Algorithm 1027: NOMAD Version 4: Nonlinear Optimization with the MADS Algorithm

CHARLES AUDET, SÉBASTIEN LE DIGABEL, VIVIANE ROCHON MONTPLAISIR, and CHRISTOPHE TRIBES, GERAD and Polytechnique Montréal, Département de Mathématiques et Génie Industriel

NOMAD is a state-of-the-art software package for optimizing blackbox problems. In continuous development since 2001, it constantly evolved with the integration of new algorithmic features published in scientific publications. These features are motivated by real applications encountered by industrial partners. The latest major release of NOMAD, version 3, dates to 2008. Minor releases are produced as new features are incorporated. The present work describes NOMAD 4, a complete redesign of the previous version, with a new architecture providing more flexible code, added functionalities, and reusable code. We introduce algorithmic components, which are building blocks for more complex algorithms and can initiate other components, launch nested algorithms, or perform specialized tasks. They facilitate the implementation of new ideas, including the MegaSearchPoll component, warm and hot restarts, and a revised version of the PsdMads algorithm. Another main improvement of NOMAD 4 is the usage of parallelism, to simultaneously compute multiple blackbox evaluations and to maximize usage of available cores. Running different algorithms, tuning their parameters, and comparing their performance for optimization are simpler than before, while overall optimization performance is maintained between versions 3 and 4. NOMAD is freely available at www.gerad.ca/nomad and the whole project is visible at github.com/bbopt/nomad.

CCS Concepts: • Software and its engineering; • Mathematics of computing → Solvers; Mathematical optimization; • Computing methodologies → Parallel algorithms;

Additional Key Words and Phrases: Optimization software, blackbox optimization, derivative-free optimization, mesh adaptive direct search

ACM Reference format:
Charles Audet, Sébastien Le Digabel, Viviane Rochon Montplaisir, and Christophe Tribes. 2022. Algorithm 1027: NOMAD Version 4: Nonlinear Optimization with the MADS Algorithm. ACM Trans. Math. Softw. 48, 3, Article 35 (September 2022), 22 pages.
https://doi.org/10.1145/3544489

All authors contributed equally to this research.
This work is supported by the NSERC CRD RDCPJ 490744-15 grant coupled with an InnovÉÉ grant, both in collaboration with Hydro-Québec and Rio Tinto, and by the NSERC Alliance grant 544900-19 in collaboration with Huawei-Canada.
Authors’ address: C. Audet, S. Le Digabel, V. Rochon Montplaisir, and C. Tribes, GERAD and Polytechnique Montréal, Département de Mathématiques et Génie Industriel, C.P. 6079, Succ. Centre-ville, Montréal, Québec, Canada, H3C 3A7; emails:  [Charles.Audet, Sebastien.Le.Digabel, Viviane.Rochon.Montplaisir, Christophe.Tribes]@gerad.ca.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2022 Association for Computing Machinery.
0098-3500/2022/09-ART35 $15.00
https://doi.org/10.1145/3544489
1 INTRODUCTION

NOMAD is a software package designed for the class of blackbox optimization (BBO) problems [17]. The term “blackbox” indicates that there is no information except the input and output. There is no analytic description of the objective and/or constraint functions; there are no available derivatives, possibly because the functions are not differentiable; and the functions may occasionally fail to return valid output and may require significant computational time to evaluate. This makes BBO problems difficult to solve, in the sense that many optimization algorithms and heuristics cannot be applied. A typical BBO problem is a computer simulation of an engineering or physical problem.

The development of NOMAD was initiated in 2001 to implement direct search algorithms, and major version 3 was released in 2008 [46]. The mesh adaptive direct search (Mads) algorithm [12] is at the core of NOMAD 3; it provides a flexible framework and is supported by a rigorous hierarchical convergence analysis based on various degrees of smoothness of the functions defining the problem. Since its original release, minor releases of NOMAD 3 have included several improvements and additions of algorithms to solve a variety of blackbox optimization problems efficiently.

NOMAD has proven its usefulness in scientific papers as well as in established companies. Our own work includes contributions in hydrology [2, 54, 60], pharmacology [68], metamaterial design [15], alloy design [35, 37, 38], chemical engineering [8, 41], and bioinformatics [34]. Many other researchers use NOMAD in a variety of fields. In astrophysics, for example, NOMAD is used for black hole observation [51], for tracking the interstellar object 1I/’Oumuamua [42], for kinematics analysis of galaxies [57], and for gravitational wave detection, in an article with more than 700 co-authors [1]. Hundreds of applications are reported in the surveys [5, 7, 36], including many on energy, engineering design, and materials science.

In retrospect, the main development avenues of the NOMAD software and the MADS algorithm may be classified into three categories:

- **Algorithmic improvements.** The MADS algorithm was modified to reduce the number of evaluations through constraint handling techniques [13, 22], by dynamically scaling and exploiting the specificity of variables [21, 23, 24] and by the improved integration of surrogates [10, 20, 61–63].
- **Sub- and super-algorithms.** MADS may call other optimization algorithms during its execution. Sub-algorithms, when used under adequate conditions, may produce good candidate points for evaluation. Using the right candidates has a strong influence on the performance of the software. Useful algorithms and techniques are proposed in [6, 9, 18, 27]. Sub-algorithms may also be tailored to exploit surrogate functions [10, 20, 62]. Conversely, the MADS algorithm can be used as part of a broader direct search super-algorithm. For example, BiMADS and MultiMADS [25, 26] solve multiobjective optimization problems by running several MADS instances while managing the progress in obtaining a detailed Pareto front. ROBUSTMADS [19] interrupts sequences of MADS runs by redefining the objective function to take into account noisy values. PsdMADS [14] divides a large problem into problems of smaller dimension and launches instances of MADS in a parallel environment.
- **Performance and parallelism.** A major effort was placed into reducing the wall clock time to obtain good solutions. The opportunistic strategy for evaluating points, combined with ordering points to promote the most promising, ensures faster convergence [59]. Quadratic models approximate the problem to rapidly find better points [31]. Methods were developed to span a limited number of directions while maintaining the convergence proof, again to limit the number of blackbox evaluations [19]. Conversely, to maximize core utilization during optimization, subspace exploration strategies [4, 14] as well as parallel strategies in
NOMAD [46] were developed. When NOMAD 3 was released, these strategies were well adapted with the computing resources available but could not be scaled up to fully utilize the current abundance of computing resources.

