Les Cahiers du GERAD ISSN: 0711–2440

Parallel surrogate-based optimization using Mesh Adaptive Direct Search

B. Talgorn, S. Alarie, M. Kokkolaras

G–2020–38

July 2020

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Parallel surrogate-based optimization using Mesh Adaptive Direct Search

Bastien Talgorn [a]

Stéphane Alarie [b,c]

Michael Kokkolaras [b,d]

[a] *Intel Corporation, Programmable Solutions Group*

[b] *GERAD, Montréal (Québec), Canada, H3T 2A7*

[c] *Institut de recherche d'Hydro-Québec, Varennes (Québec) Canada, J3X 1S1*

[d] *Département de génie mécanique, Université McGill, Montréal (Québec), Canada, H3A 0C3*

alarie.stephane@hydroquebec.com
michael.kokkolarase@mcgill.ca

July 2020
Les Cahiers du GERAD
G–2020–38

1 Introduction

We consider the optimization problem

$$\begin{array}{ll} \min\limits_{\mathbf{x}\in\mathcal{X}} & f(\mathbf{x}) \\ \text{subject to} & c_j(\mathbf{x}) \leq 0, \quad j = 1, 2, \ldots, m, \end{array} \tag{P}$$

where $f(\mathbf{x})$ is the objective function, $\mathbf{x} \in \mathbb{R}^n$ is the vector of decision variables, $\mathcal{X}$ is a subset of $\mathbb{R}^n$, and $c_j(\mathbf{x})$ are general nonlinear constraints. We assume that some (at least one) of the functions $\{f, c_1, c_2, \ldots, c_m\}$ are evaluated using simulations or other computational procedures that are black-boxes. In particular, we consider that the objective and/or some of the constraints are computationally expensive, possibly nonsmooth, and/or nonconvex and that the process used to evaluate them may crash or fail to return a value. Moreover, we assume that the function gradients do not exist, are not available, or cannot be approximated in practice with resonable computational effort.

Metaheuristics and derivative-free search techniques are commonly used for solving (P). The former (e.g., genetic algorithms(GAs), particle swarm optimization (PSO), tabu search (TS), etc.) are commonly used for global exploration while the latter (e.g., generalized pattern search (GPS), mesh adaptive direct search (MADS) and trust-region methods (DFO, COBYLA, CONDOR)) have are local methods with conergence properties [22]. We choose to solve (P) using the MADS algorithm [5], by means of its NOMAD implementation [8, 23].

Multiprocessor computers, supercomputers, cloud computing, or just a few connected PCs, can provide parallel (or concurrent) computing opportunities to speed up so-called trajectory-based optimization algorithms that consider multiple points during an iteration. According to [1], three ways are commonly used to achieve this, namely: (i) parallel evaluation of neighborhood solutions (distributed evaluations), (ii) parallel trajectories from the same (or different) initial guess(es) (independent optimization runs), (iii) the evaluation of a point $\mathbf{x}$ is done in parallel (the search is sequential). Item (iii) depends only on the blackbox, while the other two are related to the search method. Both are implemented in NOMAD through p-MADS for (i) and Coop-MADS for (ii) [24]. In both cases, the parallel evaluations are handled by NOMAD using MPI calls to the blackbox [23]. However, if one prefers to take in charge the distribution of evaluations, one can implement p-MADS with the blocks by using NOMAD in batch mode [24]. Then, instead of doing MPI calls, NOMAD writes the set of $\mathbf{x}$ to be evaluated in a text file (one $\mathbf{x}$ by line) and waits that one proceeds to the evaluations and writes the values of $f(\mathbf{x})$ and $c_j(\mathbf{x})$ in a second text file (all the values on the line corresponding to $\mathbf{x}$) before to return the control to NOMAD. Then, NOMAD may decide to stop if a local optimum is reached or to submit a new block of $\mathbf{x}$ to pursue. Because of its flexibility and the resulting generality, the block functionality of NOMAD is here used to solve (P), and by hence, we opt for (i) as parallel strategy.

MADS is a two-step method, namely the SEARCH and the POLL. The SEARCH step is flexible and tries to identify a new $\mathbf{x} \in \mathcal{X}$ that improves the current best solution. The POLL step is well defined (somewhat rigid), at the basis of the convergence analysis and generates, according to a given pattern, trial points around the current best solution. The number of trial points at each iteration is typically $2n$ or $n + 1$ depending on the considered pattern (n being the size of $\mathbf{x}$). The number of points evaluated in the SEARCH step depends on the methods that are used. Several search methods are implemented in NOMAD, among which there are the speculative search, the quadratic model search (QUAD), the variable neighborhood search (VNS), the Latin-Hypercube search (LHS), the Nelder Mead search (NMS). One can implement its own search method if desired (called *user search*). Excepted LHS, the provided search methods usually return only one trial point. When several search methods are activated, they are called one after the other along the SEARCH step, each method providing its own trial point, which is evaluated by the blackbox before to proceed with the following search method. Then, assuming that $2n$ CPUs are rented for solving (P), the POLL step will naturally provide good uses of CPUs by the number of trial points it can generate at each iteration, but not the SEARCH step. According to the above, evaluations will momentarily become sequential, slowing down all progress by keeping almost all CPUs idle. One may argue we should then only use LHS since

it can systematically generate $2n$ trial points. It is true this will provide good uses of CPUs, but since LHS is based on randomness, its trial points will quickly become useless after few SEARCH steps, i.e., bringing no improvement to the current best solution, and so, also be a source of lost time.

Considering that the number of available CPUs are now relatively cheap and unlimited, particularly with the emergence of cloud computing, we should rethink the SEARCH step to be as effective as the POLL step in the use of CPUs. However, this must be done in a smart manner and we should avoid easy solutions based on randomness (such as LHS). We want that the trial points provided by the SEARCH step allow, at each iteration, to identify new potential optima and/or to accelerate the overall optimization process (by fulfilling the blocks, among others). In this work, we describe a model search method that can return for evaluation a large number of different and diverse trial points.

The paper is structured as follows. The general idea behind the proposed solution is given in Section 2. In Section 3, six different methods are presented for selecting various candidates from a large set of points. In Section 4, the resulting model search for parallel computing is completely specified. In Section 5, we test the model search methodology on five engineering design optimization problems with up to 64 processors. A short discussion concludes the paper.

2 Proposed solution

One of the drawbacks of the model search step as described in previous works [7, 27, 31, 32] is that each model search step only returns one candidate whereas a significant time is spent on solving the model. This implies that whatever the number of CPUs available, only one CPU is used in the SEARCH step for blackbox evaluations (excepted for LHS and the like). Here is our proposition for generating a large set of points among which the search candidates will be selected to be evaluated by the blackbox.

Let $(\hat{P})$ be the surrogate problem of the main problem (P). More specifically, $(\hat{P})$ is

$$\begin{array}{rl} \min\limits_{\mathbf{x}\in\mathcal{X}} & \hat{f}(\mathbf{x}) \\ \text{subject to} & \hat{c}_j(\mathbf{x}) \leq 0, \quad j = 1, 2, \ldots, m, \end{array} \tag{$\hat{P}$}$$

where $\{\hat{f}, \hat{c}_1, \hat{c}_2, \ldots, \hat{c}_m\}$ are surrogate models of $\{f, c_1, c_2, \ldots, c_m\}$. Surrogate models are chosen so that the evaluations of $\hat{f}(\mathbf{x})$ and $\hat{c}_j(\mathbf{x})$ are much more cheaper (for real life applications) than the evaluations of $f(\mathbf{x})$ and $c_j(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$. We want also that the minimizers of (P) and $(\hat{P})$ are the same (or close enough for assuming this). It then follows that a minimizer of $(\hat{P})$ will be a good candidate for the solution of (P).

If both problems have the same minimizers, we may assume this is also true for other features of $\mathcal{X}$. As for (P), one can solve $(\hat{P})$ with metaheuristics and/or derivative-free search methods. This means that the exploration of $\mathcal{X}$ will be made by following a trajectory, and that trajectory will go through different features of $\mathcal{X}$. Since the evaluations of $\hat{f}(\mathbf{x})$ and $\hat{c}_j(\mathbf{x})$ are rather cheap (compared to f and c_j), one can allow a very large budget of model evaluations (e.g. 10,000) to solve $(\hat{P})$, and as a result, to extend the number of features that will be visited. This is acceptable as long as the solution of $(\hat{P})$ remains faster than any single evaluation of $\mathbf{x}$ with the blackbox functions $\{f, c_1, c_2, \ldots, c_m\}$. Considering there are q CPUs available for blackbox evaluations, one may then select q points from the 10,000 visited by solving $(\hat{P})$. The q points can be selected to consider features that have been neglected until now in the solution of (P).

The above proposition involves an use of surrogate models $\{\hat{f}, \hat{c}_1, \hat{c}_2, \ldots, \hat{c}_m\}$ that is not reported in [18]. The authors mention two ways of exploiting surrogates in the context of high parallelization. The simplest is to fit q different surrogate models to the same points $\mathbf{x}$ already evaluated by the blackbox functions. This allows to get q different promising candidates and requires no uncertainty model for the surrogate models. One can also combined the individual surrogates in different way and still get q candidates from them. Distance-based criteria can be added on the top to ensure diversity between the candidates. The other way is to use one single surrogate model and find q points we

should evaluate with the blackboxes to improve its accuracy. This usually assumes that the surrogate has a given uncertainty model. Kriging is such a surrogate. The selection is then done to maximize the expected improvement (EI), to maximize the probability of improvement beyond a given target or to minimize the "bumpiness" in the surface response. Among them, EI is certainly the most popular, but solving exactly q-EI remains very expensive due to the Monte Carlo sampling it requires. Its solution is still an active area of research. Multiple heuristics and methods are proposed to mitigate that difficulty.

In our case, we propose an intermediate solution. Only one surrogate model is used for $\hat{f}$ and $\hat{c}_j$, instead of several. The q candidates are extracted from that unique surrogate, but not with the specific aim to improve it. This is left to its rebuilding from one iteration to the other. The q candidates are selected to be the most interesting to advance optimization. The way how to select the candidates is given next. It is based on distance criteria, no uncertainty model is involved.

3 Selection methods

Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ be the set of all the points evaluated by the blackbox up to now, i.e. for which the true values of functions $\{f, c_1, c_2, \ldots, c_m\}$ are known. Naturally, $\mathbf{X} \subset \mathcal{X} \subset \mathbb{R}^n$. We denote $\hat{\mathbf{X}}$ the *surrogate cache*, i.e. the set of all the points for which $\{\hat{f}, \hat{c}_1, \hat{c}_2, \ldots, \hat{c}_m\}$ have been evaluated during the resolution of ($\hat{P}$). As previously, $\hat{\mathbf{X}} \subset \mathcal{X} \subset \mathbb{R}^n$. Let $\mathbf{S}$ be the set of points that are selected by the SEARCH step to be evaluated with the blackbox. The set $\mathbf{S}$ is initialized as $\varnothing$ and is built from points of $\hat{\mathbf{X}}$ (ensuring that $\mathbf{S} \subset \hat{\mathbf{X}}$) with a greedy algorithm by use of up to six selection methods. Each selection method has a different goal:

- Method 1 selects the best point of $\hat{\mathbf{X}}$ not in $\mathbf{X} \cup \mathbf{S}$;
- Method 2 selects the most distant point of $\hat{\mathbf{X}}$ from $\mathbf{X} \cup \mathbf{S}$;
- Method 3 selects the best point of $\hat{\mathbf{X}}$ at a certain distance of $\mathbf{X} \cup \mathbf{S}$;
- Method 4 selects the best point of $\hat{\mathbf{X}}$ under additional feasibility constraints;
- Method 5 selects a point of $\hat{\mathbf{X}}$ that is a possible local minima of the surrogate problem ($\hat{P}$);
- Method 6 selects a point of $\hat{\mathbf{X}}$ in a non-explored area.

