominated by all points belonging to the different approximations computed for the Pareto front of problem $p \in \mathcal{P}$ by all solvers tested). Thus, the hypervolume indicator can be defined as follows:

$$HV_{p,s} = Vol\{y \in \mathbb{R}^q |\, y \le U_p \wedge \exists x \in F_{p,s} : x \le y\} = Vol\left(\bigcup_{x \in F_{p,s}} [x, U_p] \right),$$

where $Vol(.)$ denotes the Lebesgue measure of a q-dimensional set of points and $[x, U_p]$ denotes the interval box with lower corner x and upper corner U_p. The approach proposed in [24] was used for its computation, and the hypervolume values obtained were then scaled to the interval $[0, 1]$ (see [11] for details).

The final metric considered is thev(wall-clock) computational time required for solving a problem.

Performance profiles [21] are computed for the different metrics. Let $t_{p,s}$ be the value of a given metric obtained for problem $p \in \mathcal{P}$ with solver $s \in \mathcal{S}$, assuming that lower values of $t_{p,s} > 0$ indicate better performance. The cumulative distribution function, corresponding to the performance profile for solver $s \in \mathcal{S}$ is given by:

$$\rho_s(\tau) = \frac{1}{|\mathcal{P}|} \big|\{p \in \mathcal{P} : r_{p,s} \le \tau\}\big|,$$

with $r_{p,s} = t_{p,s} / \min\{t_{p,\bar{s}} : \bar{s} \in \mathcal{S}\}$. Thus, the value of $\rho_s(1)$ represents the probability of solver s winning over the remaining ones. Since for purity and hypervolume larger values indicate better performance, for these metrics the profiles considered $t_{p,s} := 1/t_{p,s}$, as proposed in [16].

For single objective optimization, data profiles [31] have been extensively used to assess how well derivative-free solvers perform in terms of the number of function evaluations. A data profile associated to a solver $s \in \mathcal{S}$ is a cumulative distribution function which returns the percentage of problems in $\mathcal{P}$ solved by the solver $s \in \mathcal{S}$ for a given budget of $\sigma \in \mathbb{N}$ function evaluations, expressed in simplexes, i.e.,

$$d_s(\sigma) = \frac{1}{|\mathcal{P}|} \left| \left\{ p \in \mathcal{P} : \frac{N_{p,s}}{n_p + 1} \leq \sigma \right\} \right|, \tag{8}$$

where $N_{p,s}$ is the number of function evaluations required by the solver $s \in \mathcal{S}$ to solve the problem $p \in \mathcal{P}$ (up to a certain accuracy), and n_p represents the dimension of problem $p \in \mathcal{P}$. A problem is considered to be solved with an accuracy $\psi > 0$ if

$$f(x_0) - f(x) \geq (1 - \psi)(f(x_0) - f_L),$$

where x_0 is the initialization provided and f_L denotes the best objective function value obtained for the problem, considering all the solvers tested. The correction $n_p + 1$ in (8) assumes that problems with higher dimensions usually require more function evaluations to be solved.

The extension of data profiles to multiobjective optimization was introduced in [16], using the purity metric to define the criterion for considering a problem as solved. Recently, data profiles based on the hypervolume were computed in [9].

Let HV_p represent the hypervolume of the reference Pareto front F_p, built for problem $p \in \mathcal{P}$ by joining the approximations to the Pareto front computed by all the solvers $s \in \mathcal{S}$ and removing all the dominated points. Solver $s \in \mathcal{S}$ is said to solve the problem $p \in \mathcal{P}$ for a given accuracy $\psi > 0$ if

$$\frac{HV_{p,s}}{HV_p/|\mathcal{S}|} \geq 1 - \psi. \tag{9}$$

The reference hypervolume HV_p is divided by the number of solvers in $\mathcal{S}$, assuming that is expected that all solvers contribute equally to the definition of the hypervolume of the reference Pareto front. A similar approach was proposed in [16], when computing data profiles based on the purity metric.

However, in the context of parallelization, data profiles should reflect the percentage of problems solved inside a given budget of computational time, rather than in a budget of function evaluations. Therefore, we propose a new variant of data profiles based on hypervolume in which $N_{p,s}$ in (8) is replaced by $t_{p,s}$, the computational time (in seconds) taken by the solver $s \in \mathcal{S}$ to solve the problem $p \in \mathcal{P}$.

Since the hypervolume of the approximated Pareto front is not available for every single second of execution time and retrieving it would perturb the computational time associated to the problem solution, the number of problems solved according to (9) is only checked for final computational times associated to the solution of each one of the problems in $\mathcal{P}$.

As test set, we considered 99 bound constrained Multiobjective Optimization problems from the collection available at `http://www.mat.uc.pt/dms`, now coded in Matlab, with a number of variables, n, between 1 and 30, and with $2, 3$, or 4 objective function components (problem `WFG1` was not considered due to some numerical instabilities of Matlab). Since these are academic problems, the time associated to function evaluation is quite small, of the order of milliseconds, very different from real applications, where function evaluation typically requires seconds or even minutes of computational time. For a fair assessment of the benefits of parallelization strategies, each function was modified with an average computational time of 0.1 seconds (in single objective optimization, computational times of similar magnitude have already allowed benefits from parallelization [34]). Results are reported for average computational time, considering five runs.

All numerical experiments, with exception of a speedup study performed in Section 5, were executed on a Xeon Platinium 8171M CPU machine from the Azure cloud, with 8 processors, running Linux as operating system. Solvers ran in Matlab R2020a, being the toolboxes of Optimization and Parallel Computing required for the minimization of the quadratic models at the search step and for distributing tasks to the different processors, whenever parallelization was required. Default options were assumed for BoostDMS [11], corresponding to the algorithmic description presented in Section 2. In particular, a maximum of 20 000 function evaluations or a minimum value of 10^{-3} for the stepsize parameter α_k of each point in the list were used as stopping criteria.

4. Performance improvement strategies

The expensive cost of function evaluation, associated to Derivative-free Optimization problems, naturally motivates its parallelization. However, additional strategies can and will be considered, not only with the goal of computational time reduction but also with the concern of keeping or improving the quality of the computed approximation to the Pareto front.

4.1. Parallelization of objective function evaluations

In BoostDMS, parallelization of function evaluations was considered both at the poll and the search steps, as a way of improving the numerical performance of the solver.

In Directional Direct Search, the poll step has a natural structure for parallelization. BoostDMS considers a complete polling approach, allowing to take full advantage of an embarrassingly parallel scheme [27]. The set of points to be evaluated at the poll step is generated and infeasible points or points considered identical to others already included in the cache (within a tolerance equal to the minimum stepsize allowed) are discarded, before function evaluation. This filtered set of points is now evaluated in parallel. This strategy corresponds to the simple replacement of “Evaluate F” by “Evaluate F in parallel” in the line numbered as 3, in Algorithm 1.

All evaluated points are included in the cache (L_{cache}) and dominance is checked, adding nondominated points to the list L_k, one by one, in a similar way to what is done in the sequential version. This approach ensures that the approximation generated for the Pareto front is exactly the same than the one generated by the sequential version, keeping the quality of the final computed solution.

As described in Section 2, the search step considers an approach by levels, when minimizing the quadratic polynomial models. Problem (5) may need to be solved for all combinations of l models. At level $l = 1$, the individual minimization of the models is achieved through function `trust.m`, from Matlab, and projection of the resulting points in the feasible region. For levels two and above ($l \geq 2$), the joint minimization of models resources to Matlab’s function `fmincon.m`. All solutions of the subproblems at a given level are evaluated and, only in case of failure in finding a new feasible nondominated point, the next level is considered.

Three different approaches have been tested, regarding the parallelization of function evaluations at the search step. The first, *Par evals within levels*, keeps the function evaluation structure by levels of the sequential implementation. A maximum of q batches of parallel function evaluations can be performed (one associated to each level). Again, this approach simply corresponds to replace "Evaluate F" by "Evaluate F in parallel", this time in line 2 of Algorithm 1. The quality of the solution continues to be identical to the one of the sequential version, although a reduction in computational time is expected. The second variant, denoted by *Par evals 2 batches*, considers function evaluation in two batches, one corresponding to level $l = 1$ and, in case of failure in finding a new feasible nondominated point, all the remaining levels, corresponding to the joint minimization of two or more models. In Algorithm 1, the cycle loop **while** starting at line 1 is replaced by the schematic description detailed in Algorithm 2.