Over the years, it became increasingly difficult to maintain and enhance the functionalities of NOMAD 3. Recent algorithmic developments required modifications in many portions of the software. The complex interactions between algorithms and sub-algorithms were not sufficiently anticipated. It was therefore decided to completely redesign the software.

The main goal of this new version of NOMAD remains to solve efficiently a variety of constrained and unconstrained BBO problems. In NOMAD 4, the Mads algorithm as well as other algorithms deemed useful in NOMAD 3 have been re-implemented using primitive algorithmic components, which are building blocks for more complex algorithms, and interfaces adapted from the experience gained during the development of NOMAD 3. This approach promotes software maintainability, as components may be reused when adding new algorithms. This is an important requisite of this new version of the software even though it forces the rewriting of most of the source code. The requisite to efficiently use a large number of available cores for parallel blackbox function evaluations also had a strong impact on the architecture when redesigning the software. Finally, the ability to tune algorithmic parameters (which control the algorithmic components) and to compare different algorithms is also an important requisite of the development. The optimization performance must be maintained between the versions.

This article describes the design of NOMAD 4 to achieve this goal with the updated requisites. Sections 2 and 3 present the Mads algorithm and other algorithmic components re-implemented from NOMAD 3. The strategies for parallel blackbox evaluations in NOMAD 4 are presented in Section 4. The software architecture and development are presented in Section 5. New algorithmic developments in NOMAD must be assessed in terms of optimization performance and compared with other blackbox optimizers on a large variety of problems: Section 6 compares the performance of the NOMAD 3 and 4 versions and illustrates the gains produced by the use of multiple cores in NOMAD 4. Finally, Section 7 discusses future developments.

## 2 THE MADS ALGORITHM

NOMAD solves optimization problems of the form

$$\min_x \{ f(x) : x \in \Omega \},$$

where $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is the objective function and $\Omega \subseteq \mathbb{R}^n$ is the feasible region. Allowing the objective function to take the value $\infty$ is useful to exclude trial points for which the evaluation failed to return valid output, for example, when the blackbox crashes or returns an error message. The original MADS article [12] handled the constraint set by minimizing the unconstrained extreme barrier function $f_\Omega : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ defined as

$$f_\Omega(x) := \begin{cases} f(x) & \text{if } x \in \Omega, \\ \infty & \text{if } x \notin \Omega. \end{cases}$$

Later, the progressive barrier [13] approach was proposed to exploit the amount by which constraints are violated. The optimization problem is redefined as

$$\min_{x \in \mathcal{X} \subseteq \mathbb{R}^n} \{ f(x) : c(x) \leq 0 \},$$

where $f : \mathcal{X} \subseteq \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ and $c : \mathcal{X} \subseteq \mathbb{R}^n \to (\mathbb{R} \cup \{\infty\})^m$ are functions with $c = (c_1, c_2, \ldots, c_m)$, and $\mathcal{X}$ is some subset of $\mathbb{R}^n$. Again, the entire feasible region is denoted by $\Omega = \{ x \in \mathcal{X} : c(x) \leq 0 \}$. The set $\mathcal{X}$ is frequently taken as being $\mathbb{R}^n$, the space of continuous variables, or as the set of...
nonnegative variables $\mathbb{R}^n_+$. The progressive barrier approach uses the constraint violation function $h : \mathbb{R}^n_+ \mapsto \mathbb{R}_+ \cup \{\infty\}$, which is defined as

$$h(x) := \begin{cases} \sum_{j=1}^{m} \left( \max(c_j(x), 0) \right)^2 & \text{if } x \in X, \\ \infty & \text{otherwise.} \end{cases}$$

The fundamental properties of the constraint violation function are that it is nonnegative, and it equals zero if and only if $x$ belongs to $\Omega$. The approach allows infeasible points whose value of $h$ is below some threshold $h_{\text{max}}^k$ that progressively decreases toward zero as the iteration number $k$ increases.

Each iteration of the MADS algorithm explores the space of variables through a global exploration called search step and a local exploration called poll step. Both these steps generate trial points, which are candidates for evaluation, on a discretization of $X$ called the mesh. At iteration $k$, there is always at least one current best-known incumbent solution: $x^{\text{feas}}$, the feasible solution with the lowest value of $f$, and $x^{\text{inf}}$, the infeasible solution with the lowest value of $f$ among those whose values of $h$ are below the threshold $h_{\text{max}}^k$. The mesh is defined as $M^k := \{x + \delta^k y : x \in \{x^{\text{feas}}, x^{\text{inf}}\}, y \in \mathbb{N}^p \} \subset \mathbb{R}^n$, where $\delta^k \in \mathbb{R}$ is the mesh size parameter and $D \in \mathbb{R}^{n \times p}$ is a positive spanning set of $p$ directions that satisfies specific requirements. The simplest possible set $D$ is the union of all positive and negative coordinate directions and is the one implemented in NOMAD, hence the following redefinition of the mesh at iteration $k$:

$$M^k := \{x + \delta^k y : x \in \{x^{\text{feas}}, x^{\text{inf}}\}, y \in \mathbb{Z}^n \} \subset \mathbb{R}^n.$$

The search step is flexible and allows the user to explore any finite number of mesh points in the set named $\delta^k$. NOMAD 4 proposes a one-point rudimentary line search in the direction of the previous success [12], a Nelder-Mead-inspired search step [27], and a search based on the minimization of a quadratic model [6, 31]. Additional search strategies include basic Latin hypercube sampling [50], Variable Neighborhood Search [9, 55] to escape locally optimal solutions, and others based on advanced statistical surrogates [20, 62, 63]. The Latin hypercube sampling can be also used as an initial search strategy (iteration 0, with its own evaluation budget). The user may also integrate their own search strategy.

The poll step follows more rigid rules than the search step. Poll points are confined to so-called frames around the incumbents $x^{\text{feas}}$ and $x^{\text{inf}}$. The dimension of a frame is set by a frame size parameter $\Lambda^k$, which is always greater than or equal to the mesh size parameter $\delta^k$. The key element of the poll step is that the poll set $P^k$ must lie within the frames around the incumbents, and for each incumbent solution and corresponding frame, the set of directions from the incumbent to the poll points must form a positive spanning set for $\mathbb{R}^n$. For MADS, as $k$ goes to infinity, the union of these normalized directions becomes dense in the unit sphere. NOMAD 3 includes many examples of poll steps, including coordinate search [33], generalized pattern search [66], and OrthoMADS with $2n$ [3] and $n + 1$ [18] directions. Of these, NOMAD 4 currently implements coordinate search and OrthoMADS with $2n$ and $n + 1$ directions; other simple direction strategies are also included.