Note that some selection methods may fail to return a candidate, particularly Methods 3 and 4. If this happens, just pursue with the following method. Repeat and loop on the methods until you get enough points in $\mathbf{S}$ for the available CPUs. Some points $\mathbf{x} \in \hat{\mathbf{X}}$ can also be in the set $\mathbf{X}$. This is not an issue since all the methods only select point $\mathbf{x} \in \hat{\mathbf{X}}$ to be added to $\mathbf{S}$ if and only if $\mathbf{x} \notin \mathbf{X}$. The selection methods are detailed below after some definitions.

3.1 Definitions

We state here on some definitions before presenting each selection method in detail. Let $d(A, B)$ be the Euclidean distance between two subsets A and B of $\mathbb{R}^n$, i.e.

$$d(A, B) = \min_{a \in A} \min_{b \in B} \|a - b\|_2. \tag{1}$$

As a convention, the distance to an empty set is infinite: $d(A, \varnothing) = d(\varnothing, \varnothing) = +\infty$. By extension, we will denote the distance between an element $a \notin B$ and the subset B simply by $d(a, B)$, which implies that a also refers to the particular subset contening only a, i.e. $\{a\}$.

For the satisfaction of constraints, we consider the aggregate constraint violation function as in [15], i.e. $h(\mathbf{x}) = \sum_{j=1}^{m} \max\{0, c_j(\mathbf{x})\}^2$. This is the same function that is used in the *progressive barrier* mechanism in NOMAD [6]. Then, let define the order operators between two points $\mathbf{x}$ and $\mathbf{x}' \in \mathcal{X}$:

$$\mathbf{x} \prec \mathbf{x}' \Leftrightarrow \begin{cases} h(\mathbf{x}) < h(\mathbf{x}') \\ \text{or} \\ h(\mathbf{x}) = h(\mathbf{x}') \text{ and } f(\mathbf{x}) < f(\mathbf{x}'), \end{cases} \tag{2}$$

$$\mathbf{x} \preceq \mathbf{x}' \Leftrightarrow \texttt{not}(\mathbf{x}' \prec \mathbf{x}), \tag{3}$$

which are transitive. By those definitions, an incumbent solution $\mathbf{x}'$ of the main problem (P) is such that $\mathbf{x}' \preceq \mathbf{x}$, $\forall \mathbf{x} \in \mathbf{X}$ where $\mathbf{X}$ is as above, the set of all points evaluated up to now along the solution of (P). Similarly, a global minimizer $\mathbf{x}^*$ is such that $\mathbf{x}^* \preceq \mathbf{x}$, $\forall \mathbf{x} \in \mathcal{X}$. In the same manner as we define $\prec$ and $\preceq$ by use of f and h, we define $\widehat{\prec}$ and $\widehat{\preceq}$ by use of $\hat{f}$ and $\hat{h}$, the surrogate models of f and h.

Finally, to simplify the description of the proposed selection methods, we define $\mathbf{s}^\infty$ as a *virtual point* (in the sense that it does not have coordinates in $\mathbb{R}^n$ or that said they are irrelevant), which represents the worst possible candidate in $\mathbb{R}^n$:

$$\hat{h}(\mathbf{s}^\infty) = \hat{f}(\mathbf{s}^\infty) = +\infty \quad \text{and} \quad d(\mathbf{s}^\infty, \mathbf{X}) = 0. \tag{4}$$

3.2 Detailed descriptions

Method 1. The first selection method selects the best point $\mathbf{s}$ of $\hat{\mathbf{X}}$ under the constraint that $d(\mathbf{s}, \mathbf{X} \cup \mathbf{S}) > 0$, which means that $\mathbf{s}$ is not in the set $\mathbf{X}$ of evaluated points nor already selected (i.e. in $\mathbf{S}$). This is what is usually done with surrogate models for investigating new local optima.

Algorithm 1 Selection of the best point (Method 1)

$\mathbf{s}^* \leftarrow \mathbf{s}^\infty$
for all $\mathbf{s} \in \hat{\mathbf{X}}$, do:
  if $\mathbf{s} \,\widehat{\prec}\, \mathbf{s}^*$ and $d(\mathbf{s}, \mathbf{X} \cup \mathbf{S}) > 0$, then:
    $\mathbf{s}^* \leftarrow \mathbf{s}$
  end
end
if $\mathbf{s}^* \neq \mathbf{s}^\infty$, then:
  $\mathbf{S} \leftarrow \mathbf{S} \cup \{\mathbf{s}^*\}$
end

Method 2. The second method is about to bring a diversification of search candidates to be evaluated. It selects the point $\mathbf{s}$ of $\hat{\mathbf{X}}$ that maximizes the distance $d(\mathbf{s}, \mathbf{X} \cup \mathbf{S})$, i.e. as far as possible of points already evaluated. See Algorithm 2.

Algorithm 2 Selection of the most distant point to $\mathbf{X} \cup \mathbf{S}$ (Method 2)

$\mathbf{s}^* \leftarrow \mathbf{s}^\infty$
for all $\mathbf{s} \in \hat{\mathbf{X}}$, do:
  if $d(\mathbf{s}, \mathbf{X} \cup \mathbf{S}) > d(\mathbf{s}^*, \mathbf{X} \cup \mathbf{S})$, then:
    $\mathbf{s}^* \leftarrow \mathbf{s}$
  end
end
if $\mathbf{s}^* \neq \mathbf{s}^\infty$, then:
  $\mathbf{S} \leftarrow \mathbf{S} \cup \{\mathbf{s}^*\}$
end

Method 3. This method selects the best point $\mathbf{s}$ of $\hat{\mathbf{X}}$ under the constraint that $d(\mathbf{s}, \mathbf{X} \cup \mathbf{S}) \geq d_{\min}$, where $d_{\min}$ is initialized at 0 at the beginning of the selection process, and increased progressively as the method is applied. Method 3 may fail to select a candidate $\mathbf{s}$ when $d_{\min}$ becomes too large. Since the selected points $\mathbf{S}$ will be projected on the current mesh $\mathcal{M}$, incrementing $d_{\min}$ by the current mesh size $\Delta^{\mathcal{M}}$ allows to avoid that several candidates become identical after the projection. See Algorithm 3.

Algorithm 3 Selection of the best point with a constraint on the distance to $\mathbf{X} \cup \mathbf{S}$ (Method 3)

$\mathbf{s}^* \leftarrow \mathbf{s}^\infty$
if first use of Method 3, then:
  $d_{\min} \leftarrow 0$
end
for all $\mathbf{s} \in \hat{\mathbf{X}}$, do:
  if $\mathbf{s} \prec \mathbf{s}^*$ and $d(\mathbf{s}, \mathbf{X} \cup \mathbf{S}) \geq d_{\min}$, then:
    $\mathbf{s}^* \leftarrow \mathbf{s}$
  end
end
if $\mathbf{s}^* \neq \mathbf{s}^\infty$, then:
  $\mathbf{S} \leftarrow \mathbf{S} \cup \{\mathbf{s}^*\}$
  $d_{\min} \leftarrow d_{\min} + \Delta^{\mathcal{M}}$
end

Method 4. Considering that the surrogate models $\hat{c}_j$ may fail to predict correctly if $\mathbf{s}$ is feasible, the present method tries to select points that will effectively appear to be feasible when evaluated by the true functions c_j. This is done by selecting the best feasible point $\mathbf{s}$ of $\hat{\mathbf{X}}$ under the constraint $\hat{c}_{\max}(\mathbf{s}) \leq \hat{c}_{\text{margin}}$, where $\hat{c}_{\max}(\mathbf{s})$ is defined as being the most violated constraint of $\mathbf{s}$, i.e.

$$\hat{c}_{\max}(\mathbf{s}) = \max_{j=1,2,\ldots,m} \hat{c}_j(\mathbf{s}), \tag{5}$$

and $\hat{c}_{\text{margin}}$ is set as

$$\hat{c}_{\text{margin}} \leftarrow \max_{\substack{\mathbf{s} \in \hat{\mathbf{X}} \\ \hat{c}_{\max}(\mathbf{s}) < 0}} \hat{c}_{\max}(\mathbf{s}) \tag{6}$$

and quantifies, among all the feasible points of $\hat{\mathbf{X}}$, the smallest amount by which those are achieved. By definition, $\hat{c}_{\max}(\mathbf{s}) \leq 0$ if $\mathbf{s}$ is predicted to be feasible by the surrogate models. The more negative is $\hat{c}_{\max}(\mathbf{s})$, the more likely $\mathbf{s}$ will appear feasible when evaluated by the true functions, i.e. $c_j(\mathbf{s}) \leq 0$ for all $j = 1, 2, \ldots, m$. By decreasing progressively the value of $\hat{c}_{\text{margin}}$ after each call to Method 4, the selection method favours candidates that are increasingly likely to be feasible (but possibly with $\hat{f}$ increasing). See Algorithm 4. As previously, $\Delta^{\mathcal{M}}$ denotes the mesh size parameter.

To work properly, we need that $\hat{c}_{\text{margin}} \leq 0$ at any moment, and naturally, we assumed above that there is at least one $\mathbf{s} \in \hat{\mathbf{X}}$ that is feasible. If it is not the case, as this may happen in the first steps of the search, we will end up with $\hat{c}_{\text{margin}} > 0$ and an inappropriate candidate will be selected. To avoid this, we will initialize $\hat{c}_{\text{margin}}$ to 0. As a result, the method may fail to return a candidate.

Algorithm 4 Selection of the best point with a constraint on the feasibility (Method 4)

$\mathbf{s}^* \leftarrow \mathbf{s}^\infty$
if first use of Method 4, then:
  $\hat{c}_{\text{margin}} \leftarrow \min\{0, \max_{\substack{\mathbf{s} \in \hat{\mathbf{X}} \\ \hat{c}_{\max}(\mathbf{s}) < 0}} \hat{c}_{\max}(\mathbf{s})\}$
end
for all $\mathbf{s} \in \hat{\mathbf{X}}$, do:
  if $\hat{c}_{\max}(\mathbf{s}) \leq \hat{c}_{\text{margin}}$ and $\hat{f}(\mathbf{s}) < \hat{f}(\mathbf{s}^*)$ and $d(\mathbf{s}, \mathbf{X} \cup \mathbf{S}) > \Delta^{\mathcal{M}}$, then:
    $\mathbf{s}^* \leftarrow \mathbf{s}$
  end
end
if $\mathbf{s}^* \neq \mathbf{s}^\infty$, then:
  $\mathbf{S} \leftarrow \mathbf{S} \cup \{\mathbf{s}^*\}$
  $\hat{c}_{\text{margin}} \leftarrow 2\,\hat{c}_{\max}(\mathbf{s}^*)$
end

Method 5. The isolation distance is used here to detect local minima of the surrogate model. This concept is inspired from the *topographic isolation* of a mountain summit, which measures the local significance of a submit. The isolation of a submit is defined as the distance from the submit to the closest higher submit.[1]

Translated to optimization, the concept of topographic isolation is interesting to quantify the local importance of a minima. Its strict application is however impossible since it will require to prove that no other point within a certain distance of $\mathbf{x}$ is better than $\mathbf{x}$. We can only compute isolation distance based on the evaluated points. Consequently, we define the isolation distance as being the distance from $\mathbf{s}$ to the closest point of $\hat{\mathbf{X}}$ that is better than $\mathbf{s}$:

$$d_{\text{iso}}(\mathbf{s}) = \min_{\substack{\mathbf{s}' \in \hat{\mathbf{X}} \\ \mathbf{s}' \prec \mathbf{s}}} d\big(\mathbf{s}, \mathbf{s}'\big). \tag{7}$$

Constraints are taken into account by using the order relationship defined in Equation (2) for surrogate models. As a convention, if no point of $\hat{\mathbf{X}}$ is better than $\mathbf{s}$, then $d_{\text{iso}}(\mathbf{s}) = +\infty$. With this definition, the point of $\hat{\mathbf{X}}$ with the highest isolation distance is also the best candidate in $\hat{\mathbf{X}}$. However, we have observed that the other points with a high isolation distance are often poor points far from any other point of $\hat{\mathbf{X}}$. To address this problem, we define the *isolation number* of $\mathbf{s}$ as the number of points of $\hat{\mathbf{X}}$ within the ball of centre $\mathbf{s}$ and radius $d_{\text{iso}}(\mathbf{s})$:

$$n_{\text{iso}}(\mathbf{s}) = \text{card}\big\{\mathbf{s}' : \mathbf{s}' \in \hat{\mathbf{X}}, d(\mathbf{s}, \mathbf{s}') < d_{\text{iso}}(\mathbf{s})\big\}. \tag{8}$$

To have a high isolation number, a point must be better than many of its neighbors, which means that this criteria allows to detect local minima. Note that Equation (4) implies that $d_{\text{iso}}(\mathbf{s}^\infty) = n_{\text{iso}}(\mathbf{s}^\infty) = 0$. Method 5 selects the point of $\hat{\mathbf{X}}$ that has the highest isolation number not yet in $\mathbf{X} \cup \mathbf{S}$. See Algorithm 5.