Algorithm 2: *Par evals 2 batches*
Procedure for function evaluation in the search step of Algorithm 1.

```
Set $S = \emptyset$.
for $i = 1, 2, \ldots, q$ do
    Compute the point $s_i$, solution of Problem (5), considering $I = \{i\}$. Update
      $S = S \cup \{s_i\}$.
Check for success: Evaluate $F$ in parallel for all the points in $S$ and update
  $L_{cache}$. Compute $L_{trial}$ by removing all dominated points from
  $L_k \cup \{(x_s, F(x_s), \alpha_k) : x_s \in S\}$. If $L_{trial} \neq L_k$, set $L_{k+1} = L_{trial}$, declare the
  iteration as successful and move to the next iteration. Otherwise, set $S = \emptyset$,
  $l = 1$, and proceed.
while $l < q$ do
    Set $l := l + 1$, define $J$ as the set of all combinations of $l$ quadratic
      polynomial models taken from the total set of $q$ models.
    for $j = 1, 2, \ldots, |J|$ do
        Compute the point $s_j$, solution of Problem (5), considering $I$ as the set
          of indexes corresponding to the polynomial models associated to
          combination $j$. Update $S = S \cup \{s_j\}$.
Check for success: Evaluate $F$ in parallel for all the points in $S$ and update
  $L_{cache}$. Compute $L_{trial}$ by removing all dominated points from
  $L_k \cup \{(x_s, F(x_s), \alpha_k) : x_s \in S\}$. If $L_{trial} \neq L_k$, set $L_{k+1} = L_{trial}$, declare the
  iteration as successful and skip the poll step.
```

Finally, the variant denoted by *Par evals no levels* solves all Problems (5), evaluating the objective function at the corresponding solutions in parallel. This strategy corresponds to the schematic description of Algorithm 3, again replacing the cycle loop **while** starting at line 1.

In any of the three strategies, Problems (5) are always solved sequentially. Only function evaluations are parallelized. Cache is always checked before function evaluation, similarly to what is performed in the poll step.

A level-by-level approach would be beneficial if success is often found in lower levels, even though it could not take advantage of all the potential of parallelizing function evaluations. However, if success is typically achieved in higher levels or if the search step is often unsuccessful, considering a single batch of parallel function evaluations

Algorithm 3: *Par evals no levels*

Procedure for function evaluation in the search step of Algorithm 1.

```
Set $S = \emptyset$ and $l = 0$.
while $l < q$ do
    Set $l := l + 1$, define $J$ as the set of all combinations of $l$ quadratic
      polynomial models taken from the total set of $q$ models.
    for $j = 1, 2, \ldots, |J|$ do
        Compute the point $s_j$, solution of Problem (5), considering $I$ as the set
          of indexes corresponding to the polynomial models associated to
          combination $j$. Update $S = S \cup \{s_j\}$.
Check for success: Evaluate $F$ in parallel for all the points in $S$ and update
  $L_{cache}$. Compute $L_{trial}$ by removing all dominated points from
  $L_k \cup \{(x_s, F(x_s), \alpha_k) : x_s \in S\}$. If $L_{trial} \neq L_k$, set $L_{k+1} = L_{trial}$, declare the
  iteration as successful and skip the poll step.
```

of points, corresponding to all levels, would be more efficient.

Figures 1, 2, 3, and 4 compare the results obtained with the sequential version to the three parallelization strategies described above, for the five metrics considered. In any of these last three versions, function evaluation at the poll step is also parallelized.

As previously described, the strategy *Par evals within levels* mimics the sequential approach, respecting the structure of levels in function evaluation at the search step. This justifies the overlapping of the curves corresponding to these two strategies in Figures 1, 2, and 3.

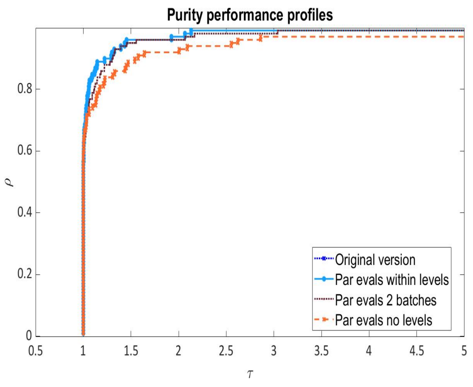


Figure 1.: Comparison between the sequential version of BoostDMS, and the parallel versions *Par evals within levels*, *Par evals 2 batches*, and *Par evals no levels*, by means of purity metric.

All strategies present a similar performance in terms of hypervolume and Δ metrics. As expected, there is a clear advantage in computational time for the parallel versions, with the variant *Par evals no levels* performing the best (see Figure 4). A strategy that gathers all the points before evaluation, reduces the number of batches to be parallelized, translating to better computational times.

However, analyzing the plots corresponding to purity and Γ metrics, we conclude that this strategy presents a worse performance than the other parallel versions. In fact, the results for the three remaining strategies are very close, indicating that fea-

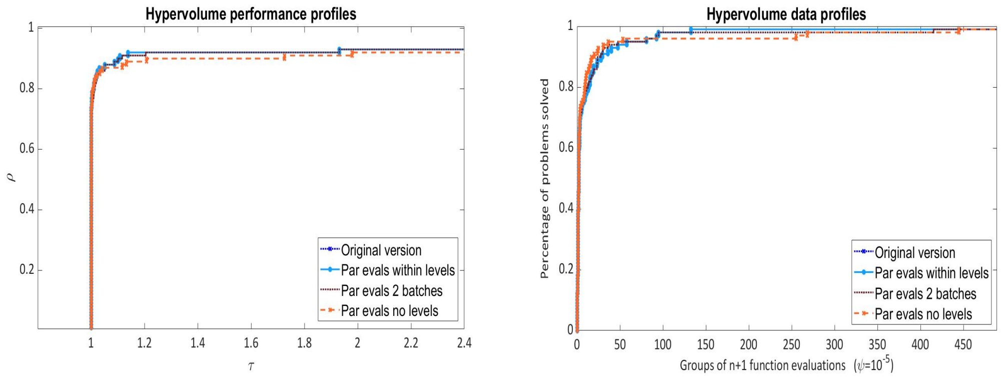


Figure 2.: Comparison between the sequential version of BoostDMS, and the parallel versions *Par evals within levels*, *Par evals 2 batches*, and *Par evals no levels*, by means of hypervolume metric (performance and data profiles).

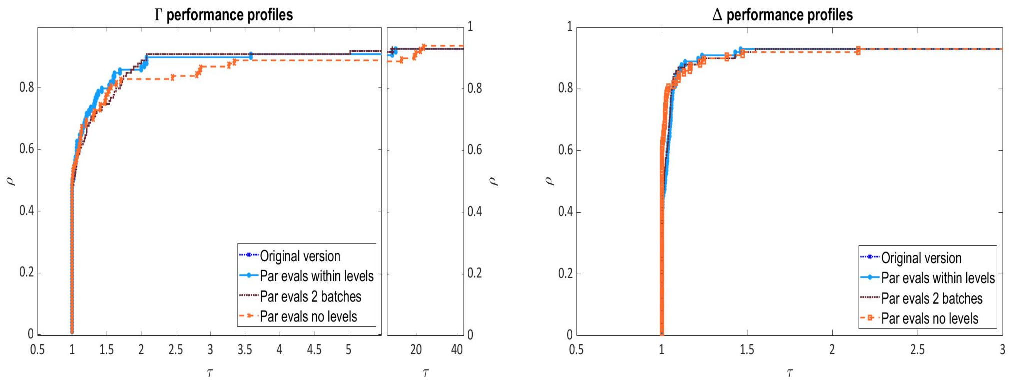


Figure 3.: Comparison between the sequential version of BoostDMS, and the parallel versions *Par evals within levels*, *Par evals 2 batches*, and *Par evals no levels*, by means of spread metrics (Γ and Δ).