Algorithm 2.1 shows MADS with the progressive barrier to handle constraints. The progressive barrier threshold update rules governing the search and poll steps involve the following definitions [17]. The feasible point $x \in \Omega$ is said to dominate $y \in \Omega$, denoted $x \prec_f y$, when $f(x) \prec f(y)$. The infeasible point $x \in X \setminus \Omega$ is said to dominate $y \in X \setminus \Omega$, denoted $x \prec_b y$, when $f(x) \leq f(y)$ and $h(x) \leq h(y)$ with at least one strict inequality. The set $V$ used in the poll step is the cache, i.e., the set of all points visited since the algorithm was started.
Algorithm 2.1: Mesh adaptive direct search with the progressive barrier (MADS)

Given \( f : \mathbb{R}^n \mapsto \mathbb{R} \cup \{ \infty \} \), \( h : \mathbb{R}^n \mapsto \mathbb{R} \cup \{ \infty \} \) and starting point \( x^0 \in \Omega \).

0. Initialization

\[
\begin{align*}
\Delta^0 & \in (0, \infty) \quad \text{initial frame size parameter} \\
\tau & \in (0, 1), \text{ with } \tau \text{ rational} \quad \text{mesh-size adjustment parameter} \\
h^0_{\max} & = \infty \quad \text{barrier threshold parameter} \\
\varepsilon_{\text{stop}} & \in [0, \infty) \quad \text{stopping tolerance} \\
k & \leftarrow 0 \quad \text{iteration counter}
\end{align*}
\]

1. Parameter Update

set the mesh size parameter to \( \delta^k = \min\{\Delta^k, (\Delta^k)^2\} \)

let \( x^{\text{feas}} \) and \( x^{\text{inf}} \) be the two incumbent solutions

2. Search

if \( t < f(x^{\text{feas}}) \) or \( t < h(x^{\text{inf}}) \) for some \( t \) in a finite subset \( S^k \) of the mesh \( M^k \)

set \( \Delta^{k+1} \leftarrow \tau^{-1} \Delta^k \), \( h_{\text{max}}^{k+1} \leftarrow h(x^{\text{inf}}) \) and go to 4

otherwise go to 3

3. Poll

let \( P^k \) be a poll set constructed around \( x^{\text{feas}} \) and \( x^{\text{inf}} \) using positive spanning sets

if \( t < f(x^{\text{feas}}) \) or \( t < h(x^{\text{inf}}) \) for some \( t \in P^k \)

set \( \Delta^{k+1} \leftarrow \tau^{-1} \Delta^k \) and \( h_{\text{max}}^{k+1} \leftarrow h(x^{\text{inf}}) \)

else, if \( 0 < h(t) < h(x^{\text{inf}}) \) for some previously evaluated point \( t \)

set \( \Delta^{k+1} = \Delta^k \) and \( h_{\text{max}}^{k+1} \leftarrow \max\{h(v) : h(v) < h(x^{\text{inf}}), v \in V\} \)

otherwise

set \( \Delta^{k+1} \leftarrow \tau \Delta^k \) and \( h_{\text{max}}^{k+1} \leftarrow h(x^{\text{inf}}) \)

4. Termination

if \( \Delta^k \geq \varepsilon_{\text{stop}} \)

increment \( k \leftarrow k + 1 \) and go to 1

otherwise stop and return the incumbents \( x^{\text{feas}} \) and \( x^{\text{inf}} \)

A high-level description of the progressive barrier is found in Chapter 12 of [17] and the detailed presentation appears in [13].

3 ALGORITHMIC COMPONENTS OF MADS

This section describes how the main elements of the MADS algorithm are encoded in NOMAD 4. In NOMAD 3, the design was mostly function oriented. The pseudo-code concepts of Algorithm 2.1 were implemented as functions and attributes. As the software evolved with new algorithms and features being implemented, abstract classes remained seldom used.

NOMAD 4 was designed with an effort on abstraction. The Initialization, Iteration (consisting of Update, Search, and Poll), and Termination concepts presented in Algorithm 2.1 are all defined as algorithmic components in the code with attributes and functions. These key MADS components are present in many optimization algorithms. Hence, they are defined as generic abstract classes in NOMAD 4 (see Section 5). Each algorithm must derive from them its own Initialization, Iteration, etc., or simply inherit the generic one. The tasks performed by components and the interactions between components define how an algorithm unfolds programmatically. An optimization program designed upon generic algorithmic components and tasks ensures easier maintenance and extension, which is a requisite for the revised version of NOMAD. The components can
be readily reused in different optimization programs and can be upgraded when new features are introduced. This flexibility was absent from the design of NOMAD 3.

More details about the concept of algorithmic components and the tasks that they perform in the code are provided in the following subsections. We focus primarily on the MADS algorithm and also present other encapsulated algorithms.

The MADS algorithm takes as input an initial point $x^0$ that is not required to be within the feasible region, a set of algorithmic parameters $P$, an objective function $f$, and some constraint functions $c$. These functions are provided through an executable code considered as a blackbox. MADS provides the best incumbent solution $x^*$ found for this problem.

### 3.1 Terminology

When running an optimization program from given inputs, the code instructions are executed following a specific logic. The present section defines the terminology to describe the execution of MADS.

A task is defined as a group of code instructions adapted for human understanding. In our context, tasks can be generic or specialized, and they can be broken down into smaller tasks. Grouping tasks into algorithmic components allows to visualize the structure of the optimization program and its unfolding. The name and purpose of a component come from the algorithm and the tasks that it performs.

The program consists of a tree of components, which are executed one at a time, by traversing the tree depth-first. The components are named after the algorithms they implement, for example, LH for the Latin hypercube sampling search step, or Mads for the MADS algorithm. The execution of a component unfolds by performing the generic tasks named Start, Run, and End, in that order. These tasks are defined in the algorithmic component abstract class and are available to all inherited classes (see Section 5). The generic tasks (Start, Run, and End) are used to dispatch what a component can do (no other tasks are directly available) and when to interact with another component. In the figures below the generic tasks presented as white boxes are within the algorithmic components.