Algorithm 5 Selection of the most isolated point (Method 5)

$\mathbf{s}^* \leftarrow \mathbf{s}^\infty$
for all $\mathbf{s} \in \hat{\mathbf{X}}$, do:
  if $n_{\text{iso}}(\mathbf{s}) > n_{\text{iso}}(\mathbf{s}^*)$ **and** $d(\mathbf{s}, \mathbf{X} \cup \mathbf{S}) > 0$, then:
    $\mathbf{s}^* \leftarrow \mathbf{s}$
  end
end
if $\mathbf{s}^* \neq \mathbf{s}^\infty$, then:
  $\mathbf{S} \leftarrow \mathbf{S} \cup \{\mathbf{s}^*\}$
end

Method 6. The purpose of this method is to select points in neglected areas. To do so, it selects points in areas heavily explored while solving ($\hat{P}$) but overlooked when solving (P). The *density number* is defined as:

$$n_{\text{density}}(\mathbf{s}) = \text{card}\big\{\mathbf{s}' : \mathbf{s}' \in \hat{\mathbf{X}}, d(\mathbf{s}, \mathbf{s}') < d(\mathbf{s}, \mathbf{X} \cup \mathbf{S})\big\}. \tag{9}$$

Method 6 then selects the point of $\hat{\mathbf{X}}$ with the highest density number. See Algorithm 6. As for n_{iso}, Equation (4) implies that $n_{\text{density}}(\mathbf{s}^\infty) = 0$.

[1] In mountaineering, the topographic isolation of Mount Everest is infinite and the summit with the second highest isolation is the Aconcagua in Argentina. The Aconcagua is not the second highest summit but there is no higher mountain in a 16,518 km range, making it the most important summit in the Americas and in the southern hemisphere.

Algorithm 6 Selection of a point in a populated area (Method 6)

$\mathbf{s}^* \leftarrow \mathbf{s}^\infty$
for all $\mathbf{s} \in \hat{\mathbf{X}}$, do:
  if $n_{\text{density}}(\mathbf{s}) > n_{\text{density}}(\mathbf{s}^*)$, then:
    $\mathbf{s}^* \leftarrow \mathbf{s}$
  end
end
if $\mathbf{s}^* \neq \mathbf{s}^\infty$, then:
  $\mathbf{S} \leftarrow \mathbf{S} \cup \{\mathbf{s}^*\}$
end

4 Model search for parallel computing

We now describe the way how we implement the proposed SEARCH step. We start with the surrogate models and follow with the algorithms used for solving ($\hat{P}$). We end with how all of this is integrated to the MADS algorithm to solve (P).

4.1 Surrogate models

Several surrogate models are mentioned in [18] about their use in parallel contexts. Among them, there are Kriging, radial basis functions (RBF), support vector regression (SVR) and polynomial response surfaces (PRS). For a more exhaustive description and comparison on the surrogate models, see [2]. Based on the work from [31], the locally weighted scatterplot smoothing (LOWESS) [10, 11, 12, 13] has been selected to be the surrogate model here.

LOWESS models generalize PRS and kernel smoothing (KS) models. PRS are good for small problems, but lose their efficiency when the problems are bigger and the functions to model are nonlinear or contain discrete variables [2]. KS models tend to overestimate lower values and to underestimate higher values, but usually predict correctly which has the smallest value between two points [7]. LOWESS models build a linear regression of kernel functions around the point to estimate. They have shown in [31] that they are suitable for surrogate-based optimization. Those models have the ability to interpolate and extrapolate accurately. Their parameters are chosen with an error metric denoted "aggregate order error with cross-validation" (AOECV), which favors equivalence between the main problem (P) and the surrogate problem ($\hat{P}$). We assumed in Section 2 there is such an equivalence between both problems.

For the experiments, we will use the implementation from SGTELIB [30], which is now integrated to NOMAD as surrogate library from version 3.8. More specifically, the considered LOWESS model is defined as follows in SGTELIB:

```
TYPE Lowess DEGREE 1 RIDGE 0 SHAPE_COEF OPTIM KERNEL_TYPE OPTIM
```

which means that the local regression is linear, the ridge (regularization) coefficient (see [31]) is 0, and the kernel shape and kernel type are optimized to minimize the aggregate order error.

The LOWESS model is built as described in Appendix A. Only the Gaussian kernel was considered in [31]. Six other kernel functions ϕ have been also implemented in SGTELIB for a total of seven. Then, not only λ (kernel shape) is chosen to minimize AOECV, but also ϕ (kernel type), and this, both together.

4.2 Surrogate problem solution

The surrogate problem ($\hat{P}$) is solved by means of an inner instance of MADS, which will be initialized by a Latin-Hypercube search (LHS) [25, 29], will use the variable neighborhood search (VNS) [19, 26]

as SEARCH step and will have a large number of model evaluations (10,000 in the current work). The POLL step will be performed with the ORTHO 2N directions. This inner MADS is implemented in the SEARCH step of the outer MADS, which is about solving (P), through the user search mechanism.

The LHS guarantees that there are surrogate cache points widely spread over the design space. For this, 30% of all model evaluations are devoted to the LHS (3,000 for the current work). Four other points are added to them, namely: The current best feasible point of the main problem (P), the current best infeasible point of (P), the best feasible point from the previous surrogate problem ($\hat{P}$) and the best infeasible point of the previous ($\hat{P}$). All of those points are used to initialize the inner MADS, which is started from the best of above points and will run until the remaining evaluation budget is exhausted. This budget will be shared between the POLL and the VNS in a default proportion where 75% of it will be devoted to the VNS. The VNS favours the exploration of multiple local attraction basins. The large number of evaluations, which defines the surrogate cache $\hat{\mathbf{X}}$, favours an accurate resolution of the surrogate problem. All of these elements (LHS, VNS and numerous evaluations) ensure that $\hat{\mathbf{X}}$ contains interesting candidates for the resolution of the main problem (P).

4.3 Resulting (outer) MADS algorithm

We recall that MADS is an iterative algorithm. At each iteration, MADS proceeds to a SEARCH step (first) and a POLL step (next). Let t be the index of iterations, and more specifically, it will denote the current one after t iterations ($t = 0, 1, 2, \ldots$). Then, $\mathbf{X}_t$, $\hat{\mathbf{X}}_t$ and $\mathbf{S}_t$ denote the sets $\mathbf{X}$, $\hat{\mathbf{X}}$ and $\mathbf{S}$ (from Section 3) that are considered by MADS at the iteration t. Let q be the number of CPUs available, i.e. the number of blackbox evaluations that can be proceeded simultaneously. The proposed MADS for exploiting effeciently q CPUs is as follows.

First, the SEARCH step proceeds by solving the current problem ($\hat{P}$) as above to constitute the set $\hat{\mathbf{X}}_t$. From that set, q search candidates are selected and returned to be evaluated in parallel. The selection is made by cycling through a user-defined subset of the six proposed selection methods (Section 3.2) until a total of q candidates are selected, or until all selection methods consecutively failed to add a candidate to $\mathbf{S}_t$. If q is smaller than the number of selection methods retained by the user, we do not necessarily go through all the methods, but stop as soon as we get q candidates. The process is summarized in Algorithm 7.

The POLL step is next run (if the SEARCH one fails) with the ORTHO 2N directions, which means that a minimum of $2n$ directions are generated at each POLL step. Let $\mathbf{P}_t$ be the set of candidates produced by the polling directions at the iteration t. The cardinality of $\mathbf{P}_t$ is denoted by $|\mathbf{P}_t|$. To be consistent with our need to fulfill continuously all the available CPUs with evaluations, additional candidates are then added so that $|\mathbf{P}_t|$ is at least q or a multiple of q. This is done with the intensification mechanism of NOMAD, and more specifically, by using ORTHO 1. For the case where $|\mathbf{P}_t| = q$, all the poll candidates of $\mathbf{P}_t$ are then evaluated simultaneously, and hence, there is no need for ordering them. In contrast, if $|\mathbf{P}_t| > q$, then the points of $\mathbf{P}_t$ are regrouped in several blocks of q candidates. The blocks are then evaluated sequentially and opportunistically, which means that if a block evaluation leads to a success, the following blocks are not evaluated. To increase the probability of success from the first block, and hence to avoid to proceed with the following ones, the candidates of $\mathbf{P}_t$ are sorted using the surrogate models $\{\hat{f}, \hat{c}_1, \hat{c}_2, \ldots, \hat{c}_m\}$ and distributed in the blocks so that the more promising ones are in the first block. The LOWESS model can be used here to achieve this.

When some points are provided to MADS as starting points, they are all evaluated first (and only them, whatever the value of q). Since this is considered as being a SEARCH step, the next (and first) points to be generated by MADS will be done according to the POLL step. This means that if one provides only one starting point, it will be evaluated alone first, followed by the above POLL step. The SEARCH step for generating q interesting candidates will come only after.

Algorithm 7 The proposed MADS optimization algorithm.

[1] Initialization
$t \leftarrow 0$
Set initial poll and mesh sizes $\Delta_0^{\mathrm{P}} \geq \Delta_0^{\mathcal{M}} > 0$
Initialize $\mathbf{X}_0$ with starting points
Evaluate $\{f(\mathbf{x}), c_1(\mathbf{x}), c_2(\mathbf{x}), \ldots, c_m(\mathbf{x})\}\ \forall \mathbf{x} \in \mathbf{X}_0$
[2] Model search
Use $\mathbf{X}_t$ to build $\hat{f}$ and $\{\hat{c}_j\}_{j \in J}$
Solve surrogate problem ($\hat{P}$) using the inner MADS instance
$\hat{\mathbf{X}}_t \leftarrow$ Set of points evaluated with surrogate model while solving ($\hat{P}$)
$\mathbf{S}_t \leftarrow$ Cycle through selection steps to select q points of $\hat{\mathbf{X}}_t$
$\mathbf{S}_t \leftarrow$ Projection of the points of $\mathbf{S}_t$ onto mesh $\mathcal{M}_t$
Parallel evaluation of $\{f(\mathbf{x}), c_1(\mathbf{x}), c_2(\mathbf{x}), \ldots, c_m(\mathbf{x})\}\ \forall \mathbf{x} \in \mathbf{S}_t$
If success, `goto` **[4]**
[3] Poll
Build poll set $\mathbf{P}_t$
Sort $\mathbf{P}_t$ according to $\hat{f}$ and $\{\hat{c}_j\}_{j \in J}$
Parallel evaluation of $\{f(\mathbf{x}), c_1(\mathbf{x}), c_2(\mathbf{x}), \ldots, c_m(\mathbf{x})\}\ \forall \mathbf{x} \in \mathbf{P}_t$
[4] Updates
$t \leftarrow t + 1$
Update $\Delta_t^{\mathcal{M}}$, Δ_t^{P}, $\mathbf{x}_t^*$ and $\mathbf{X}_t$
If no stopping condition is met, `goto` **[2]**

We recall that $\mathcal{M}_t$ is the current mesh (the one prevailing at iteration t), $\Delta_t^{\mathcal{M}}$ is the associated mesh size parameter, Δ_t^{P} the corresponding mesh poll parameter and $\mathbf{x}_t^*$ the best solution found at iteration t from $t = 0$. Since the best solution found evolves over the iterations, we prefer to use $\mathbf{x}_t^*$ for denoting this instead of simply $\mathbf{x}^*$. Finally, the set $\mathbf{X}_t$ is updated with all the points $\mathbf{x}$ in $\mathbf{S}_{t-1}$ and $\mathbf{P}_{t-1}$ that have been evaluated during the previous iteration.