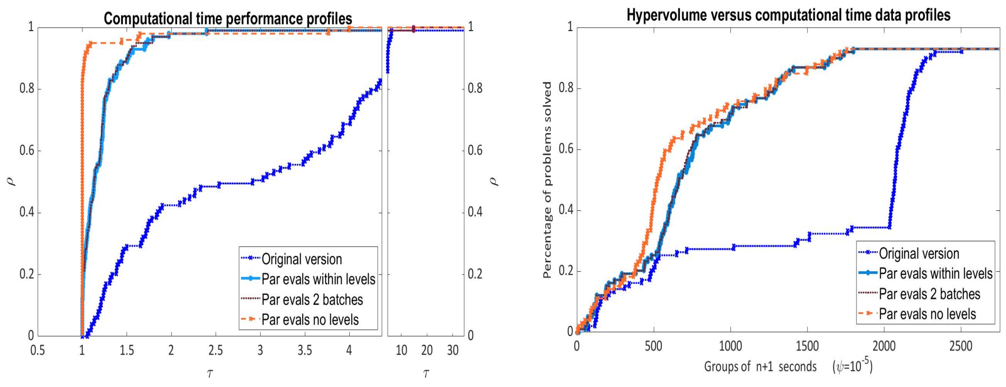


Figure 4.: Comparison between the sequential version of BoostDMS, and the parallel versions *Par evals within levels*, *Par evals 2 batches*, and *Par evals no levels*, considering the computational time (performance profiles on the left and hypervolume versus computational time data profiles on the right).

sible nondominated points are typically found in the first level, when the individual minimization of models occurs. Strategy *Par evals no levels* performs a larger number of function evaluations per search step, resulting in a slightly better performance in terms of hypervolume at the beginning of the optimization process (see Figures 2 and 4), but the exhaustion of the function evaluations budget at early iterations does not allow it to refine the quality of the computed approximation to the Pareto front (although, it contributes to better computational times).

The variant *Par evals within levels*, that keeps the structure of the parallel function evaluations in batches corresponding to levels at the search step, allows a good performance in terms of computational time, keeping the quality of the final solution generated by the sequential version. This was the option taken in terms of strategy for function evaluation and will be adopted in the following numerical experiments.

4.2. Parallelization of models computation and minimization

The parallelization can obviously be extended to the computation of the quadratic models for each component of the objective function and to the corresponding individual or joint minimization. Thus, three new parallel strategies were developed.

The first, denoted by *Model min (level 1) and par evals*, parallelizes the individual computation and minimization of models, for each objective function component. In Algorithm 1, the search step will be replaced by Algorithm 4.

Algorithm 4: *Model min (level 1) and par evals*
Search step of Algorithm 1.

```
Select a subset of points in $L_{cache}$ to build the quadratic polynomial models. If
  the cardinality of the subset is smaller than $n+2$ go to the poll step.
In parallel, build the $q$ quadratic polynomial models $m_i$, corresponding to the
  objective function components $f_i$, and compute the points $s_i$, solution of
  Problems (5), considering $I = \{i\}$. Define $S = \{s_1, \ldots, s_q\}$.
Check for success: Evaluate $F$ in parallel for all the points in $S$ and update
  $L_{cache}$. Compute $L_{trial}$ by removing all dominated points from
  $L_k \cup \{(x_s, F(x_s), \alpha_k) : x_s \in S\}$. If $L_{trial} \neq L_k$, set $L_{k+1} = L_{trial}$, declare the
  iteration as successful and move to the next iteration. Otherwise, set $l = 1$
  and proceed.
while $l < q$ do
    Set $l := l + 1$, define $J$ as the set of all combinations of $l$ quadratic
      polynomial models taken from the total set of $q$ models, and set $S = \emptyset$.
    for $j = 1, 2, \ldots, |J|$ do
        Compute the point $s_j$, solution of Problem (5) considering $I$ as the set
          of indexes corresponding to the polynomial models associated to
          combination $j$. Update $S = S \cup \{s_j\}$.
    Check for success: Evaluate $F$ in parallel at each point in $S$ and update
      $L_{cache}$. Compute $L_{trial}$ by removing all dominated points from
      $L_k \cup \{(x_s, F(x_s), \alpha_k) : x_s \in S\}$. If $L_{trial} \neq L_k$, set $L_{k+1} = L_{trial}$, stop the
      cycle loop while, declare the iteration as successful and skip the poll step.
```

Strategy *Model min (level ≥ 2) and par evals* builds models and performs their individual minimization sequentially, only parallelizing the joint minimization of models. This time, the search step of Algorithm 1 will be replaced by Algorithm 5. Considering the results of Section 4.1, the approach of evaluating batches of points by level is always followed. Since the numerical results of Section 4.1 indicate that the search step is mainly successful at level $l = 1$, the number of search steps where levels $l \geq 2$ will be considered is reduced, thus no major improvement is expected with this second variant.

Algorithm 5: *Model min (level $\geq$ 2) and par evals*
Search step of Algorithm 1.

```
Select a subset of points in $L_{cache}$ to build the quadratic polynomial models. If
  the cardinality of the subset is smaller than $n+2$ go to the poll step.
Set $S = \emptyset$.
for $i = 1, 2, \dots, q$ do
    Build the quadratic polynomial model $m_i$, corresponding to the objective
      function component $f_i$, and compute the point $s_i$, solution of Problem (5),
      considering $I = \{i\}$. Update $S = S \cup \{s_i\}$.
Check for success: Evaluate $F$ in parallel for all the points in $S$ and update
  $L_{cache}$. Compute $L_{trial}$ by removing all dominated points from
  $L_k \cup \{(x_s, F(x_s), \alpha_k) : x_s \in S\}$. If $L_{trial} \neq L_k$, set $L_{k+1} = L_{trial}$, declare the
  iteration as successful and move to the next iteration. Otherwise, set $l = 1$
  and proceed.
while $l < q$ do
    Set $l := l + 1$ and define $J$ as the set of all combinations of $l$ quadratic
      polynomial models taken from the total set of $q$ models.
    In parallel, compute the set of points $S$, solution of Problems (5),
      considering $I$ as the set of indexes associated to each combination $j \in J$.
    Check for success: Evaluate $F$ in parallel at each point in $S$ and update
      $L_{cache}$. Compute $L_{trial}$ by removing all dominated points from
      $L_k \cup \{(x_s, F(x_s), \alpha_k) : x_s \in S\}$. If $L_{trial} \neq L_k$, set $L_{k+1} = L_{trial}$, stop the
      cycle loop while, declare the iteration as successful and skip the poll step.
```

A final strategy, denoted by *Model min (all) and par evals* is detailed in Algorithm 6, parallelizing model building and minimization at all levels.

The performance of *Par evals within levels* is very similar to the one of any of these three new strategies, so we omit the corresponding profiles. The two strategies that parallelize the model computation present a slightly worse performance in terms of computational time. This could be explained by the fact that we are addressing problems with a low number of components in the objective function, with a reasonably small dimension ($n \leq 30$), which has implications in the linear systems of equations to be solved for computing the quadratic models. It is also noteworthy the fact that some of the Matlab's functions that are used to solve the linear systems of equations are implicitly parallelized in Matlab. Since the results did not bring advantages, the strategy *Par evals within levels* continues to be the default.

4.3. Iterate point selection based on spread

As described in Section 2, each iteration of the algorithm starts with the selection of an iterate point, which will be used as model center, at the search step, or, in case of failure of the search step in finding a new feasible nondominated point, as poll center, at the poll step. The selection is made from the list L_k of feasible nondominated points

Algorithm 6: *Model min (all) and par evals*
Search step of Algorithm 1.

```
Select a subset of points in $L_{cache}$ to build the quadratic polynomial models. If
  the cardinality of the subset is smaller than $n + 2$ go to the poll step.
In parallel, build the $q$ quadratic polynomial models $m_i$, corresponding to the
  objective function components $f_i$, and compute the points $s_i$, solution of
  Problems (5), considering $I = \{i\}$. Define $S = \{s_1, \ldots, s_q\}$.
**Check for success:** Evaluate $F$ in parallel for all the points in $S$ and update
  $L_{cache}$. Compute $L_{trial}$ by removing all dominated points from
  $L_k \cup \{(x_s, F(x_s), \alpha_k) : x_s \in S\}$. If $L_{trial} \neq L_k$, set $L_{k+1} = L_{trial}$, declare the
  iteration as successful and move to the next iteration. Otherwise, set $l = 1$
  and proceed.
**while** $l < q$ **do**
    Set $l := l + 1$ and define $J$ as the set of all combinations of $l$ quadratic
      polynomial models taken from the total set of $q$ models.
    In parallel, compute the set of points $S$, solution of Problems (5),
      considering $I$ as the set of indexes associated to each combination $j \in J$.
    **Check for success:** Evaluate $F$ in parallel at each point in $S$ and update
      $L_{cache}$. Compute $L_{trial}$ by removing all dominated points from
      $L_k \cup \{(x_s, F(x_s), \alpha_k) : x_s \in S\}$. If $L_{trial} \neq L_k$, set $L_{k+1} = L_{trial}$, stop the
      cycle loop **while**, declare the iteration as successful and skip the poll step.
```

and corresponds to the point with the largest value of the Γ metric (2).