Specialized tasks are tasks other than the generic tasks, with their name describing their purpose. The Start task may initiate another component or perform a list of specialized tasks (from top to bottom in the figures below). The End task may call specialized tasks required for the closure of the component. Start and End tasks may also be void. The Run task may initiate another component or combine specialized tasks with iterations or testing conditions. Nested components and tasks model the execution of the entire program.

### 3.2 Execution Model for Algorithms

The execution model is illustrated in Figure 1 with the Latin hypercube sampling algorithm, and in Figure 2 with the more complex MADS algorithm, which requires connecting several nested components and specialized tasks. The LH component presented in Figure 1 sequentially performs four groups of specialized tasks: Generate trial points (called by Start), Insert trials point in evaluation queue (called by Run), Evaluate trial points in queue (called by Run), and Display results (called by End). The LH component can be called during the MADS search step (see Section 2) or as a standalone optimization program. In both cases, all trial points are generated during the Start task of the component.

The LH trial point generation details are found in [64]. Some tasks specialized for the generation and evaluation of trial points are presented in detail as they are common to all blackbox optimization programs. Some specialized tasks are optional or are only necessary in some cases; they are written within square brackets. For example, trial points are required to be located on the mesh.
Algorithm 1027: NOMAD Version 4: Nonlinear Optimization with the MADS Algorithm

Fig. 1. The LH component (Latin hypercube sampling algorithm) with its connected tasks. Tasks within brackets are optional.

Fig. 2. The Mads component with its nested tasks and algorithmic components.

when the LH component is called during the MADS search step. In this case, trial points must be projected on the mesh prior to the actual evaluation. However, when the LH component is called as a standalone algorithm, there is no mesh and therefore no projection is required.

When enabled, the cache contains the set of previously evaluated points, and the incumbent point is the best solution found yet. Each trial point is looked up in the cache. If it is found, it does not need to be re-evaluated. Otherwise, it is added to the evaluation queue of points to be evaluated by the blackbox. Evaluations are run, possibly in parallel when multiple cores are available. If a trial point evaluation is better than the incumbent point evaluation, it is possible to skip the points remaining in the queue and to save the cost of evaluating them: this strategy is called opportunism. There is a direct correspondence between some statements of Algorithm 2.1 and the tasks/sub-tasks. The comparison between $f_0(t)$ and $f_0(x^k)$ seen in the search and poll steps corresponds to the tasks Call blackbox on trial point, Retrieve evaluation, and Check for success.

The Start task of the Mads component executes the Initialization component referring to Step 0 in Algorithm 2.1. As illustrated in the left part of Figure 2, the Run task of the
Initialization component first performs the mesh initialization and then conducts the provided initial point evaluation. The task Evaluate $x^0$ in Figure 2 is identical to the previously described tasks Insert trials point in evaluation queue and Evaluate trial points in queue, with the trial point set to the initial point $x^0$. The Run task of the Initialization component concludes with the initialization of the progressive barrier parameters to handle the constraints. The Run task of the Mads component then repeatedly executes Iteration components until a termination criterion is met. Each Iteration component involves respectively an Update, multiple Search, and a Poll component. Each Search component can initiate a sequence of nested algorithmic components. The MADS algorithm offers the flexibility to use any type of algorithm during the search step, as long as a finite number of points is generated, they are projected on the current mesh, and the evaluation budget from the set of algorithmic parameters $P$ is not exceeded.

A component can call another instance of itself, directly or indirectly, as long as it is ensured that no infinite recursion is induced. The nested components of the Search component depend on which search step is performed. An example of search step based on quadratic models is described in Section 3.3 and Figure 3. The Poll and Search components perform the tasks Generate trial points (using different strategies), Insert trial points in evaluation queue, Evaluate trial points in queue, and Update barrier. While it is deployed, the program alternates generation and evaluation of trial points, which can be a limiting factor to the number of parallel evaluations. Section 4 presents a different way to deploy the execution of MADS to exploit parallel blackbox evaluations.

### 3.3 Combining Algorithmic Components

Algorithmic components that represent algorithms can be run standalone. For instance, using NOMAD, it is possible to run the NelderMead algorithm to optimize a problem, and the solution may be compared to the solution found using other algorithms such as MADS. Algorithmic components are building blocks that may be combined and connected together to produce new algorithms. The LH component presented in Figure 1 can be used as a sub-algorithm by the Search component of MADS to generate trial points, as long as the mesh projection is performed. For the same purpose, we developed a QuadraticModelSearch [31] component, in which previous blackbox evaluations are used to construct a quadratic model. This model is used as a surrogate
problem and is optimized to provide new trial points; this optimization is performed by a new instance of Mads, with the quadratic model search step disabled to avoid infinite recursion. Therefore, we have a Mads component that, through a QuadraticModelSearch component and the task Quadratic model optimization, starts the execution of another instance of the Mads component. This is illustrated in Figure 3, with a dashed line connecting the two items.

Various components can be used during a search with some control over the evaluation budget or the number of iterations. A SpeculativeSearch [12] component generates trial points by using the direction of last success, which is speculated as a possible direction of improvement, starting from the current incumbent solution. We also developed a NelderMead component [27], based on the Nelder-Mead algorithm, to iteratively generate and evaluate trial points. A VariableNeighborhoodSearch component [9, 40, 55] to attempt escaping local solution is also available.

The Mads component, with all its nested components and tasks, can itself be used within a super-algorithm that does not necessarily rely on a mesh: see, for example, [48] where NOMAD is hybridized with a mesh-free line search method. It may also be executed repeatedly to solve a biobjective optimization problem through a series of single-objective formulations [25].

Another example of the Mads component being used as part of a super-algorithm is in ParallelSpaceDecomposition (PsdMads [14]), where large problems are solved using an asynchronous parallel algorithm in which the parallel processes are launched on subproblems over subsets of variables. A version of PsdMads is implemented in NOMAD 4 using algorithmic components. It is described in Section 4.5.

4 PARALLEL BLACKBOX EVALUATIONS
A typical user of NOMAD with access to a specific computational capacity would like to obtain the best possible solution for an optimization problem within a certain time limit. This implies that NOMAD must efficiently exploit all available cores. In some cases, the blackbox evaluation itself runs in parallel, using all cores, but that is not always the case. An assumption for the software development is that running a blackbox evaluation requires significantly more computational time compared to the other algorithmic tasks. Therefore, NOMAD must efficiently distribute the blackbox evaluations in parallel, on secondary threads, while all other tasks are executed on a single main thread. Several strategies for such parallel blackbox evaluations are presented in this section.