5 Tests on five engineering design optimization problems

The proposed model search methodology is here tested on five optimization problems with the algorithm presented in the previous section. We start by describing the solvers considered for the current benchmarking. Next, the five engineering design problems are presented. Numerical resultats will follow.

5.1 Compared algorithms

Five solvers are compared through the numerical experiments. All those are based on MADS algorithm and implemented using NOMAD 3.8 [23]. Doing so, we ensure to avoid coding biases since a same feature will be identical between two solvers. The considered solvers are described below:

- **MADS.** Refers to the POLL step of MADS, without any SEARCH step, where $2n$ directions are generated on the grid with the usual method (ORTHO) and evaluated in parallel. If needed, k additional directions are generated such that $2n + k$ is a multiple of q.
- **Multi-Start.** Consists of q parallel runs of MADS. They are totally independent and each instance runs on its own CPU. Each instance proceeds to its evaluations sequentially, one after the other. Only the POLL step is executed. No SEARCH step, no cache is shared between running instances.
- **LH Search.** The "MADS" solver above with a Latin-Hypercube search (LHS) for the SEARCH step, where q candidates are generated with its own LH sampling and evaluated next in parallel.
- **Lowess-A.** The "MADS" solver above with the proposed surrogate model search for the SEARCH step. The q search candidates are selected cycling through Methods 1 and 2, and next, all evaluated in parallel.

- **Lowess-B.** The "MADS" solver above with the proposed surrogate model search for the SEARCH step. The q search candidates are selected cycling through Methods 3, 4, 5 and 6, and as previously, all evaluated in parallel.

Both LOWESS solvers are exactly like Algorithm 7, excepted for the used selection methods. The only difference between them and the "LH Search" solver is the model search that is replaced by a LHS in the SEARCH step. By this, we should be able to estimate if model searches have some benefits over a random search. The "MADS" solver is introduced as baseline and the three previous solvers should do better since it has no SEARCH step. Finally, the "Multi-Start" solver is added to ensure that one should not proceed with q independent narrow trajectories instead of one single trajectory having q wide evaluations (that is the four other solvers).

5.2 Engineering design optimization problems

The above solvers are compared on five engineering design application problems. A short description follows below for each problem. For more details, see Appendix B.

- **TCSD.** The Tension/Compression Spring Design problem [3, 9, 16] consists of minimizing the weight of a spring under mechanical constraints. This problem has three variables and four constraints. The design variables define the geometry of the spring. The constraints concern shear stress, surge frequency and minimum deflection.
- **Vessel.** This problem [16, 21] consists of designing a compressed air storage tank and has four design variables and four constraints. The variables define the geometry of the tank and the constraints are related to the volume, pressure and solidity of the tank. The objective consists of minimizing the total cost of the tank, including material and labour.
- **Welded.** The welded beam design problem (Version I) [16, 28] has four variables and six constraints. It consists of minimizing the construction cost of a beam, under shear stress, bending stress, load and deflection constraints. The design variables define the geometry and the characteristics of the welded joint.
- **Solar 1.** This optimization problem consists of maximizing the energy received over a period of 24 hours under five constraints on budget and heliostat field area [17]. This problem has nine variables, among which, one is integer and has no upper bound.
- **Solar 7.** The problem consists of maximizing the efficiency of the receiver over a period of 24 hours, for a given heliostats field, under six binary constraints [17]. This problem has seven design variables, among which, one is also integer and has no upper bound. The objective function is the energy transferred to the molten salt.

Progressive barriers will be used to deal with the constraints [6]. Maximization problems will be transformed into minimization by multiplying the objective functions by -1. The three first problems are the easiest ones. However, it is difficult to find feasible solution for TCSD. The two other problems, which are related with solar farm designs, are the difficult ones. Among all the considered problems, Solar 1 is certainly the most difficult.

5.3 Numerical experiments

We compare the efficiency of each solver for different values of block size $q \in \{1, 2, 4, 8, 16, 32, 64\}$. As an example, we will use "Lowess-A 16" to refer to the solver that relies on LOWESS models cycling over Methods 1 and 2 considering a block size $q = 16$. For each problem, we generated 50 sets of 64 starting points with Latin-Hypercube sampling [25]. For all solvers other than "Multi-Start", only the first point of each set is used to perform optimizations. Doing so, we get 50 runs from the same starting points for each solver, each problem and each value of q. For "Multi-Start", since q independent and parallel sequential runs of MADS must be performed, we will used then the q first points of each set. Doing so, we still get 50 runs for each problem and each q, but all first starting points are in common with the other solvers.

To avoid that all the "LH Search" runs end up with nearly identical solutions for a given q, we told to NOMAD to use a random seed for initializing its LHS. This ensures that each LHS performed at each iteration is completely unrelated with any other LHS (from the same run or not, with the same q or not).

For the three simpler problems (TCSD, Vessel and Welded), a budget of 100 block evaluations is allocated. For the two difficult problems (Solar 1 and Solar 7), the budget is increased to 200 block evaluations. This means that, for a given problem, all solvers will have the same "wall-clock" time, but not necessarily the same resources (number of CPUs available for block evaluations) nor the same total number of blackbox evaluations.

Solution quality

Figures 1 and 2 represent the distribution of the final objective function over the 50 runs, and this, for each problem, each solver and each block size q. The minimum and maximum objective values that we obtained from the runs are indicated by circles in the figures. Lower and upper quantiles are delimited by boxes. Median value is represented by a bar into the boxes. More a distribution is on the left side, better is the combination of solver and q. Since we are mostly interested by the best combinations, the figures only focus on the smallest values. Otherwise, it will be difficult to appreciate and see the difference between the best combinations. All combinations for which the distribution is cut on the right side are irrelevant.

On the three simpler problems (TCSD, Vessel and Welded, Figure 1), the LOWESS solvers (and in particular "Lowess-B") are by far superior to the solvers that do not rely on model search. For the TCSD problem, the "MADS" solver often failed to find a feasible design (thus leading to infinite objective values), even with a large number of evaluations per block. The four other solvers always managed to find a feasible point on at least 75% of the runs. On this problem, the "Lowess-B" solver performs better than any of the other ones. We see that "Lowess-A 64" is outperformed by "Lowess-B 8". As the TCSD problem is very constrained, final objective function is very dependent on the starting point. This is why the "Multi-Start" solver performs quite well on this problem.

The same trend is observed for the Vessel and Welded problems: "Lowess-B" better than "Lowess-A" better than "LH" or "MADS". In particular, "Lowess-B 8" outperforms the solvers "Lowess-A 8/16/32". As expected, increasing the block size improves the performance. However, for the "Lowess-B" solver, these three problems are "too easy" and it is difficult to see the advantage of using parallel computing because the global optimum is found most of the time within 100 block evaluations for block size of 16 or more. The "Multi-Start" solver performs rather poorly on these two problems.

The numerical results generally follow the same trend on the two Solar problems (Figure 2). With an equal block size, the LOWESS models outperform the other solvers and "Lowess-B" outperform "Lowess-A".

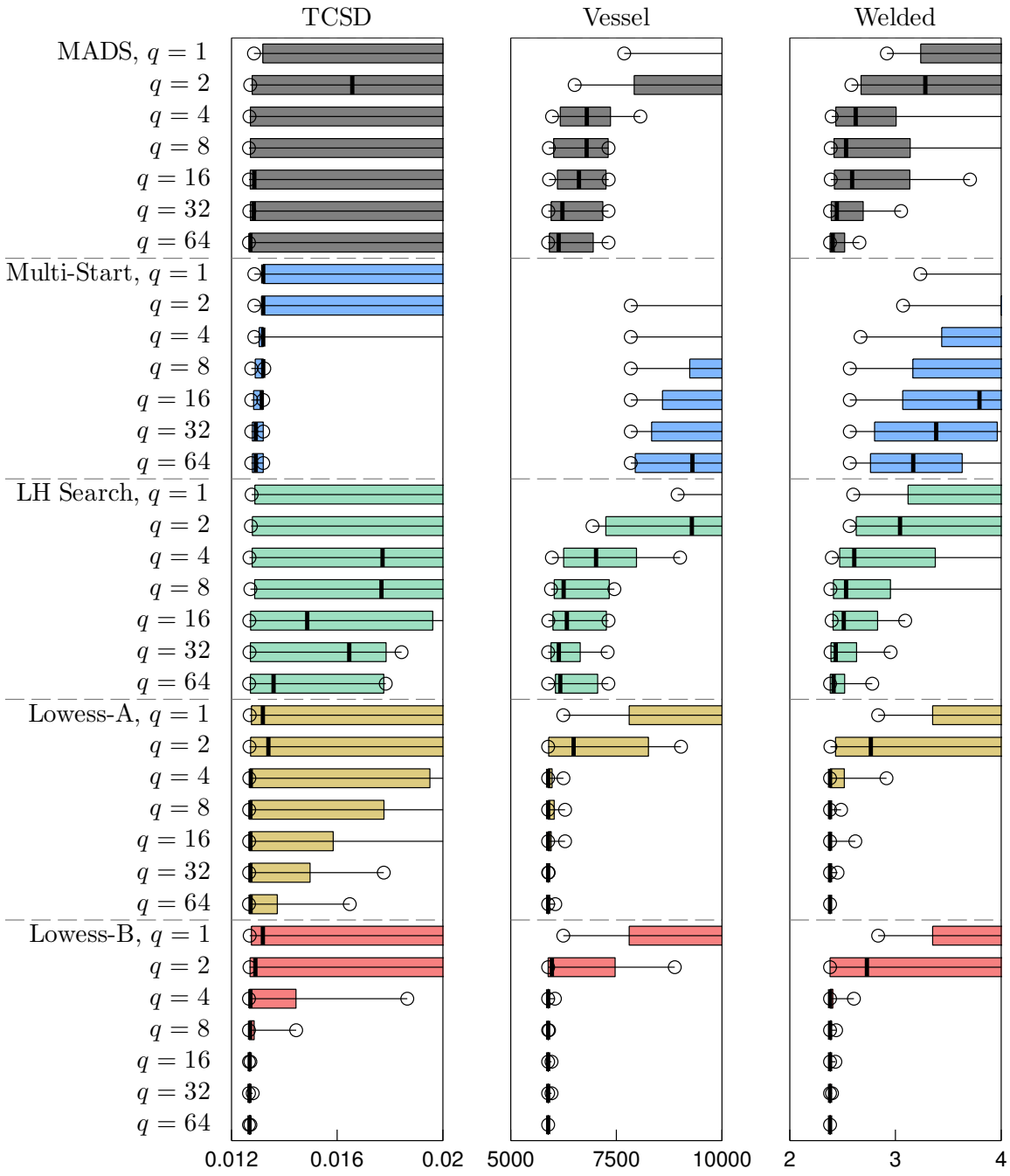

Figure 1: Summary of the final performance for the TCSD, Vessel and Welded problems over 50 runs.