This spread metric is considered in an attempt to reduce the gap between consecutive points lying in the current approximation to the Pareto front, after projection of each objective function component in the corresponding dimension. Since the range of each objective function component can be in very different scales, this metric could be biased towards one/some of the function components.

Two additional strategies were considered to prevent this fact. The first, denoted by *Par evals within levels (Gamma normalized)*, before computing the largest gap, using (2), normalizes the values of each objective function component $i \in \{1, \ldots, q\}$ using the formula:

$$\frac{f_i(y_j) - \min_{j=1,\ldots,N} f_i(y_j)}{\max_{j=1,\ldots,N} f_i(y_j) - \min_{j=1,\ldots,N} f_i(y_j)},$$

where $y_1, \ldots, y_N$ represent the points in the current list of feasible nondominated points L_k. This way, a fairer selection of the largest gap among all components is expected.

The second strategy, denoted by *Par evals within levels (Gamma cyclic)*, considers a cyclic approach, changing the objective function component for which the largest gap is computed from iteration to iteration in a recurrent way. In (2), the maximum over the different components of the objective functions is no longer considered. Instead, a single component of the objective function is selected, changing it from iteration to iteration, in a cyclic way. For example, on a tri-objective problem, iterations $1, 4, 7, \ldots$, select the iterate point corresponding to the largest gap between consecutive points only analyzing the first component of the objective function, while iterations $2, 5, 8, \ldots$, and $3, 6, 9, \ldots$, consider the gaps related to the second and third components of the objective function, respectively.

Figures 5, 6, 7, and 8 compare these two strategies with *Par evals within levels*. As usual, function evaluation is parallelized both at the search and the poll steps, in the former respecting the levels structure.

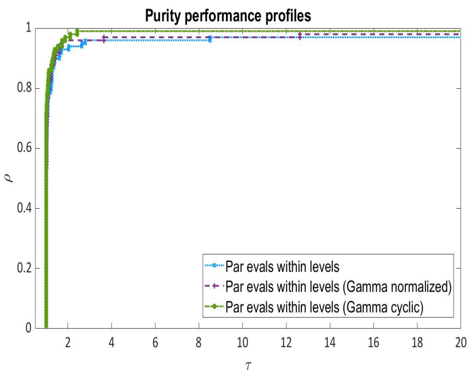
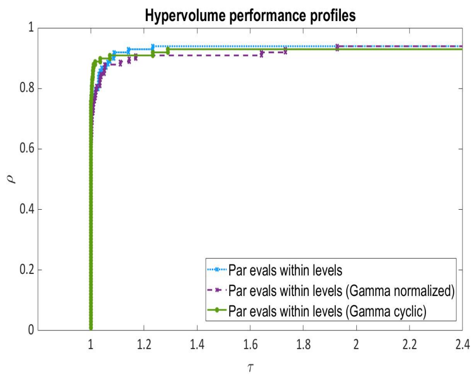


Figure 5.: Comparison between the strategies *Par evals within levels*, *Par evals within levels (Gamma normalized)*, and *Par evals within levels (Gamma cyclic)*, by means of purity metric.



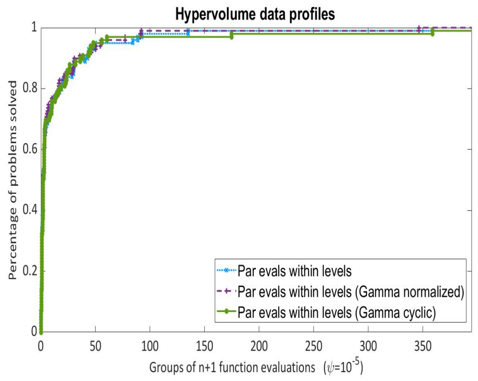


Figure 6.: Comparison between the strategies *Par evals within levels*, *Par evals within levels (Gamma normalized)*, and *Par evals within levels (Gamma cyclic)*, by means of hypervolume metric (performance and data profiles).

Although the results are very similar for hypervolume and Γ metrics, a slight advantage of *Par evals within levels (Gamma cyclic)* is noticed for purity, Δ and computational time, which justifies our option to use it as strategy for the iterate point selection.

4.4. Selection of more than one iterate point

Iterate points are important not only as base points for trying to close gaps in the Pareto front, but can also be a way of expanding the Pareto front, in an attempt of reaching the corresponding extreme points. Several parallel strategies were developed, focusing on these two goals. Variants included the selection of more than one iterate

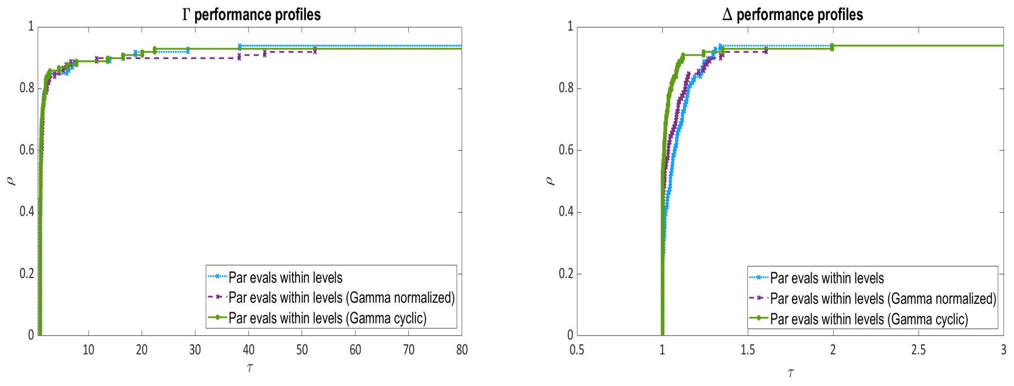


Figure 7.: Comparison between the strategies *Par evals within levels*, *Par evals within levels (Gamma normalized)*, and *Par evals within levels (Gamma cyclic)*, by means of spread metrics (Γ and Δ).

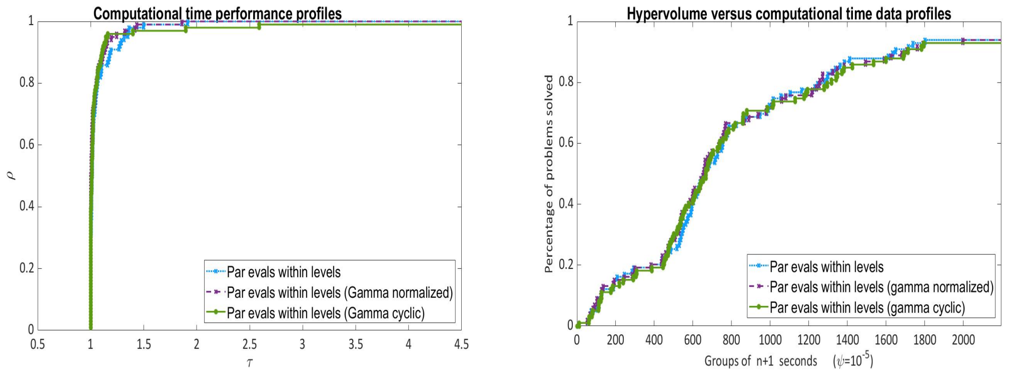


Figure 8.: Comparison between the strategies *Par evals within levels*, *Par evals within levels (Gamma normalized)*, and *Par evals within levels (Gamma cyclic)*, considering the computational time (performance profiles on the left and hypervolume versus computational time data profiles on the right).

point per iteration, considering the Γ metric applied in a cyclic way, the objective function value (minimum and maximum values at each component), and the stepsize parameter (larger values indicating more promising points). We only report results for the two most successful variants.

Both new strategies take into account the two points defining the largest gap for the objective function component corresponding to the current iteration, as well as the q points corresponding to the lowest value for each objective function component. In the first strategy, denoted by *It centers based on spread + best values*, each iteration considers $q+2$ points as iterate points whereas in the second strategy, *It centers based on spread* $\vee$ *best values*, the number of iterate points alternates between iterations: selection of the two points corresponding to the largest gap or the q points corresponding to the lowest objective function components values. The selected iterate points will be used as model centers at the search step and all those that fail in generating a new feasible nondominated point will be used as poll centers. As usual, function evaluation is performed in parallel, keeping the levels structure at the search step.