4.1 The Evaluation Queue
Regardless of the strategy used to manage the parallel evaluations, an evaluation queue is maintained to manage evaluations, in which the elements are trial points to be evaluated. It behaves as a priority queue and is implemented as a sorted vector. When a trial point is generated, it is inserted in the queue, provided that it was not previously evaluated. The trial points may be ordered so that the most promising ones are evaluated first. Sorting the points is important when opportunism is used, because when the evaluation of a point leads to a new success, the remaining points in the queue are not evaluated. Different ordering strategies are available to sort the trial points in the queue: based on the evaluations of a quadratic model, on the direction of the last success, on the order in which trial points were generated, on the lexicographic order of their coordinates, or simply mixed randomly.

4.2 Grouping Evaluations
NOMAD provides the option to group trial points, with a given maximum group size, for evaluation. With this strategy, users are in charge of managing the dispatching of the groups of points, depending on the specifics of their blackbox and computers, in order to maximize core usage. It is
preferable that groups be filled at maximal capacity so that all available cores are exploited. Exploratory work on filling the groups when there is an insufficient number of trial points in the queue appears in [45].

4.3 Parallel Evaluations on Multiple Threads

NOMAD manages one or more main threads and, if additional cores are available, optional secondary threads. In this subsection, we consider a single main thread. See Section 4.5 for a case where multiple main threads are used. The main thread performs all algorithmic tasks, including some of the evaluations of trial points. The secondary threads only execute evaluation tasks. These tasks, independently of the algorithm, are grouped under the name “Evaluate one trial point in queue” and are the same as those listed in “Evaluate trial points in queue” in Figure 1. Each thread performs one evaluation task at a time.

Figure 4 illustrates the tasks’ workflow performed in the main thread and two secondary threads. The specialized tasks for an algorithm are not presented. In the main thread, the iterative process of dequeuing and evaluating points terminates when there are no more points in the queue or when some criterion is met (budget of evaluations is filled, opportunism condition is met, etc.). At this stage, the evaluation queue is cleared from any remaining unevaluated trial points, the progressive barrier parameters are updated, new trial points are generated, and the algorithm continues. In the meantime, the secondary threads keep on working and the iterative process of dequeuing and evaluating points goes on until the algorithm terminates in the main thread. All information relative to a trial point (success, objective, and constraint values) that is evaluated in a secondary thread is made available to the other tasks of the algorithm, and this information is taken into account in the main thread for the continuation of the algorithm.

The management of threads is currently done using the OpenMP API [32]. The user may provide the number of threads $n_t$ to efficiently access the computer cores; otherwise, an OpenMP routine computes the number of available threads.

Using OpenMP, NOMAD can handle parallel evaluations of trial points. Nevertheless, to allow distribution of evaluations using another strategy, we have the option to group the points contained in the evaluation queue into blocks of points of selected size. These points can be evaluated by any means the user finds appropriate.
Algorithm 1027: NOMAD Version 4: Nonlinear Optimization with the MADS Algorithm

Fig. 5. The MegaSearchPoll version of MADS with its nested tasks and algorithmic components. A MegaSearchPoll component is added to the Mads component of Figure 2. The MegaSearchPoll component and its dependant tasks and components are presented in a dashed box. The Run and End tasks of the Search and Poll are not called.

4.4 Grouping Trial Points Generation
In every Search and Poll component of the generic MADS algorithm described in Section 2, evaluations are performed immediately after the trial points are generated. This approach generates few points, sometimes a single point, to be evaluated, which makes it difficult to exploit multiple cores for evaluations and to use parallelism at its full capacity. We developed a new combination of Search and Poll components, called MegaSearchPoll (see Figure 5). It generates all the trial points for the Search and Poll components, and only then the points are inserted in the queue and evaluated in parallel. This way, more points are evaluated at a given time. Additionally, search strategies were reworked to provide more promising points to evaluate.

The Search components of the MegaSearchPoll cannot consist of iterative algorithms that alternate generation and evaluation of trial points. When possible, a single iteration of trial point generation is performed. For example, the reflection, expansion, inside contraction, and outside contraction steps of the Nelder-Mead algorithm may be performed by a Search component to generate four trial points. Different strategies to enlarge the number of points generated by the poll step are also proposed and examined in [45].

4.5 PsdMADS: MADS with Parallel Space Decomposition
The sequential implementation of MADS is recommended for problems whose dimension is reasonably small ($n \leq 50$). PsdMADS was developed [14] to solve larger problems using space decomposition and parallelism. In PsdMADS, the problem is divided into random subspaces of dimension $n_s$, much smaller than $n$. Values of $n_s$ ranging from 2 to 4 are frequently used. Each subproblem is solved by launching a MADS algorithmic component called Worker. An additional MADS algorithmic component called Pollster is launched in the space of dimension $n$ but evaluates a single point generated by a Poll component. In practice, the Pollster rarely improves the incumbent solution, but its presence is necessary to ensure that the theoretical convergence results of PsdMADS are satisfied. The Pollster and Workers are repeatedly launched within an Iteration component.
The mesh sizes of the Pollster and Workers are bounded by a master mesh that is updated at every iteration. These bounds, the Pollster’s single evaluation, and the Worker’s small dimension allow for a fast resolution of each Mads. Figure 6 presents the main algorithmic components involved in PsdMads.

In the original PsdMads implementation described in [14], the management of parallel processes is done using Message Passing Interface (MPI). In the new implementation, OpenMP manages the parallel execution of algorithmic components on main threads. The total number of available threads is $n_t$. The Pollster and Workers are run by Mads components on $n_{mt}$ main threads, with $n_{mt} \leq n_t$ as illustrated by Figure 7. Main thread 0 is used for the Pollster. Main threads 1 to $n_{mt} - 1$ are used for the Workers. Additional threads $n_{mt}$ to $n_t - 1$ are secondary threads.

The main threads are used for algorithmic components and evaluations, whereas the secondary threads are used only for evaluations. The master mesh size update (enlarged or refined) depends on the success of one of the Workers or Pollster. In the new implementation, a finer update control delays the mesh size update until a prescribed minimum number of variables are explored by subproblems solved by Workers.