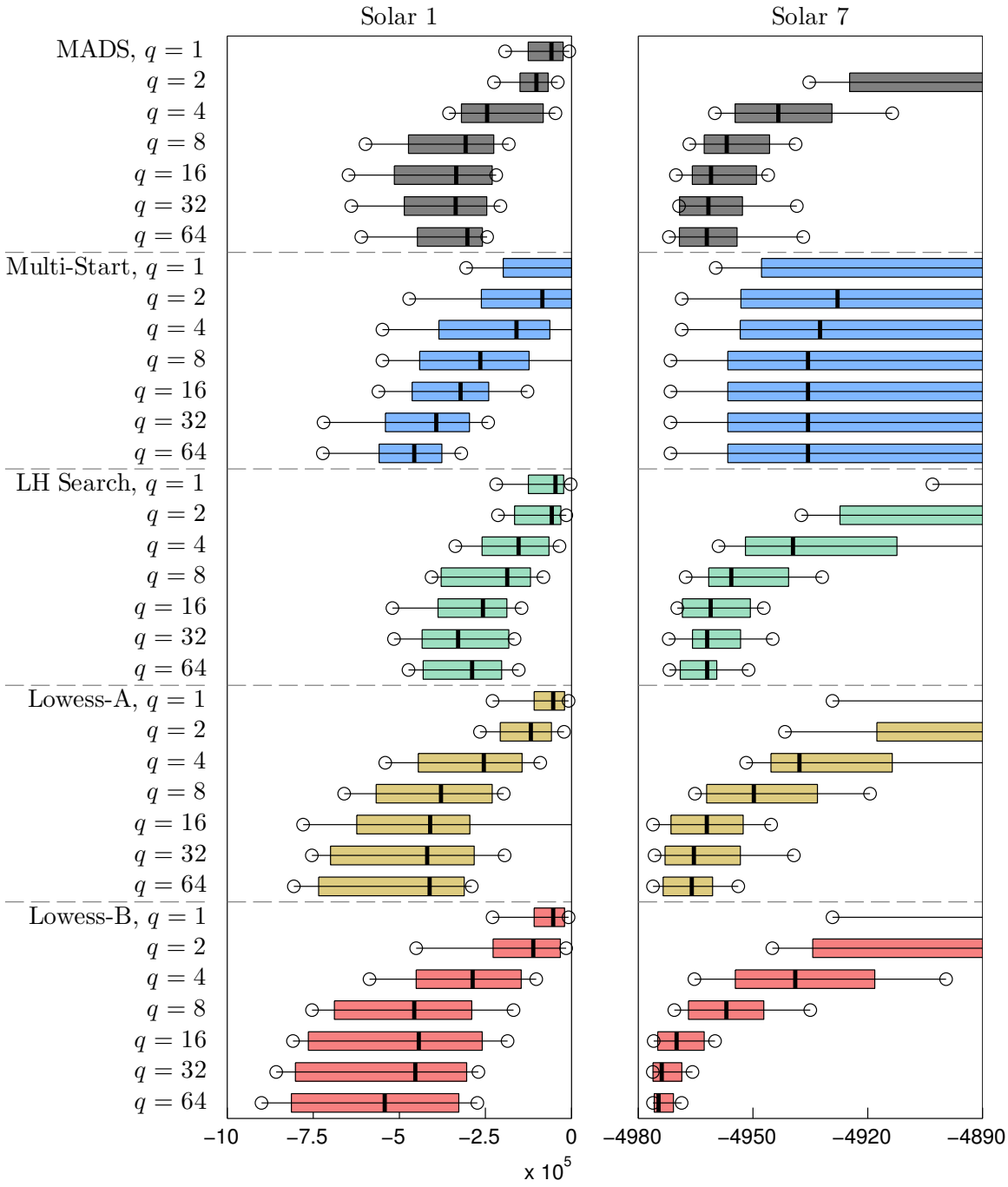

Figure 2: Summary of the final performance for the Solar 1 and Solar 7 problems over 50 runs.

Convergence rates

Let's now consider the rate at which the solvers decrease the objective function. We focus on the best above results, i.e. "Lowess-B" and $q = 64$. Figures 3 and 4 show the evolution of the median objective function of 50 runs depending on the number of block evaluations. For each problem, the left subplot compares the five solvers with blocks of size $q = 64$. The right subplot compares the convergence of "Lowess-B" for block sizes ranging from $q = 1$ to 64. In all subplots, lower and below is a curve, better is the solver.

From the left subplots, it appears that not only "Lowess-B" provides the better solutions (as reported above), it is also faster than any other solver (for $q = 64$). Depending on the problem, the worst solver is "Multi-Start" (TCSD and Solar 1) or "LH Search" (Vessel, Welded and Solar 7). Although "MADS" is only based on the POLL step, it provides good performances in general, excepted on Solar 1 (but remains better than the worst solver). "Lowess-A" provides also good performances, but on some problems (TCSD, Solar 1 and Solar 7), it shows no particular advantage compared to the other solvers.

The right subplots show how q modulates the convergence rate of "Lowess-B". As expected, larger is q, faster is the convergence. However, depending on the problem, there is a saturation point beyond which an increase of q brings no more contribution. On easy problems (TCSD, Vessel and Welded), the saturation point arises with $q = 8$ or 16. On the most difficult one (Solar 1), all CPUs are contributory and q could be even larger than 64 that will be helpful. One can notice that the curve for $q = 1$ does not appear on three subplots (Vessel, Welded and Solar 7), among which (Solar 7), the curve for $q = 2$ should appear in the upper right corner but is hidden by the box containing the legend of the curves. This suggests that the apparent difficulty of problems increases when q is smaller.

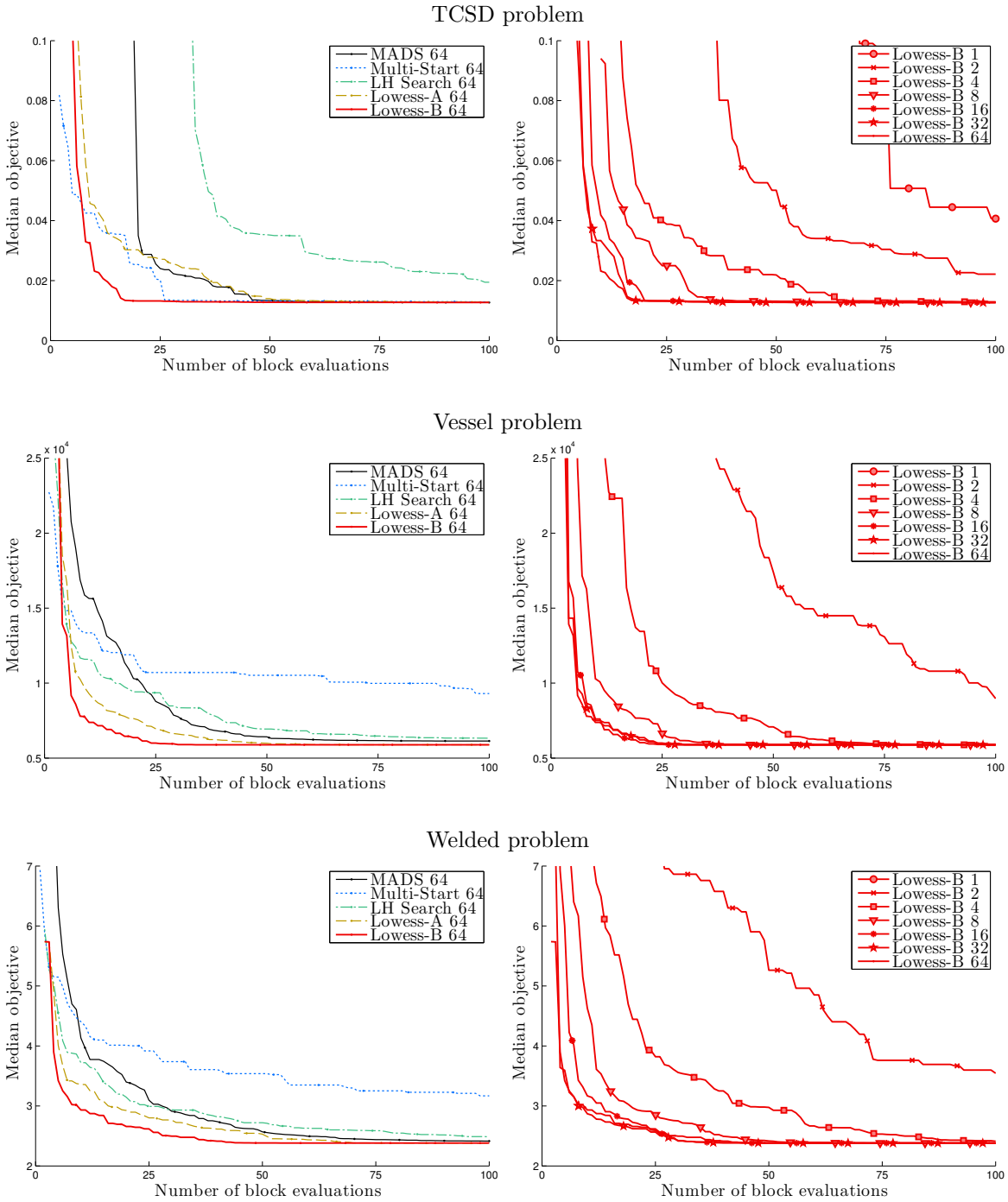

Figure 3: Results on the three simple problems: TCSD, Vessel and Welded. Median objective over 50 runs depending on the number of block evaluations.

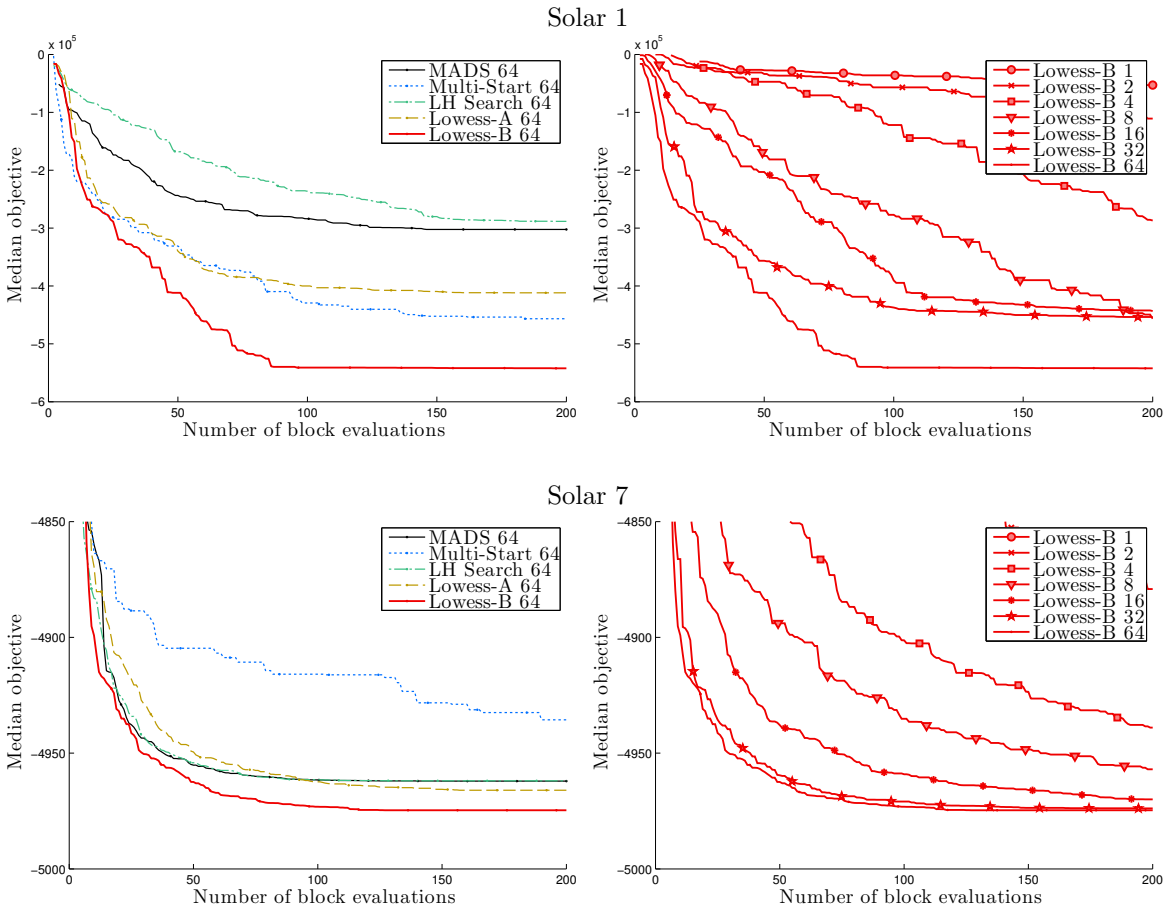

Figure 4: Results on the two difficult problems: Solar 1 and Solar 7. Median objective over 50 runs depending on the number of block evaluations.