Figures 9, 10, 11, and 12 compare these two strategies with *Par evals within levels (Gamma cyclic)*.

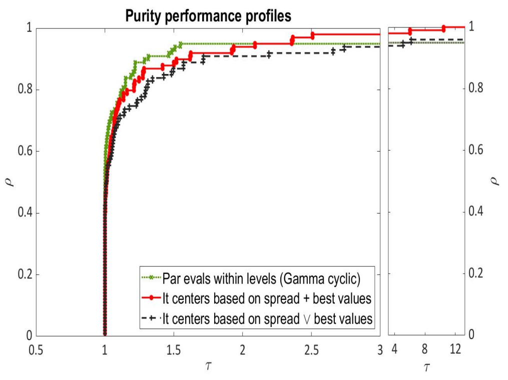


Figure 9.: Comparison between the parallel strategies *Par evals within levels (Gamma cyclic)*, *It centers based on spread + best values*, and *It centers based on spread* $\vee$ *best values*, by means of purity metric.

Although *Par evals within levels (Gamma cyclic)* presents a better performance for purity and Δ metrics, strategy *It centers based on spread + best values* clearly outperforms the other strategies in computational time, and presents a slightly better performance in what respects to the hypervolume metric, justifying our option for it.

4.5. Selected parallelization approach

Finally, we compared our selected parallel strategy with the initial variant of BoostDMS. This initial variant of BoostDMS is a sequential implementation, with selection of one iterate point per iteration, based on the Γ metric (see (2)).

The parallel strategy selected considers $q+2$ iterate points from the current list L_k, corresponding to the q points with the lowest value for each component of the objective function and to the two points defining the largest gap in the component of the objective function associated to the current iteration (which changes between iterations in a cyclic way).

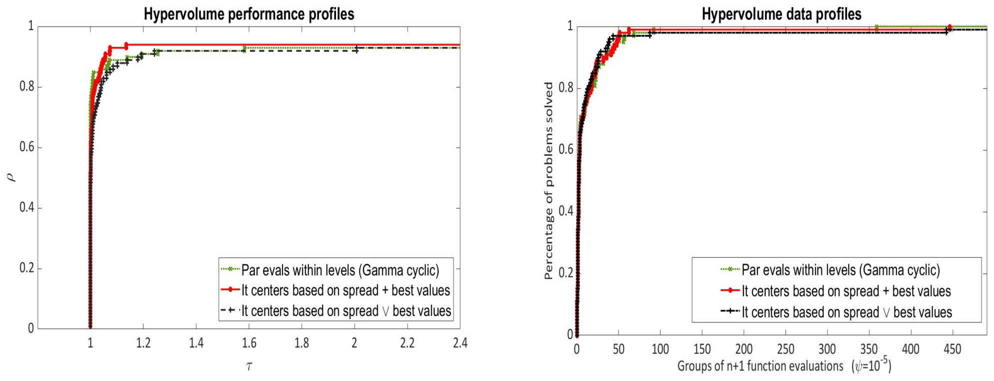


Figure 10.: Comparison between the parallel strategies *Par evals within levels (Gamma cyclic)*, *It centers based on spread + best values*, and *It centers based on spread ∨ best values*, by means of hypervolume metrics (performance and data profiles).

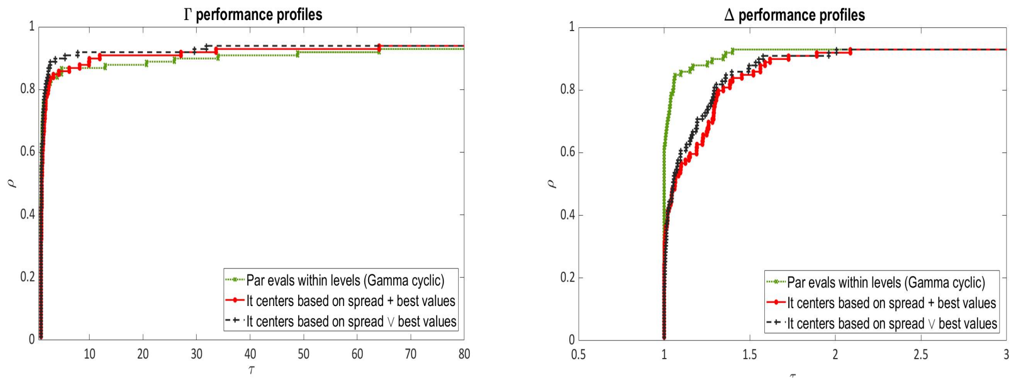


Figure 11.: Comparison between the parallel strategies *Par evals within levels (Gamma cyclic)*, *It centers based on spread + best values*, and *It centers based on spread ∨ best values*, by means of spread metrics (Γ and Δ).

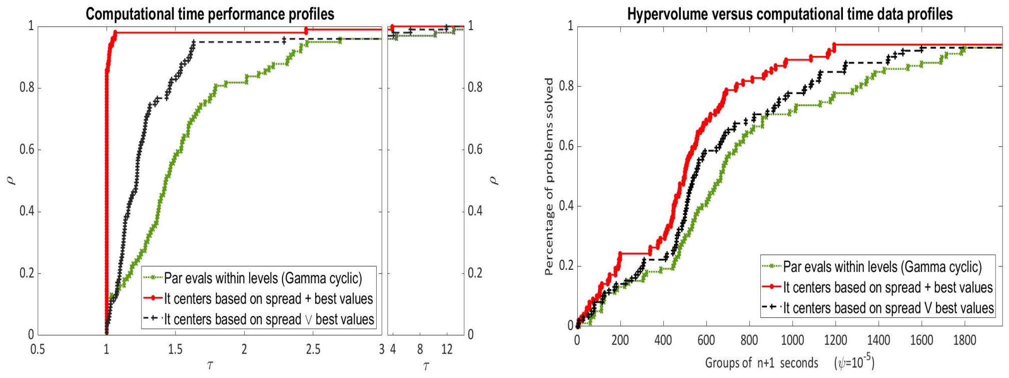


Figure 12.: Comparison between the parallel strategies *Par evals within levels (Gamma cyclic)*, *It centers based on spread + best values*, and *It centers based on spread ∨ best values*, considering the computational time (performance profiles on the left and hypervolume versus computational time data profiles on the right).

These $q + 2$ points will be used as model centers at the search step and all those that fail in generating a new feasible nondominated point will be used as poll centers. Function evaluations are always performed in parallel, at the search and poll steps. In the former, the levels structure is respected, only evaluating a given level if the previous one was unable to generate a new feasible nondominated point.

Figures 13, 14, 15, and 16 report the performance profiles comparing *It centers based on spread + best values* strategy with the original sequential version of BoostDMS.

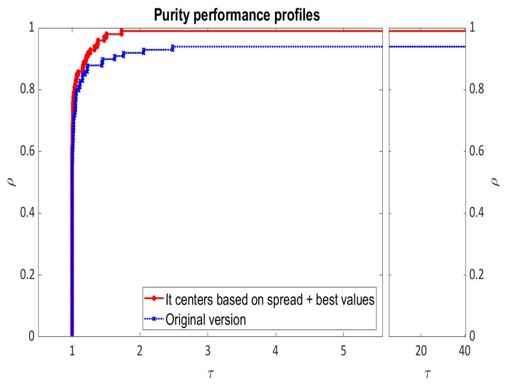
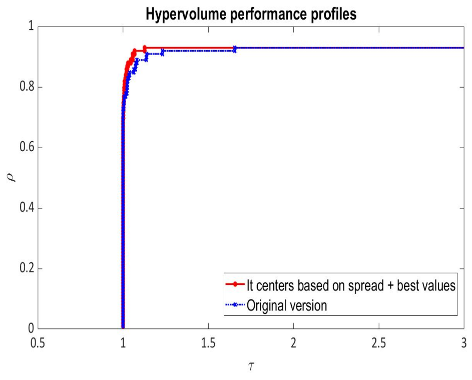


Figure 13.: Comparison between the selected parallel version and the original sequential implementation of BoostDMS, by means of purity metric.