5 SOFTWARE ARCHITECTURE AND DEVELOPMENT

This section reviews high-level choices made for the different software components of NOMAD. Next, details relative to the processes and tools used while creating this software are mentioned.

5.1 Software Architecture

The previous section illustrated how NOMAD 4 and algorithmic components facilitate the construction of new algorithms, such as PsdMads. Since this new version is written from scratch, an effort was made to ensure that the architecture is modular, with clear and reusable code. The algorithmic components described in Section 3 are building blocks for algorithms. The evaluation queue presented in Section 4 is designed to launch evaluations in parallel. The code is implemented using object-oriented programming. Algorithmic components and evaluation queue are
translated to code as classes and objects. Polymorphism is used: for instance, all algorithmic components are of the base class Step; classes Algorithm and SearchMethod inherit from Step; class SpeculativeSearch derives from SearchMethod (see Figure 8).

Algorithms other than MADS can be added to run standalone optimization or to be embedded in a search step, as, for example, with the NelderMead algorithm. To use a new algorithm during the search step, a specific search component (see Figure 3) must be implemented, along with some parameters controlling the algorithm. Minimally, a function to generate the trial points must be implemented. The functions for insertion of new trial points and their evaluation are provided by
The code is organized into libraries that provide a range of functionalities for the implementation of algorithms to solve optimization problems:

- **Utils**: Math functionalities; parameter definition and checking; output management, including an output queue to correctly display information coming from different threads; file utilities, clock, and other utilities.
- **Eval**: All that relates to the management of evaluations: evaluation queue, evaluator, results of evaluations, and cache for points that have already been evaluated.
- **Algos**: Algorithmic components and algorithms: LH, Mads, NelderMead, QuadraticModelSearch, ParallelSpaceDecomposition, and SgtelibModelSearch, where the SgtelibModelSearch is a more general case of the QuadraticModelSearch algorithm.
- **sgtelib** [61], a library containing many surrogate models developed by B. Talgorn.

### 5.2 Software Development

The development process of NOMAD 4 is done by working closely with students and industrial partners to ensure the most valuable features are added and important issues are promptly addressed. Code quality is verified through unit tests, for classes and methods, and through integration tests, for algorithmic functionality. Performance profiles (presented in Section 6) are processed regularly, comparing NOMAD 4 with NOMAD 3 or with previous versions of NOMAD 4, to establish that development is going in the right direction to efficiently solve optimization problems.

The hot/warm restart feature is an example of software development instigated by collaboration. This new feature makes it possible to continue the solving process after it has started, without having to restart it from the beginning. In the case of hot restart, the user interrupts the solver to change the value of a parameter. With warm restart, the user changes a parameter from a resolution that has already reached a termination condition. In both cases, the solving process is then continued from its current state.

NOMAD 4 is a standalone program coded in C++14 available for Linux, macOS, and Windows. CMake is used for compilation. By default, the OpenMP API for the selected compiler is automatically detected during the CMake configuration and used when building.

Google Test is used for unit tests. We have over 60 unit tests to verify that the methods and classes of key components continue to behave as expected while the software development continues on new features. The tests cover all the spectrum of software functionalities, from high-level component functions such as algorithms to basic low-level mathematical operations on numbers.

Stable code is available and updated frequently at github.com/bbopt/nomad. The unit tests are not part of this public repository.

### 6 COMPUTATIONAL RESULTS

The code of NOMAD 4 differs significantly from that of NOMAD 3; only a few base classes were preserved. Comparing the performance of the two versions is crucial to validate that algorithms have been correctly coded.

Tests are conducted using an in-house application called the Runner. The Runner is designed to compare the performance of different optimization solvers including different versions of NOMAD and various algorithmic choices. The benchmark tests presented below include constrained and unconstrained analytical problems, engineering test problems from the literature, and tests involving parallelism.
Fig. 9. Data profiles obtained on 53 smooth unconstrained problems. The precision to detect if a problem is solved is set to $\tau = 10^{-3}$ (left) and $\tau = 10^{-5}$ (right).

Comparisons are made through data profiles [56]. The vertical axis shows the proportion of problem instances solved within a prescribed tolerance of a parameter $\tau$, and the horizontal axis measures the effort deployed by the compared methods in terms of groups of $n+1$ function evaluations. A steep curve indicates that the corresponding method rapidly improves the solutions. A method having its curve above the others performs better for the prevailing test conditions. The optimization runs are conducted on a series of problems for a given evaluation budget. In the tests below, each graph has two curves, one for NOMAD 3 and the other for NOMAD 4, with the same default parameters.

6.1 Validation on Analytical Problems

Figure 9 compares NOMAD 3 and NOMAD 4 on a collection of 53 unconstrained smooth problems [56] with a number of variables $n$ ranging from 2 to 12 and with a budget of $400(n+1)$ function evaluations. Each version is launched 10 times with different random seeds, resulting in a total of 530 run instances per algorithm. The profiles on the left use a tolerance of $\tau = 10^{-3}$ (the reader is invited to consult [56] for the precise description of the role of $\tau$), and the ones on the right use a smaller tolerance of $\tau = 10^{-5}$. In both cases, the two versions exhibit comparable performance, and there is no clear dominance of one over the other, which indicates that NOMAD 4 performs as well as NOMAD 3.

Figure 10 compares NOMAD 3 and NOMAD 4 on the collection of 18 constrained problems listed in Table 1 with a budget of $800(n+1)$ function evaluations. The number of variables ($n$) varies from 2 to 20, the number of constraints ($m$) ranges from 1 to 15, and 13 of the problems have bounds on the variables. Again, each version is launched with 10 random seeds. The profiles on the left use a tolerance of $\tau = 10^{-3}$, and the ones on the right use a smaller tolerance of $\tau = 10^{-5}$. As with the unconstrained case, NOMAD 3 and NOMAD 4 have a similar performance.

6.2 Tests on PsdMads

The implementations of PsdMads in NOMAD 3 and in NOMAD 4 are significantly different, principally because of the strategy used to perform parallel subproblem optimizations. First we present a comparison of the two PsdMads implementations on two bound constrained variants of the Rosenbrock test problem [39]. The two variants, called SRosenbr50 and SRosenbr250, have $n = 50$ and $n = 250$ variables, respectively, with all starting point coordinates set to 0.5. The lower bounds...
Fig. 10. Data profiles obtained on 18 problems with constraints. The precision to detect if a problem is solved is set to $\tau = 10^{-3}$ (left) and $\tau = 10^{-5}$ (right).