Performance profiles

Consider now the global performance of solvers through performance profiles [14]. Performance profiles indicate, for each solver, the proportion of runs (%) that is solved at a given tolerance τ with some budgets. More specifically, for each solver s, each instance r and each problem p, we compute the number of block evaluations $b_{s,p,r}(\tau)$ such that

$$\frac{|f_{s,b,p,r} - f_p^*|}{|f_p^*|} \leq \tau \tag{10}$$

where f_p^* is the best known objective value for the problem p and $f_{s,b,p,r}$ is the value obtained with the solver s after b bloc evaluations on the instance r of problem p. Let $b_{p,r}^{\min}(\tau)$ be the smallest budget (whatever the solver) for solving the instance r of problem p at τ-precision, e.i.

$$b_{p,r}^{\min}(\tau) = \min_s b_{s,p,r}(\tau).$$

Then, for each solver s, we can plot the proportion of runs that satisfy (10) given α, which is defined as a multiple of the smallest budget, i.e. $\alpha\, b_{p,r}^{\min}(\tau)$ block evaluations. Figure 5 presents such performance profiles for $q = 64$ over the 50 runs (instances) of the five considered problems, and this, for four values of τ from 10^{-1} to 10^{-4}.

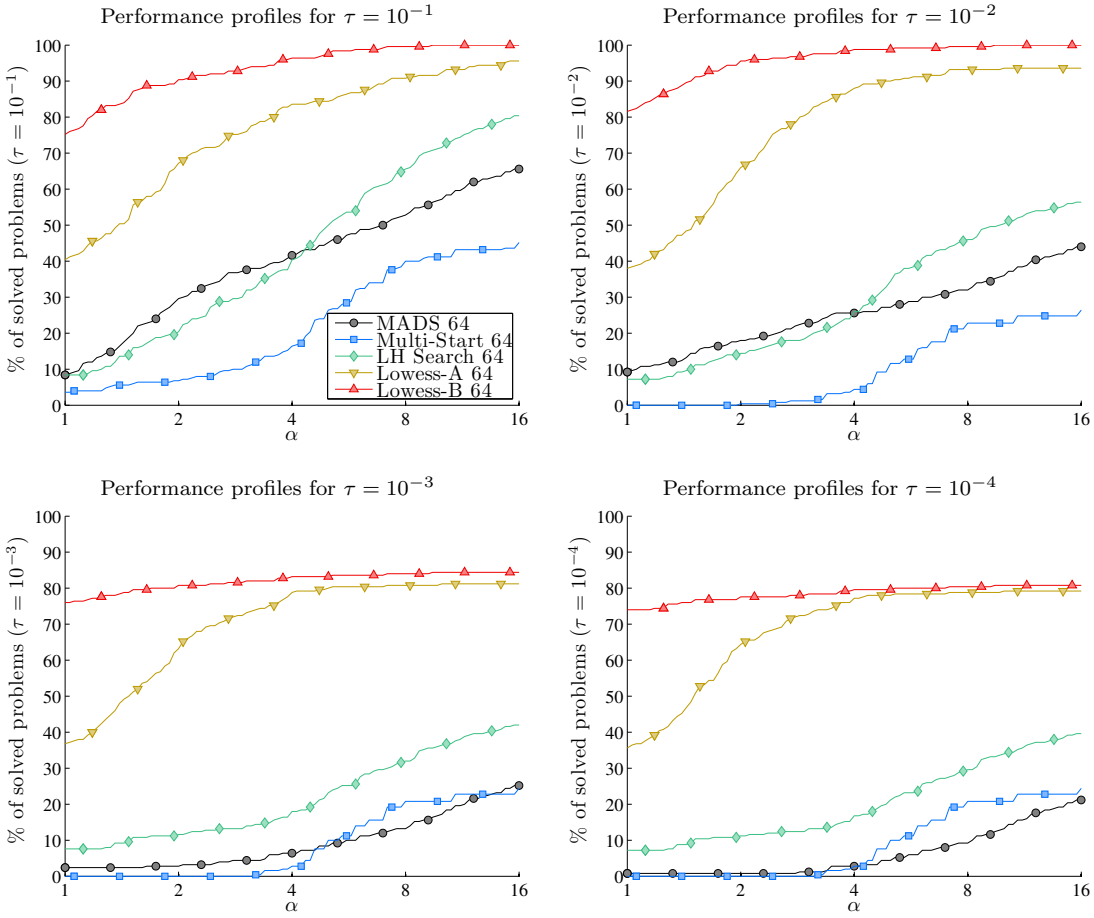

Figure 5: Performance profiles for $q = 64$ over the 50 runs of all five problems.

On performance profiles, higher is a curve, better is a solver. Moreover, a solver that performs well for small values of α is a solver that can solve, for the considered τ, a large number of problems within a small evaluation budget. Figure 5 confirms our previous observations, i.e. "Lowess-B" outperforms

all the solvers, followed by "Lowess-A" and "LH Search" in second and third position. "MADS" and "Multi-Start" are in the last position. For large tolerances, "MADS" does better than "Multi-Start", and even "LH Search" for small values of α ($\alpha \leq 4$). In the case of small tolerances, "MADS" is outperformed by the other solvers, restoring the expected order.

The most interesting thing about Figure 5 is the important gap between the performance curves of LOWESS solvers and the ones from the three other solvers. The gap quickly increases when the tolerance decreases. From moderate to lower tolerances ($\tau \leq 10^{-2}$), "Lowess-B" systematically solves twice more problems than "LH Search", "Multi-Start" and "MADS", if it is not three or four times more. It is unusual to observe such cleaver differences on performance profiles. "Lowess-A" tend to be as good as "Lowess-B", particularly when the tolerance is small ($\tau = 10^{-4}$), but needs four times more block evaluations ($\alpha \geq 4$) to achieve this.

Scalability analysis

We wish to establish the reduction of wall-clock time yield by the use of additional resources for each solver. For this, we follow the methodology proposed in [20]. Let's define $f_{s,b,p,r,q}$ as being the value of the objective function obtained by solver s after b block evaluations on instance r of problem p when using q CPUs. Then, for each solver s and each instance r of problem p, we define the *reference* objective value as the best value achieved with only one CPU ($q = 1$), i.e.

$$f^{\text{ref}}_{s,p,r} = \min_b f_{s,b,p,r,1}, \tag{11}$$

and $b^{\text{ref}}_{s,p,r,q}$ the number of block evaluations necessary to reach $f^{\text{ref}}_{s,p,r}$ when q CPUs are used, i.e.

$$b^{\text{ref}}_{s,p,r,q} = \min\{b : f^{\text{ref}}_{s,p,r} \leq f_{s,b,p,r,q}\}. \tag{12}$$

Let define the *speed-up* for solver s when solving with q CPUs as being

$$\text{speed-up}(s,q) = \underset{p,r}{\text{geomean}} \left(\frac{b^{\text{ref}}_{s,p,r,q}}{b^{\text{ref}}_{s,p,r,1}} \right) \tag{13}$$

and its *efficiency* as

$$\text{efficiency}(s,q) = \frac{\text{speed-up}(s,q)}{q}. \tag{14}$$

Speed-up and efficiency values from the numerical experiments are displayed in Figure 6.

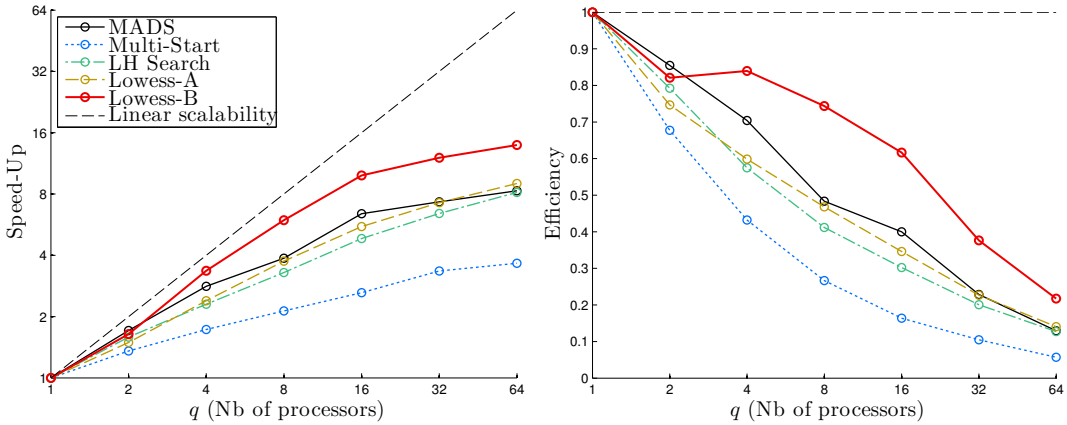

Figure 6: Speed-up and efficiency of the solvers depending on the number of processors.

Perfect scalability is obtained when the speed-up equal to q and the efficiency equal to 1. The speed-up curves show that the information brought by new CPUs decreases as their number increases.

This has been observed on Figures 3 and 4 where problems show saturation points around $q = 8$ and 16. "Lowess-B" has the best speed-up, which is followed by "MADS" (probably as a result of its POLL mechanism). "Lowess-A" and "LH Search" follow next close to "MADS". The last one is "Multi-Start", definitely showing that one has advantage to proceed with one search doing q parallel evaluations at each iteration than to proceed with q independent searches doing one single evaluation at the time (obviously, the worst way to exploit a bunch of CPUs).

The efficiency curves show a fast decrease at the beginning, which next reduces with q, excepted for "Lowess-B". The solver has a bump on its efficiency curve at $q = 4$. For $q = 2$, only Methods 3 and 4 are effectively solicited for generating candidates. With $q = 4$ (and greater), Methods 5 and 6 are also solicited for fulfilling the blocks. The bump shows the important contributions of these methods to the efficiency of "Lowess-B". Clearly, fulfilling the search blocks with various unevaluated potential optima and points from areas intensively explored in ($\hat{P}$) but ignored up to now solving (P) made a difference.

6 Conclusion

It seems that linear LOWESS models with optimized kernel shape and coefficient provide accurate representation of the blackbox outputs. The use of diverse selection methods (Methods 3 to 6) allows for a better exploration of the design space, for a better local convergence, and for a better use of additional CPU resources. Methods 5 and 6 are particularly efficient, outperforming the other considered selection methods. This means that the way how the surrogates are currently used for accelerating the searches (Method 1) is not so appropriate. Same conclusion about diversification strategies (Method 2): It is not enough to select points far from the ones already evaluated, they must also be distant between them.