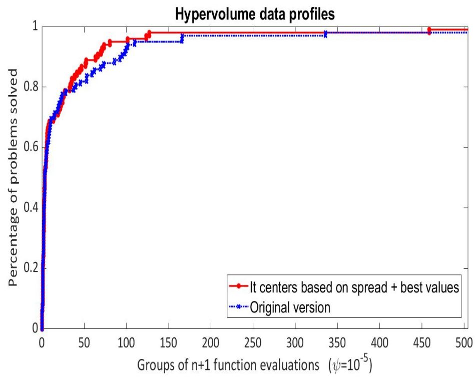


Figure 14.: Comparison between the selected parallel version and the original sequential implementation of BoostDMS, by means of hypervolume metric (performance and data profiles).

Although there is a decrease in performance for spread (see Figure 15), the advantage brought by the parallel strategy selected is clear. Not only it presents a huge increase in performance regarding computational time (see Figure 16), but also a better performance for purity and a slight advantage in terms of hypervolume (see Figures 13 and 14).

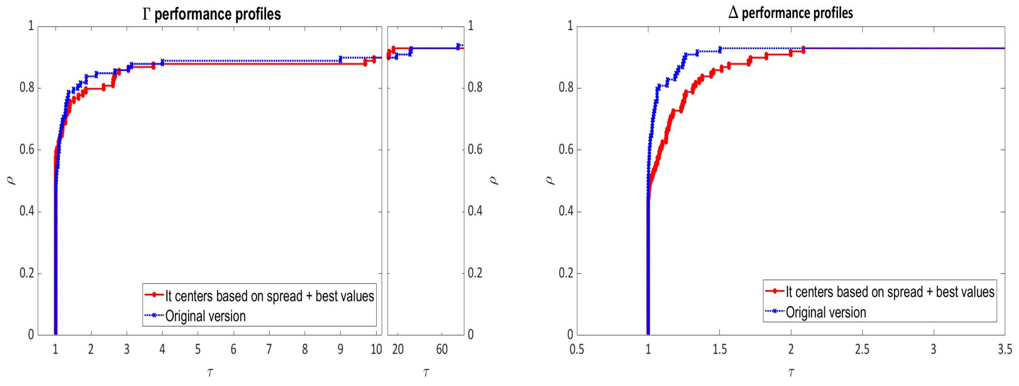


Figure 15.: Comparison between the selected parallel version and the original sequential implementation of BoostDMS, by means of spread metrics (Γ and Δ).

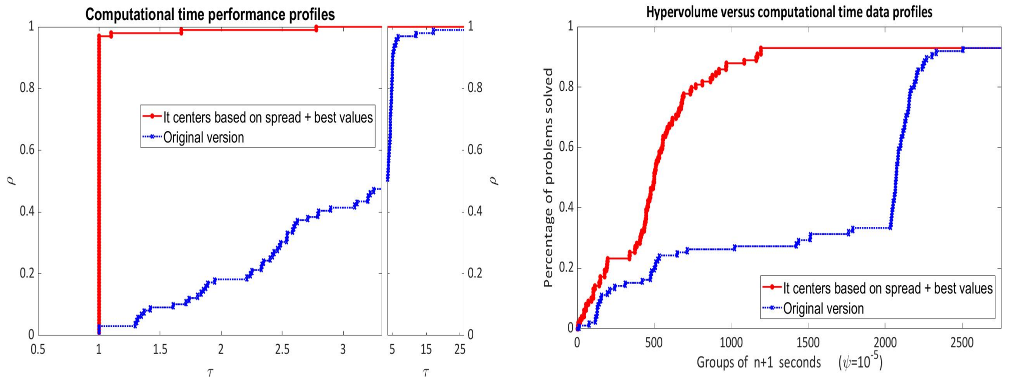


Figure 16.: Comparison between the selected parallel version and the original sequential implementation of BoostDMS, considering computational time (performance profiles on the left and hypervolume versus computational time data profiles on the right).

5. A chemical engineering application

Often academic problems, like the ones considered in Section 4, do not reflect all the challenges of real applications. In this section, we consider a real chemical engineering problem, related to the production of styrene, as described in [2, 7].

The styrene production process involves four steps: preparation of reactants, catalytic reactions, a first distillation – where styrene is recovered, and a second distillation – where benzene is recovered. During this second distillation, unreacted ethylbenzene is recycled, as an initial reactant on the process. This is a tri-objective problem ($q = 3$), where it is intended to maximize the net present value associated to the process (f_1), the purity of the produced styrene (f_2), and the overall ethylbenzene conversion into styrene (f_3) [7]. The problem has 8 variables, subject to bounds constraints, and 9 other constraints, some process-related (*e.g.* environmental norms regarding excesses) and some economical (*e.g.* investment value). More details can be found in [2].

A C++ numerical simulator has been developed for this chemical engineering process (available at `https://github.com/bbopt/styrene`), where each complete simulation corresponds to a function evaluation. Due to the presence of recycling loops, like the one of ethylbenzene, the process must be entirely simulated until a result is provided. The total time associated to the computation of a function value fluctuates, with an average of 1 second, when the code succeeds in evaluating a given point. Additionally, the problem reveals hidden constraints, since feasible points, according to the defined constraints, often fail to produce a finite numerical value for the objective function. In this case, the simulation is generally faster, with an average computational time of 0.001 seconds. Note that these computational times were obtained with the hardware/software configuration described in Section 3.

We used DMS [16], BoostDMS [11], and the selected parallel version of BoostDMS, described in Section 4.5, to solve the problem. All the defaults were considered for the solvers, with exception of the initialization, where line sampling was replaced by a single point, provided in literature [2]. Figure 17 corresponds to the approximations to the Pareto front generated by each one of the three solvers.

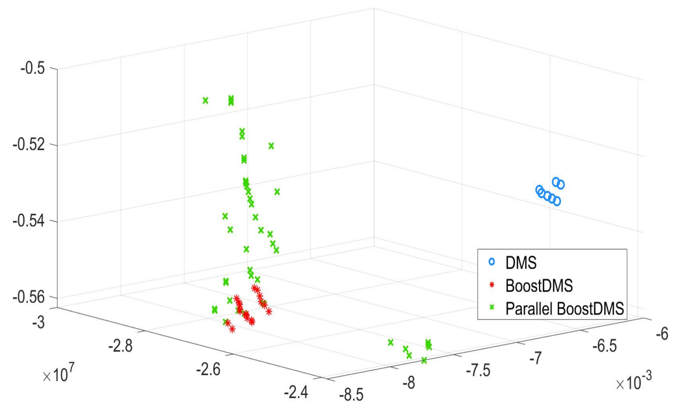


Figure 17.: Final approximations to the Pareto front computed for the chemical engineering problem by solvers DMS, BoostDMS, and the selected parallel version of BoostDMS (points coordinates are available on Appendix A).

The 7 points in the final solution computed by DMS are all dominated by each one of the approximations to the Pareto front computed by BoostDMS or its parallel version (which comprise 19 and 44 nondominated points, respectively). Table 1 presents the values of the different multiobjective metrics for the three solvers, as well as the computational time (an average of five runs), and the total number of function evaluations performed.

Solver	**Purity**	**Hypervolume**	**Gamma**	**Delta**	**AvgTime(s)**	**FuncEvals**
DMS	0%	0.002	3.96e+06	1.071	3987	9260
BoostDMS	53%	0.70	3.13e+06	1.173	6970	14978
Parallel BoostDMS	89%	0.78	2.52e+06	1.224	1670	20000

Table 1.: Metrics associated to the solution of the chemical engineering problem, computed by solvers DMS, BoostDMS, and the selected parallel version of BoostDMS.

BoostDMS is considerably slower than DMS, but presents a remarkable improvement in the quality of the solution computed. This quality increases in the solution computed by the parallel version of BoostDMS, particularly in what respects to purity. These results corroborate what was observed in the academic test set, with the parallel version of BoostDMS allowing relevant gains in terms of purity, a slight advantage regarding hypervolume, but a worse performance in the spread metric Δ.

Although the parallel version of BoostDMS performs a larger number of function evaluations than any of the two other solvers, it is considerably faster than any of the two sequential versions. In fact, if we compute the speedup associated to the parallel version, we will get a value of 4.17, corresponding to an efficiency of 52.17% (for the test set of Section 4, the average speedup is in the interval [3.22; 4.45] and the average efficiency in [40.31%; 55.68%], in both cases with 95% of confidence). However, considering that the number of function evaluations performed by the sequential version is clearly lower than the one of the parallel version, a crude correction, assuming that a total of 20 000 function evaluations was performed by both variants, would give a speedup of 5.6, corresponding to an efficiency of 70%.