Table 1. Description of the Set of 18 Analytical Problems with Constraints

| #  | Name            | Source | $n$ | $m$ | Bounds |
|----|-----------------|--------|-----|-----|--------|
| 1  | CHENWANG_F2     | [30]   | 8   | 6   | yes    |
| 2  | CHENWANG_F3     | [30]   | 10  | 8   | yes    |
| 3  | CRESCENT        | [13]   | 10  | 2   | no     |
| 4  | DISK            | [13]   | 10  | 1   | no     |
| 5  | G210            | [14]   | 10  | 2   | yes    |
| 6  | G220            | [14]   | 20  | 2   | yes    |
| 7  | HS19            | [43]   | 2   | 2   | yes    |
| 8  | HS83            | [43]   | 5   | 6   | yes    |
| 9  | HS114           | [49]   | 9   | 6   | yes    |
| 10 | MAD6            | [49]   | 5   | 7   | no     |
| 11 | MDO             | [67]   | 10  | 10  | yes    |
| 12 | MEZMONTES       | [32]   | 2   | 2   | yes    |
| 13 | OPTENG_RBF      | [44]   | 3   | 4   | yes    |
| 14 | PENTAGON        | [49]   | 6   | 15  | no     |
| 15 | SNAKE           | [13]   | 2   | 2   | no     |
| 16 | SPRING          | [58]   | 3   | 4   | yes    |
| 17 | TAOWANG_F2      | [65]   | 7   | 4   | yes    |
| 18 | ZHAOWANG_F5     | [69]   | 13  | 9   | yes    |

are all set to $-10$, and the upper bounds are set to 10. Then we consider the B250 and B500 instances from [29] with 60 variables. Both have one inequality constraint. The initial point for B250 is feasible and satisfies $F(x_0) \approx 3678$. The initial point for B500 is infeasible. Finally, the G2 problem [53] with 50 variables and 2 inequality constraints is also considered.

PsdMads is not a deterministic algorithm, because some of its decisions may be affected by the chronological order in which the subproblems produce solutions, which might vary from one run to another. Because of the stochastic nature of PsdMads, both instances of the problem are solved 30 times each to perform a fair comparison. Figure 11 plots the average incumbent objective function value versus the number of function evaluations. The plot also shows the best and worst objective function values.

The two implementations of PsdMads use four workers to perform subproblem optimizations with two randomly selected variables. NOMAD 3 uses MPI with six processes (one process for the pollster, three processes for the regular workers, one process for the cache server, and one process
for the master). To obtain a comparable task distribution, NOMAD 4 uses only $n_t = 4$ main threads for OpenMP (no secondary thread) for the pollster and regular workers (see Figure 7).

On the problems tested, the PsDMADS version in NOMAD 4 has a comparable performance to that of NOMAD 3. The latter performs slightly better on the B500 problem, and the converse is true for the G2 problem.
6.3 Improvements in Solution Times

The previous section compared the performance of PsdMads in terms of the number of function evaluations. We next study the impact of using multiple cores on the overall computational time. The Rosenbrock test problem is not adequate for such comparisons, as it is evaluated nearly instantaneously. We present results on the problem Solar 7 \cite{47}, which requires approximately 5 seconds for each evaluation. This problem simulates the operation of a solar thermal power plant. It has seven bounded variables, one of which is an integer, and six constraints.

Figure 12(a) shows data profiles for different parameter settings of NOMAD 4 (using the OrthoMads algorithm with $2n$ directions), where the x-axis represents the wall-clock time in seconds rather than the number of function evaluations. A method having its corresponding curve above the others performs faster than the others. Figure 12(b) illustrates the speed-up, which plots the wall-clock time in seconds in terms of the number of function evaluations. Low values on the plot indicate better performance. Three cases were tested on a machine containing eight cores, by varying $n_t$, the number of threads used, which here is equal to the number of cores used. In the first case, a single core is used ($n_t = 1$). In the second case, eight cores are used ($n_t = 8$). The third case combines eight cores with the MegaSearchPoll component described in Section 4.4. Each case is launched 10 times with different random seeds, which explains why the plots in Figure 12(b) are not monotone.

Unsurprisingly, using multiple cores in parallel allows NOMAD 4 to generate solutions faster. In addition, using the MegaSearchPoll component improves the speed even more. In summary, running NOMAD 4 on Solar 7 using eight cores as well as the MegaSearchPoll component leads to an overall computational time up to 3.3 times faster than using a single core.

7 CONCLUSION AND FUTURE DEVELOPMENTS

The NOMAD blackbox optimization package has been completely redesigned. The new design defines easily interfaceable building blocks named algorithmic components, for constructing elaborate algorithms. This approach promotes software maintainability and modularity. The architecture is strongly impacted by the requirement of efficiently using a large number of cores.

The new version’s numerical performance is comparable to that of the previous version. However, the code is easily accessible for students and developers. The modularity of the algorithmic
components makes the code flexible and reusable and allows for easy development of new algorithms. Because the architecture is designed with parallelism in mind, NOMAD is now able to manage advantageously a large number of cores.

The first release of NOMAD 4 will serve as a basis for future developments, starting with the integration of some of the improvements from the last 12 years.

Some functionalities of NOMAD 3 are not available in NOMAD 4, and some are new in NOMAD 4. Table 2 enumerates the main differences between NOMAD version 3.9 and NOMAD version 4.2. It also indicates which features will be included in a future distribution of NOMAD and which are discarded. BtMADS will be replaced by the newer DMULTI-MADS methodology, and for parallelism, MPI will be replaced by OpenMP. The table does not report the numerous functionalities that are present in both versions.

Another research direction is the application of the software to real industrial and engineering optimization problems to facilitate modeling, solving, analyzing, and finding solutions for users. Each application has its specificity, which may result in a new generic feature within NOMAD. Close collaborations with industry users are crucial for the development of NOMAD, and new collaborations are welcomed.