Unfortunately, we cannot state which of Method 5 or 6 really allows to accelerate the search. We may assume the good performance of "Lowess-B" is due to Method 5, but we did not test both methods separately. This should be done to clarify the situation. Also, may be other combinations and/or selection methods could give good results. This is an open question.

The proposed selection methods are not specific to the LOWESS model considered here for the experiments, but are applicable to any other surrogate that can make (P) and ($\hat{P}$) equivalent. In particular, we believe they will work well with reduced-fidelity (or variable-fidelity) physical-based models since high- and low-fidelity models typically have similar structured solution domains. The selection methods are also applicable to other algorithms using surrogates (such as in [18]) to identify new potential good points to evaluate.

Appendix

A LOWESS predictions

As a convention, we denote $\boldsymbol{\xi} \in \mathcal{X} \subseteq \mathbb{R}^n$ the point of the design space where we want to predict the value of the blackbox outputs. Locally weighted scatterplot smoothing (LOWESS) models build a local linear regression at the point $\boldsymbol{\xi}$ where the blackbox outputs $[f\ c_1 \dots c_m]$ are to be estimated [4, 10, 11, 12, 13, 31]. This local regression emphasizes data points that are close to $\boldsymbol{\xi}$. The interested reader can refer to [31] for details about the method described below. In the current work, only local linear regressions are considered, while local quadratic regressions and Tikhonov regularisation are also considered in [31]. Moreover, only Gaussian kernel was considered in [31]. Six others are here added as kernel functions.

We define the output matrix $\mathbf{Y} \in \mathbb{R}^{p\times(m+1)}$, the design matrix $\mathbf{Z}_{\boldsymbol{\xi}} \in \mathbb{R}^{p\times(n+1)}$ and the weight matrix $\mathbf{W}_{\boldsymbol{\xi}} \in \mathbb{R}^{p\times p}$ such that

$$\mathbf{Y} = \begin{bmatrix} f(\mathbf{x}_1) & c_1(\mathbf{x}_1)\dots c_m(\mathbf{x}_1) \\ \vdots & \vdots \quad\quad \vdots \\ f(\mathbf{x}_p) & c_1(\mathbf{x}_p)\dots c_m(\mathbf{x}_p) \end{bmatrix}, \quad \mathbf{Z}_{\boldsymbol{\xi}} = \begin{bmatrix} 1 & (\mathbf{x}_1-\boldsymbol{\xi})^\top \\ \vdots & \vdots \\ 1 & (\mathbf{x}_p-\boldsymbol{\xi})^\top \end{bmatrix}, \quad \mathbf{W}_{\boldsymbol{\xi}} = \begin{bmatrix} w_1(\boldsymbol{\xi}) & & \\ & \ddots & \\ & & w_p(\boldsymbol{\xi})^\top \end{bmatrix}. \tag{15}$$

The details of the computation of $w_i(\boldsymbol{\xi})$ are described in Section A.1. Then, we define $\mathbf{u}_{\boldsymbol{\xi}} \in \mathbb{R}^{n+1}$ as the first column of $(\mathbf{Z}_{\boldsymbol{\xi}}^\top \mathbf{W}_{\boldsymbol{\xi}} \mathbf{Z}_{\boldsymbol{\xi}})^{-1}$, which means that $\mathbf{u}_{\boldsymbol{\xi}}$ is the solution of the linear system $\mathbf{Z}_{\boldsymbol{\xi}}^\top \mathbf{W}_{\boldsymbol{\xi}} \mathbf{Z}_{\boldsymbol{\xi}} \mathbf{u}_{\boldsymbol{\xi}} = \mathbf{e}_1$. The prediction of the blackbox outputs at $\boldsymbol{\xi}$ is then

$$\hat{\mathbf{y}}(\boldsymbol{\xi}) = \begin{bmatrix} \hat{f}(\boldsymbol{\xi}) & \hat{c}_1(\boldsymbol{\xi}) & \dots & \hat{c}_m(\boldsymbol{\xi}) \end{bmatrix} = \mathbf{u}_{\boldsymbol{\xi}}^\top \mathbf{Z}_{\boldsymbol{\xi}}^\top \mathbf{W}_{\boldsymbol{\xi}} \mathbf{Y}. \tag{16}$$

The cross-validation value $\hat{\hat{y}}(\mathbf{x}_i)$ (i.e., the value of the LOWESS model at $\mathbf{x}_i$ when the data point $\mathbf{x}_i$ is not used to build the model) are computed by setting w_i to 0. Unfortunately, unlike for RBF (radial basis function) models or PRS (polynomial response surfaces), we do not know any computational shortcut allowing a more efficient computation of the values of $\hat{\hat{y}}$. However, each value $\hat{\hat{y}}(\mathbf{x}_i)$ is computed for the same computational cost as a prediction $\hat{y}(\mathbf{x}_i)$.

A.1 Weights computation in LOWESS models

The weight $w_i(\boldsymbol{\xi})$ quantifies the relative importance of the data point $\mathbf{x}_i$ in the construction of the local regression at $\boldsymbol{\xi}$. Like for kernel smoothing, it relies on a kernel function ϕ and depends on the distance between $\boldsymbol{\xi}$ and $\mathbf{x}_i$. In our method, we use

$$w_i(\boldsymbol{\xi}) = \phi\left(\lambda \frac{\|\boldsymbol{\xi}-\mathbf{x}_i\|_2}{d_{n+1}(\boldsymbol{\xi})}\right), \tag{17}$$

where $\phi(d)$ is one of the kernel functions described in Table 1 and Figure 7. All kernel functions are normalized so that $\phi(0) = 1$ and, if applicable, $\int_{\mathbb{R}} \phi = 1$. As the integral of the inverse multiquadratic kernel does not converge, the normalization constant 52.015 is introduced to minimize the $\mathcal{L}^2$ distance between the inverse multiquadratic and inverse quadratic kernel. $\lambda > 0$ is a parameter that control the general shape of the model, and $d_{n+1}(\boldsymbol{\xi})$ is a local scaling coefficient that estimates the distance of the $n+1^{th}$ closest data point to $\boldsymbol{\xi}$. The kernel function ϕ and the shape parameter λ are chosen to minimize the AOECV (aggregate order error with cross-validation) described in Section A.2. The fact that some of the available kernel function have a compact domain gives to LOWESS models the ability to ignore outliers or aberrant data points. As an example, if the blackbox fails to compute correctly the objective function for a given data point, the value returned by the blackbox might be an arbitrarily high value (e.g. $1.8\ 10^{308}$ for a C++ code returning the standard max double). With non compact kernel function, this would perturb the LOWESS model on the entire design space. However, if there is no such aberrant data points, non-compact kernel functions tend to yield better results.

Table 1: Possible values for the kernel function ϕ.

#	Kernel name	$\phi : \mathbb{R} \to \mathbb{R}^+$	Compact domain
1	Tri-cubic	$\phi(d) = (1 - \lvert\frac{162}{140}d\rvert^3)^3 \mathbb{1}_{\lvert d\rvert \le \frac{140}{162}}$	Yes
2	Epanechnikov	$\phi(d) = (1 - \frac{16}{9}d^2)\mathbb{1}_{\lvert d\rvert \le \frac{3}{4}}$	Yes
3	Bi-quadratic	$\phi(d) = (1 - \lvert\frac{16}{15}d\rvert^2)^2 \mathbb{1}_{\lvert d\rvert \le \frac{15}{16}}$	Yes
4	Gaussian	$\phi(d) = \exp(-\pi d^2)$	No
5	Inverse quadratic	$\phi(d) = \frac{1}{1+\pi^2 d^2}$	No
6	Inverse multiquadratic	$\phi(d) = \frac{1}{\sqrt{1+52.015 d^2}}$	No
7	Exp-root	$\phi(d) = \exp(-2\sqrt{\lvert d\rvert})$	No

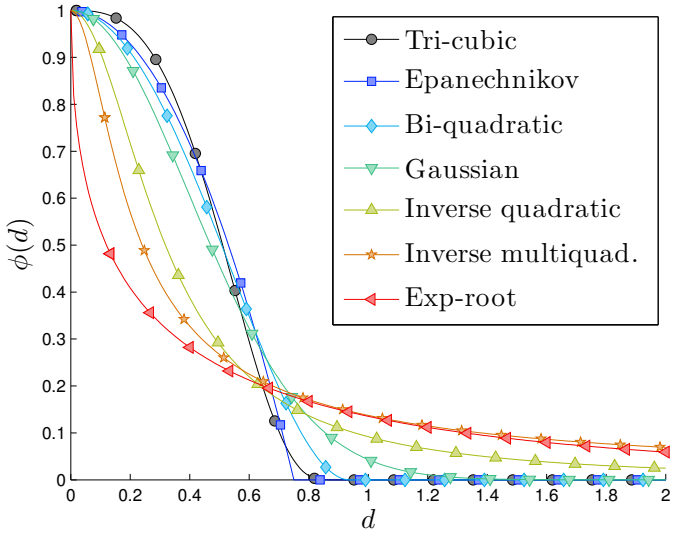

Figure 7: Representation of the 7 kernels listed in Table 1.

To obtain a model $\hat{y}$ differentiable everywhere, [31] defines $d_{n+1}(\boldsymbol{\xi})$ such that the expected number of training points in the ball of centre $\boldsymbol{\xi}$ and radius $d_{n+1}(\boldsymbol{\xi})$ is $n+1$:

$$\mathbb{E}\left[\text{card}\left\{\mathbf{x}_i : \mathbf{x}_i \in \mathbf{X}, \|\boldsymbol{\xi} - \mathbf{x}_i\|_2 \leq d_{n+1}(\boldsymbol{\xi})\right\}\right] = n+1. \tag{18}$$

Moreover, [31] observes that the values $\left\{\|\boldsymbol{\xi} - \mathbf{x}_i\|_2^2\right\}_{i=1,\ldots,p}$ can be well fitted by a Gamma distribution and therefore defines the local scaling parameter as

$$d_{n+1}(\boldsymbol{\xi}) = \sqrt{g^{(-1)}\left(\frac{\mu_{\boldsymbol{\xi}}^2}{\sigma_{\boldsymbol{\xi}}^2}, \frac{\sigma_{\boldsymbol{\xi}}^2}{\mu_{\boldsymbol{\xi}}}; \frac{n+1}{p}\right)}, \tag{19}$$

where $\mu_{\boldsymbol{\xi}}$ (resp. $\sigma_{\boldsymbol{\xi}}^2$) denotes the mean (resp. variance) of $\|\boldsymbol{\xi} - \mathbf{x}_i\|_2^2$ over $\mathbf{X}$ and $g^{(-1)}(k, \theta; .)$ is the inverse function of the cumulative density function of a Gamma distribution with shape parameter k and scale parameter θ.

A.2 Aggregate Order Error with Cross-Validation

The AOECV is an error metric that aims at quantifying the quality of a *multi-output* surrogate model (i.e. a surrogate model that returns predictions for each of the blackbox outputs: objective and constraint functions) in the context of surrogate-based (or surrogate-assisted) optimization. Specifically, it aims at quantifying the equivalence between problems (P) and ($\hat{P}$) for a given surrogate model. We first define the aggregate constraint violation function [15] $h(\mathbf{x}) = \sum_{j=1}^{m} \max\{0, c_j(\mathbf{x})\}^2$. Note that other definitions of h are possible (notably: number of violated constraints, most violated constraint, etc.) but as the previous definition of h is used in the main MADS instance to solve (P), we need to use the same aggregate constraint in our definition of the AOECV.