For a fair study of the speedup, the problem was again solved with the parallel version of BoostDMS on a Xeon Platinium 8171M CPU machine from the Azure cloud, now with 32 processors available, selecting as number of workers integer powers of 2 between 1 and 32. Results are reported in Table 2 and illustrated in Figure 18. As usual, average computational time is based on five runs.

Table 2.: Average computational time and speedup for the styrene problem.

#Workers	AvgTime(s)	Speedup
1	9033	–
2	5029	1.80
4	2871	3.15
8	1801	5.01
16	1194	7.56
32	982	9.20

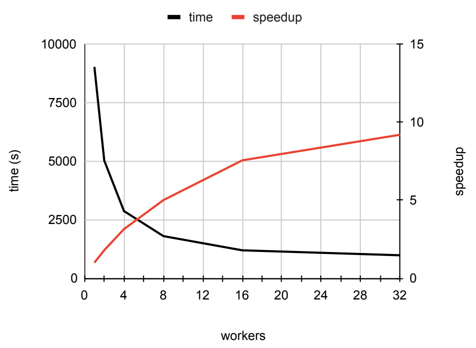


Figure 18.: Average computational time and speedup for different number of workers in the styrene problem.

When starting to increase the number of workers, a steep decrease in the execution time can be observed (see Figure 18). However, for larger numbers of workers, the improvements are less significant. In this particular problem, the objective function has three components, and 5 iterate points are selected at each iteration. The strategy selected for function evaluation at the search step, *Par evals within levels*, considers sequential batches of function evaluations, corresponding to levels. The first and second batches have a maximum of 15 function evaluations to be performed and the third batch a maximum of 5. Thus, in the search step, the maximum number of parallel function evaluations is 15. Only the unsuccessful iterate points will be explored at the poll step. Additionally, checking infeasibility and the cache before function evaluation, contributes for the reduction in the number of points to be evaluated at the poll step, often resulting in a much smaller number of parallel function evaluations than the maximum of 80, corresponding to polling for all the 5 iterate points. Figure 19 illustrates the number of function evaluations performed by each one of the workers, during a run with a total of 32 workers.

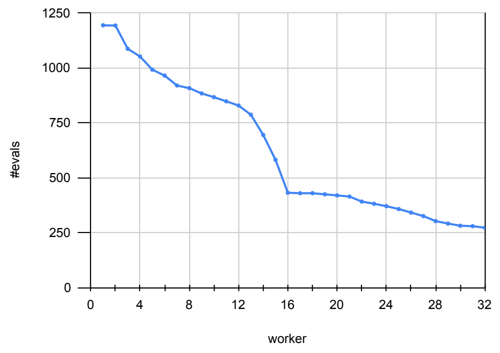


Figure 19.: Number of function evaluations performed by each one of the workers on a run of the styrene problem, considering a total of 32 workers.

While continuing to reduce the execution time, there are fewer opportunities to use more than 15 workers in this problem, as expected.

6. Conclusions

In this work we have proposed and analyzed the numerical performance of parallelization strategies for BoostDMS, an implementation of a Direct Multisearch method.

Additionally to the obvious parallelization of function evaluations at the poll step, strategies for parallelizing the function evaluations at the search step were also considered, stressing the benefits associated to the evaluation by levels performed in the sequential version of the code. The value of parallel strategies for models computation and minimization at the search step was also assessed. New strategies for the selection of the iterate point were proposed, based on modifications of the Γ spread metric, and considering the possibility of selecting several iterate points at a given iteration.

The best strategy, at each iteration selects $q + 2$ iterate points from the current approximation to the Pareto front, corresponding to the best value for each component of the objective function and to the two points defining the largest gap in a given component of the objective function, with cyclic changes between iterations. Function evaluation is always performed in parallel, respecting the levels approach of the search step.

This best version allows a remarkable improvement in computational time, with a significant improvement in purity and a slight improvement in hypervolume metrics. Conclusions hold both in an extensive academic test set and in a real engineering application. For other problems, with more components in the objective function or larger dimension, it is expected that the parallel version provides increased speedups.

BoostDMS is available by itself or as part of a Derivative-free Optimization Toolbox of solvers, suited for local and global single objective and multiobjective problems, implemented in Matlab and freely available for use at `https://docentes.fct.unl.pt/algb/pages/boostdfo`, under a GNU Lesser General Public License. The toolbox provides a GUI, which allows a non-expert user to take advantage of the different solvers options, including parallelization. For running BoostDMS, the Matlab toolboxes of Optimization and Parallel Computing are required for the minimization of the quadratic models at the search step and for distributing tasks to the different processors, respectively.

Funding

Support for authors was provided by national funds through FCT – Fundação para a Ciência e a Tecnologia I. P., under the scope of projects PTDC/MAT-APL/28400/2017, UIDB/00297/2020, and UIDP/00297/2020, the last two only for second and third authors.

Data availability

The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

Conflict of interest

The authors declare that they have no conflict of interest.

References

[1] R. Andreani, A.L. Custódio, and M. Raydan, *Using first-order information in direct multisearch for multiobjective optimization*, Optim. Methods Softw. (accepted for publication).

[2] C. Audet, V. Béchard, and S. Le Digabel, *Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search*, J. Global Optim. 41 (2008), pp. 299–318.

[3] C. Audet, J. Bigeon, D. Cartier, S. Le Digabel, and L. Salomon, *Performance indicators in multiobjective optimization*, European J. Oper. Res. 292 (2021), pp. 397–422.

[4] C. Audet, J.E. Dennis Jr., and S. Le Digabel, *Parallel space decomposition of the mesh adaptive direct search algorithm*, SIAM J. Optim. 19 (2008), pp. 1150–1170.

[5] C. Audet and W. Hare, *Derivative-Free and Blackbox Optimization*, Springer, Cham, Switzerland, 2017.

[6] C. Audet, G. Savard, and W. Zghal, *Multiobjective optimization through a series of single-objective formulations*, SIAM J. Optim. 19 (2008), pp. 188–210.

[7] C. Audet, G. Savard, and W. Zghal, *A mesh adaptive direct search algorithm for multiobjective optimization*, European J. Oper. Res. 204 (2010), pp. 545–556.

[8] S. Bandyopadhyay, S. Saha, U. Maulik, and K. Deb, *A simulated annealing-based multiobjective optimization algorithm: AMOSA*, IEEE Trans. Evol. Comp. 12 (2008), pp. 269–283.

[9] J. Bigeon, S. Le Digabel, and L. Salomon, *DMulti-MADS: mesh adaptive direct multisearch for bound-constrained blackbox multiobjective optimization*, Comput. Optim. Appl. 79 (2021), pp. 301–338.

[10] J. Branke, H. Schmeck, K. Deb, and M. Reddy S., *Parallelizing multi-objective evolutionary algorithms: cone separation*, in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, Portland, Oregon, USA. 2004, pp. 1952–1957.

[11] C.P. Brás and A.L. Custódio, *On the use of polynomial models in multiobjective directional direct search*, Comput. Optim. Appl. 77 (2020), pp. 897–918.

[12] B. Cao, J. Zhao, Z. Lv, and X. Liu, *A distributed parallel cooperative coevolutionary multiobjective evolutionary algorithm for large-scale optimization*, IEEE T. Ind. Inform. 13 (2017), pp. 2030–2038.

[13] G. Cocchi, G. Liuzzi, A. Papini, and M. Sciandrone, *An implicit filtering algorithm for derivative-free multiobjective optimization with box constraints*, Comput. Optim. Appl. 69 (2018), pp. 267–296.

[14] A.R. Conn, K. Scheinberg, and L.N. Vicente, *Introduction to Derivative-Free Optimization*, MOS-SIAM Series on Optimization, SIAM, Philadelphia, USA, 2009.

[15] A.L. Custódio, Y. Diouane, R. Garmanjani, and E. Riccietti, *Worst-case complexity bounds of directional direct-search methods for multiobjective derivative-free optimization*, J. Optim. Theory Appl. 188 (2021), pp. 73–93.

[16] A.L. Custódio, J.F.A. Madeira, A.I.F. Vaz, and L.N. Vicente, *Direct multisearch for multiobjective optimization*, SIAM J. Optim. 21 (2011), pp. 1109–1140.

[17] C. Davis, *Theory of positive linear dependence*, Amer. J. Math. 76 (1954), pp. 733–746.

[18] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, *A fast and elitist multiobjective genetic algorithm: NSGA-II*, IEEE T. Evolut. Comput. 6 (2002), pp. 182–197.

[19] K. Deb, P. Zope, and A. Jain, *Distributed computing of Pareto-optimal solutions with evolutionary algorithms*, in *Evolutionary Multi-Criterion Optimization*, C.M. Fonseca, P.J. Fleming, E. Zitzler, L. Thiele, and K. Deb, eds., Vol. 2632, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2003, pp. 534–549.

[20] J.E. Dennis Jr. and V. Torczon, *Direct search methods on parallel machines*, SIAM J.

Optim. 1 (1991), pp. 448–474.

[21] E.D. Dolan and J.J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program. 91 (2002), pp. 201–213.

[22] S. Fan and J. Chang, *A parallel particle swarm optimization algorithm for multi-objective optimization problems*, Eng. Opt. 41 (2009), pp. 673–697.

[23] D. Fioriti, S. Pintus, G. Lutzemberger, and D. Poli, *Economic multi-objective approach to design off-grid microgrids: A support for business decision making*, Renewable Energy 159 (2020), pp. 693–704.

[24] C.M. Fonseca, L. Paquete, and M. López-Ibánez, *An improved dimension-sweep algorithm for the hypervolume indicator*, in *Proceedings of the 2006 Congress on Evolutionary Computation (CEC'06)*, Vancouver, British Columbia, Canada. IEEE, 2006, pp. 1157–1163.

[25] J.D. Griffin and T.G. Kolda, *Asynchronous parallel hybrid optimization combining DIRECT and GSS*, Optim. Methods Softw. 25 (2010), pp. 797–817.

[26] J. He, A. Verstak, L.T. Watson, and M. Sosonkina, *Performance modeling and analysis of a massively parallel direct – part 1*, The Int. J. of High Perf. Comp. Appl. 23 (2009), pp. 14–28.

[27] M. Herlihy and N. Shavit, *The Art of Multiprocessor Programming*, Morgan Kaufmann, Waltham, USA, 2012.

[28] P.D. Hough, T.G. Kolda, and V.J. Torczon, *Asynchronous parallel pattern search for nonlinear optimization*, SIAM J. Sci. Comput. 23 (2001), pp. 134–156.

[29] A. Li, B. Xue, and M. Zhang, *Multi-objective feature selection using hybridization of a genetic algorithm and direct multisearch for key quality characteristic selection*, Inf. Sci. 523 (2020), pp. 245–265.

[30] G. Liuzzi, S. Lucidi, and F. Rinaldi, *A derivative-free approach to constrained multiobjective nonsmooth optimization*, SIAM J. Optim. 26 (2016), pp. 2744–2774.

[31] J.J. Moré and S.M. Wild, *Benchmarking derivative-free optimization algorithms*, SIAM J. Optim. 20 (2009), pp. 172–191.

[32] D.A. Pereira, T.P. Sales, and D.A. Rade, *Multi-objective frequency and damping optimization of tow-steered composite laminates*, Composite Struct. 256 (2021), p. 112932.

[33] E.G. Talbi, S. Mostaghim, T. Okabe, H. Ishibuchi, G. Rudolph, and C.A. Coello Coello, *Parallel approaches for multiobjective optimization*, in *Multiobjective Optimization*, J. Branke, K. Deb, K. Miettinen, and R. Slowinski, eds., Vol. 5252, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2008, pp. 349–372.

[34] S. Tavares, *Contributions to the development of an integrated toolbox of solvers in derivative-free optimization*, Master's thesis, NOVA School of Science and Technology, Portugal, 2020.

[35] A.I.F. Vaz and L.N. Vicente, *A particle swarm pattern search method for bound constrained global optimization*, J. Global Optim. 39 (2007), pp. 197–219.

[36] L.T. Watson and C.A. Baker, *A fully-distributed parallel global search algorithm*, Engineering Computations 18 (2001), pp. 155–169.

[37] Y. Zhou and Y. Tan, *GPU-based parallel multi-objective particle swarm optimization*, Int. J. Artif. Intell. 7 (2011), pp. 125–141.

[38] E. Zitzler and L. Thiele, *Multiobjective optimization using evolutionary algorithms – A comparative case study*, in *Parallel Problem Solving from Nature - PPSN V: 5th International Conference*, A.E. Eiben, T. Bäck, M. Schoenauer, and H.P. Schwefel, eds., Vol. 1498, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, Germany, 1998, pp. 292–301.

[39] E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, and V.G. Fonseca, *Performance assessment of multiobjective optimizers: An analysis and review*, IEEE T. Evolut. Comput. 7 (2003), pp. 117–132.

Appendix A. Pareto front approximations for the styrene problem

Table A1.: Pareto front approximations for the styrene problem.

	Parallel BoostDMS			**BoostDMS**			**DMS**		
	$f_1 \times 10^7$	$f_2 \times 10^{-3}$	f_3	$f_1 \times 10^7$	$f_2 \times 10^{-3}$	f_3	$f_1 \times 10^7$	$f_2 \times 10^{-3}$	f_3
1	-2.85742	-7.25873	-0.535667	-2.70948	-8.16182	-0.559667	-2.40819	-6.57919	-0.524333
2	-2.84934	-7.70017	-0.539000	-2.69909	-8.16348	-0.561000	-2.41940	-6.57843	-0.524000
3	-2.79146	-7.49654	-0.547333	-2.75351	-7.94175	-0.556000	-2.48618	-6.37774	-0.531667
4	-2.43936	-7.51873	-0.560667	-2.75541	-7.73039	-0.558667	-2.47461	-6.37855	-0.532000
5	-2.87686	-7.23766	-0.524333	-2.74743	-7.94314	-0.557000	-2.49659	-6.37693	-0.531333
6	-2.87516	-7.46022	-0.526000	-2.75918	-7.72988	-0.558333	-2.51036	-6.37613	-0.531000
7	-2.87513	-7.46137	-0.526667	-2.74593	-7.94360	-0.557333	-2.51569	-6.37451	-0.530333
8	-2.86833	-7.46997	-0.531667	-2.75030	-7.73089	-0.559000			
9	-2.79775	-7.71101	-0.546000	-2.76238	-7.72786	-0.557000			
10	-2.78734	-7.71870	-0.551000	-2.76766	-7.72583	-0.555667			
11	-2.83571	-7.70482	-0.542000	-2.77540	-7.72532	-0.555333			
12	-2.76913	-7.72176	-0.553000	-2.73043	-7.94682	-0.559667			
13	-2.84213	-7.48473	-0.540333	-2.72724	-7.94774	-0.560333			
14	-2.85250	-7.47964	-0.537333	-2.73064	-7.94636	-0.559333			
15	-2.82772	-7.48981	-0.543333	-2.71737	-7.94819	-0.560667			
16	-2.80691	-7.49037	-0.543667	-2.73988	-7.73291	-0.560333			
17	-2.79992	-7.49430	-0.546000	-2.71671	-7.94865	-0.561000			
18	-2.75274	-7.73039	-0.558667	-2.74400	-7.94590	-0.559000			
19	-2.85650	-7.47737	-0.536000	-2.74429	-7.94544	-0.558667			
20	-2.86128	-7.47453	-0.534333						
21	-2.86495	-7.47225	-0.533000						
22	-2.78353	-7.72074	-0.552333						
23	-2.46277	-7.73241	-0.560000						
24	-2.86781	-7.47111	-0.532333						
25	-2.43928	-7.51983	-0.561333						
26	-2.86808	-7.47054	-0.532000						
27	-2.91183	-7.43525	-0.511667						
28	-2.71484	-8.16182	-0.559667						
29	-2.91173	-7.43642	-0.512333						
30	-2.38609	-7.73594	-0.562333						
31	-2.74026	-8.15892	-0.557333						
32	-2.91120	-7.43701	-0.512667						
33	-2.74149	-8.15851	-0.557000						
34	-2.73450	-7.94636	-0.559333						
35	-2.74833	-7.94590	-0.559000						
36	-2.90686	-7.65524	-0.510667						
37	-2.41983	-7.73543	-0.562000						
38	-2.88295	-7.44867	-0.519333						
39	-2.42879	-7.73342	-0.560667						
40	-2.43603	-7.52038	-0.561667						
41	-2.77907	-7.93666	-0.552333						
42	-2.77852	-7.93713	-0.552667						
43	-2.88249	-7.45099	-0.520667						
44	-2.76797	-7.94314	-0.557000						