We then define the order operators

$$\mathbf{x} \prec \mathbf{x}' \Leftrightarrow \begin{cases} h(\mathbf{x}) < h(\mathbf{x}') \\ \text{or} \\ h(\mathbf{x}) = h(\mathbf{x}') \text{ and } f(\mathbf{x}) < f(\mathbf{x}'), \end{cases} \tag{20}$$

$$\mathbf{x} \preceq \mathbf{x}' \Leftrightarrow \texttt{not}(\mathbf{x}' \prec \mathbf{x}) \tag{21}$$

which are transitive. In particular, the incumbent solution $\mathbf{x}^t$ of the main problem (P) is such that $\mathbf{x}^t \preceq \mathbf{x}$, $\forall \mathbf{x} \in \mathbf{X}$. Similarly, a global minimizer $\mathbf{x}^*$ is such that $\mathbf{x}^* \preceq \mathbf{x}$, $\forall \mathbf{x} \in \mathcal{X}$. By the same principle,

we define the operator $\hat{\prec}$ by using the cross-validation values $\hat{f}$ and $\hat{h} = \sum_{j=1}^{m} \max\{0, \hat{c}_j(\mathbf{x})\}^2$ instead of f and h. We then define the aggregated order error with cross-validation (AOECV) metric:

$$\mathcal{E}_{AOECV} = \frac{1}{p^2} \sum_{i=1}^{p} \sum_{j=1}^{p} \texttt{xor}\left(\mathbf{x}_i \prec \mathbf{x}_j, \mathbf{x}_i \hat{\prec} \mathbf{x}_j\right). \tag{22}$$

where `xor` is the exclusive `or` operator (i.e., `xor` $(A, B) = 1$ if the booleans A and B differ and 0 otherwise). The metric allows to quantify how often the model is able to correctly decide which of two points is better.

The shape parameter λ and the kernel function ϕ are then chosen to minimize $\mathcal{E}_{AOECV}(\lambda, \phi)$. If two couples (λ, ϕ) lead to the same metric value (because of the piecewise-constant nature of the metric), the couple with the smallest value of λ (i.e. the smoother model) is preferred.

B Detailed description of test problems

The five engineering design application problems considered for testing the solvers are listed below in Table 2. Problem sizes are given by n and m, where n denotes the number of design variables and m the number of constraints to satisfy. Table 2 also indicates whether variables are integer or unbouded, as well as the best known value of objective functions. For each problem, a short description follows.

Table 2: Summary of the five engineering design optimization problems.

Problem name	n	m	Integer variables	Infinite bounds	Best obj. function
TCSD	3	4	No	No	0.0126652
Vessel	4	4	No	No	5,885.332
Welded	4	6	No	No	2.38096
Solar 1	9	5	Yes	Yes	–900,417
Solar 7	7	6	Yes	Yes	–4,976.17

The TCSD (Tension/Compression Spring Design) problem [3, 9, 16] consists of minimizing the weight of a spring under mechanical constraints. The design variables define the geometry of the spring. The constraints concern shear stress, surge frequency and minimum deflection. The best known solution, denoted $\mathbf{x}^*$ hereafter, and the bounds on the variables, denoted by $\underline{\mathbf{x}}$ and $\bar{\mathbf{x}}$, are given in Table 3.

Table 3: Variables of the TCSD problem.

Variable description	$\underline{\mathbf{x}}$	$\bar{\mathbf{x}}$	$\mathbf{x}^*$
Mean coil diameter	0.05	2	0.051686696913218
Wire diameter	0.25	1.3	0.356660815351066
Number of active coil	2	15	11.292312882259289

The Vessel problem [16, 21] consists of designing a compressed air storage tank. The design variables define the geometry of the tank The constraints are related to the volume, pressure and solidity of the tank. The objective consists of minimizing the total cost of the tank, including material and labour. See Table 4 for details about the best known solutions and the lower and upper bounds on the variables.

The Welded (or welded beam design) problem (Version I) [16, 28] consists of minimizing the construction cost of a beam, under shear stress, bending stress, load and deflection constraints. The design variables define the geometry the beam and the characteristics of the welded joint. For the best known solution and the bounds on the variables, see Table 5.

Table 4: Variables of the Vessel problem.

Variable description	$\underline{\mathbf{x}}$	$\bar{\mathbf{x}}$	$\mathbf{x}^*$
Thickness of the vessel	0.0625	6.1875	0.778168641330718
Thickness of the head	0.0625	6.1875	0.384649162605973
Inner radius	10	200	40.319618721803231
Length of the vessel without heads	10	200	199.999999998822659

Table 5: Variables of the Welded problem.

Variable description	$\underline{\mathbf{x}}$	$\bar{\mathbf{x}}$	$\mathbf{x}^*$
Thickness of the weld	0.1	2	0.244368407428265
Length of the welded joint	0.1	10	6.217496713101864
Width of the beam	0.1	10	8.291517255567012
Thickness of the beam	0.1	2	0.244368666449562

The Solar1 and Solar7 [17] problems concern the optimization of a solar farm, including the heliostat field and/or the receiver. The Solar1 optimization problem consists of maximizing the energy received over a period of 24 hours under several constraints of budget and heliostat field area [17]. This problem has one integer variable that has no upper bound. The design variables for the Solar1 problem are described in Table 6.

Table 6: Variables of the Solar1 problem.

Variable description	$\underline{\mathbf{x}}$	$\bar{\mathbf{x}}$	$\mathbf{x}^*$
Heliostat height	1	40	6.165258994385601
Heliostat width	1	40	10.571794049143792
Tower height	20	250	91.948461670428486
Receiver aperture height	1	30	6.056202026704944
Receiver aperture width	1	30	11.674984434929991
Max number of heliostats (Integer)	1	$+\infty$	1507
Field maximum angular span	1	89	51.762281627953051
Minimum distance to tower	0.5	20	1.347318830713629
Maximum distance to tower	1	20	14.876940809562798

The Solar7 problem consists of maximizing the efficiency of the receiver over a period of 24 hours, for a given heliostats field, under 6 binary constraints [17]. This problem has one integer variable that has no upper bound. The objective function is the energy transferred to the molten salt. The design variables for the Solar7 problem are described in Table 7.

Table 7: Variables of the Solar7 problem.

Variable description	$\underline{\mathbf{x}}$	$\bar{\mathbf{x}}$	$\mathbf{x}^*$
Aperture height	1	30	11.543687848308958
Aperture width	1	30	15.244236061098078
Outlet temperature	793	995	803.000346734710888
Number of tubes (Integer)	1	$+\infty$	1292
Insulation thickness	0.01	5	3.399190219909724
Tubes inside diameter	0.005	0.1	0.010657067457678
Tubes outside diameter	0.0055	0.1	0.011167646941518

References

[1] E. Alba, G. Luque, and S. Nesmachnow. Parallel metaheuristics: recent advances and new trends. International Transactions in Operational Research, 20(1):1–48, 2013.

[2] R. Alizadeh, J.K. Allen, and F. Mistree. Managing computational complexity using surrogate models: a critical review. Research in Engineering Design, 2020.

[3] J. Arora. Introduction to Optimum Design. Elsevier Science, 2004.

[4] C.G. Atkeson, A.W. Moore, and S. Schaal. Locally weighted learning. Artificial Intelligence Review, pages 11–73, 1997.

[5] C. Audet and J.E. Dennis, Jr. Mesh adaptive direct search algorithms for constrained optimization. SIAM Journal on Optimization, 17(1):188–217, 2006.

[6] C. Audet and J.E. Dennis, Jr. A progressive barrier for derivative-free nonlinear programming. SIAM Journal on Optimization, 20(1):445–472, 2009.

[7] C. Audet, M. Kokkolaras, S. Le Digabel, and B. Talgorn. Order-based error for managing ensembles of surrogates in derivative-free optimization. Journal of Global Optimization, 70(3):645–675, 2018.

[8] C. Audet, S. Le Digabel, C. Tribes, and V. Rochon Montplaisir. The NOMAD project. Software available at https://www.gerad.ca/nomad.

[9] A.D. Belegundu. A Study of Mathematical Programming Methods for Structural Optimization. University of Iowa, 1982.

[10] W.S. Cleveland. Robust locally weighted regression and smoothing scatterplots. Journal of the American Statistical Association, 74:829–836, 1979.

[11] W.S. Cleveland. LOWESS: A Program for Smoothing Scatterplots by Robust Locally Weighted Regression. The American Statistician, 35(1), 1981.

[12] W.S. Cleveland and S.J. Devlin. Locally weighted regression: An approach to regression analysis by local fitting. Journal of the American Statistical Association, 83:596–610, 1988.

[13] W.S. Cleveland, S.J. Devlin, and E. Grosse. Regression by local fitting: methods, properties, and computational algorithms. Journal of Econometrics, 37(1):87–114, 1988.

[14] E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. Mathematical Programming, 91(2):201–213, 2002.

[15] R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. Mathematical Programming, Series A, 91:239–269, 2002.

[16] H. Garg. Solving structural engineering design optimization problems using an artificial bee colony algorithm. Journal of Industrial and Management Optimization, 10(3):777–794, 2014.

[17] Mathieu Lemyre Garneau. Modelling of a solar thermal power plant for benchmarking blackbox optimization solvers. Master's thesis, École Polytechnique de Montréal, 2015.

[18] Raphael T. Haftka, Diane Villanueva, and Anirban Chaudhuri. Parallel surrogate-assisted global optimization with expensive functions – a survey. Structural and Multidisciplinary Optimization, 54(1):3–13, Jul 2016.

[19] P. Hansen and N. Mladenović. Variable neighborhood search: principles and applications. European Journal of Operational Research, 130(3):449–467, 2001.

[20] Prasad Jogalekar and Murray Woodside. Evaluating the scalability of distributed systems. IEEE Trans. Parallel Distrib. Syst., 11(6):589–603, June 2000.

[21] B. K. Kannan and S. N. Kramer. Augmented Lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design. Journal of Mechanical Design, 65:103–112+, 1993.

[22] O. Kramer, D.E. Ciaurri, and S. Koziel. Derivative-free optimization. In S. Koziel and XS. Yang, editors, Computational Optimization, Methods and Algorithms, volume 356 of Studies in Computational Intelligence, pages 61–83. Springer, 2011.

[23] S. Le Digabel. Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm. ACM Transactions on Mathematical Software, 37(4):44:1–44:15, 2011.

[24] S. Le Digabel, M.A. Abramson, C. Audet, and J.E. Dennis, Jr. Parallel versions of the MADS algorithm for black-box optimization. In Optimization days, Montreal, May 2010. GERAD. Slides available at http://www.gerad.ca/Sebastien.Le.Digabel/talks/2010_JOPT_25mins.pdf.

[25] M.D. McKay, R.J. Beckman, and W.J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics, 21(2):239–245, 1979.

[26] N. Mladenović and P. Hansen. Variable neighborhood search. Computers and Operations Research, 24(11):1097–1100, 1997.

[27] Mahdi Pourbagian, Bastien Talgorn, WagdiG. Habashi, Michael Kokkolaras, and Sébastien Le Digabel. Constrained problem formulations for power optimization of aircraft electro-thermal anti-icing systems. Optimization and Engineering, pages 1–31, 2015.

[28] Singiresu S. Rao. Engineering Optimization: Theory and Practice, 3rd Edition. Wiley-Interscience, 1996.

[29] T.J. Santner, B.J. Williams, and W.I. Notz. The Design and Analysis of Computer Experiments, chapter 5.2.2, Designs Generated by Latin Hypercube Sampling, pages 127–132. Springer, New York, NY, 2003.

[30] B. Talgorn. SGTELIB: Surrogate model library for derivative-free optimization. https://github.com/bbopt/sgtelib, 2019.

[31] B. Talgorn, C. Audet, M. Kokkolaras, and S. Le Digabel. Locally weighted regression models for surrogate-assisted design optimization. Optimization and Engineering, 19(1):213–238, 2018.

[32] B. Talgorn, S. Le Digabel, and M. Kokkolaras. Statistical Surrogate Formulations for Simulation-Based Design Optimization. Journal of Mechanical Design, 137(2):021405–1–021405–18, 2015.