REFERENCES
[1] J. Aasi et al. 2013. Einstein@Home all-sky search for periodic gravitational waves in LIGO S5 data. *Phys. Rev. D* 87, 4 (February 2013), 042001. https://doi.org/10.1103/PhysRevD.87.042001
[2] M. A. Abramson, C. Audet, J. W. Chrissis, and J. G. Walston. 2009. Mesh adaptive direct search algorithms for mixed variable optimization. *Optimization Letters* 3, 1 (2009), 35–47. https://doi.org/10.1007/s11590-008-0089-2
[3] M. A. Abramson, C. Audet, J. E. Dennis Jr., and S. Le Digabel. 2009. OrthoMADS: A deterministic MADS instance with orthogonal directions. *SIAM Journal on Optimization* 20, 2 (2009), 948–966. https://doi.org/10.1137/080716980
[4] L. Adjengue, C. Audet, and I. Ben Yahia. 2014. A variance-based method to rank input variables of the mesh adaptive direct search algorithm. *Optimization Letters* 8, 5 (2014), 1599–1610. https://doi.org/10.1007/s11590-013-0688-4
[5] S. Alarie, C. Audet, A. E. Gheribi, M. Kokkolaras, and S. Le Digabel. 2021. Two decades of blackbox optimization applications. *EURO Journal on Computational Optimization* 9 (2021), 100011. https://doi.org/10.1016/j.ejco.2021.100011
[6] N. Amaioua, C. Audet, A. R. Conn, and S. Le Digabel. 2018. Efficient solution of quadratically constrained quadratic subproblems within a direct-search algorithm. *European Journal of Operational Research* 268, 1 (2018), 13–24. https://doi.org/10.1016/j.ejor.2017.10.058
[7] C. Audet. 2014. A survey on direct search methods for blackbox optimization and their applications. In *Mathematics without Boundaries: Surveys in Interdisciplinary Research*, P. M. Pardalos and T. M. Rassias (Eds.). Springer, New York, NY, Chapter 2, 31–56. https://doi.org/10.1007/978-1-4939-1124-0
[8] C. Audet, V. Béchard, and J. Chaouki. 2008. Spent potliner treatment process optimization using a MADS algorithm. *Optimization and Engineering* 9, 2 (2008), 143–160. https://doi.org/10.1007/s11081-007-9030-2
[9] C. Audet, V. Béchard, and S. Le Digabel. 2008. Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search. *Journal of Global Optimization* 41, 2 (2008), 299–318. https://doi.org/10.1007/s10898-007-9234-1

[10] C. Audet and J. Côté-Massicotte. 2019. Dynamic improvements of static surrogates in direct search optimization. *Optimization Letters* 13, 6 (2019), 1433–1447. https://doi.org/10.1007/s11590-019-01452-7

[11] C. Audet and J. E. Dennis Jr. 2001. Pattern search algorithms for mixed variable programming. *SIAM Journal on Optimization* 11, 3 (2001), 573–594. https://doi.org/10.1137/S1052623499352024

[12] C. Audet and J. E. Dennis Jr. 2006. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization* 17, 1 (2006), 188–217. https://doi.org/10.1137/040603371

[13] C. Audet and J. E. Dennis Jr. 2009. A progressive barrier for derivative-free nonlinear programming. *SIAM Journal on Optimization* 20, 1 (2009), 445–472. https://doi.org/10.1137/070692662

[14] C. Audet, J. E. Dennis Jr., and S. Le Digabel. 2008. Parallel space decomposition of the mesh adaptive direct search algorithm. *SIAM Journal on Optimization* 19, 3 (2008), 1150–1170. https://doi.org/10.1137/070707518

[15] C. Audet, K. Diest, S. Le Digabel, L. A. Sweatlock, and D. E. Marthaler. 2013. Metamaterial design by mesh adaptive direct search. In *Numerical Methods for Metamaterial Design*. Topics in Applied Physics, Vol. 127. Springer, Dordrecht, 71–96. https://doi.org/10.1007/978-94-007-6664-8_4

[16] C. Audet, K. J. Dzhahini, M. Kokkolaras, and S. Le Digabel. 2021. Stochastic mesh adaptive direct search for blackbox optimization using probabilistic estimates. *Computational Optimization and Applications* 79, 1 (2021), 1–34. https://doi.org/10.1007/s10589-020-00249-0

[17] C. Audet and W. Hare. 2017. Derivative-free and Blackbox Optimization. Springer, Cham, Switzerland. https://doi.org/10.1007/978-3-319-68913-5

[18] C. Audet, A. Ianni, S. Le Digabel, and C. Tribes. 2014. Reducing the number of function evaluations in mesh adaptive direct search algorithms. *SIAM Journal on Optimization* 24, 2 (2014), 621–642. https://doi.org/10.1137/120895056

[19] C. Audet, A. Ihaddadene, S. Le Digabel, and C. Tribes. 2018. Robust optimization of noisy blackbox problems using the mesh adaptive direct search algorithm. *Optimization Letters* 12, 4 (2018), 675–689. https://doi.org/10.1007/s11590-017-1226-6

[20] C. Audet, M. Kokkolaras, S. Le Digabel, and B. Talgorn. 2018. Order-based error for managing ensembles of surrogates in mesh adaptive direct search. *Journal of Global Optimization* 70, 3 (2018), 645–675. https://doi.org/10.1007/s10898-017-0574-1

[21] C. Audet and S. Le Digabel. 2012. The mesh adaptive direct search algorithm for periodic variables. *Pacific Journal of Optimization* 8, 1 (2012), 103–119. http://www.ybook.co.jp/online2/oppjo/vol8/p103.html

[22] C. Audet, S. Le Digabel, and M. Peyrega. 2015. Linear equalities in blackbox optimization. *Computational Optimization and Applications* 61, 1 (2015), 1–23. https://doi.org/10.1007/s10589-014-9708-2

[23] C. Audet, S. Le Digabel, and C. Tribes. 2016. Dynamic scaling in the mesh adaptive direct search algorithm for blackbox optimization. *Optimization and Engineering* 17, 2 (2016), 333–358. https://doi.org/10.1007/s10581-015-9283-0

[24] C. Audet, S. Le Digabel, and C. Tribes. 2019. The mesh adaptive direct search algorithm for granular and discrete variables. *SIAM Journal on Optimization* 29, 2 (2019), 1164–1189. https://doi.org/10.1137/18M1175872

[25] C. Audet, G. Savard, and W. Zghal. 2008. Multiobjective optimization through a series of single-objective formulations. *SIAM Journal on Optimization* 19, 1 (2008), 188–210. https://doi.org/10.1137/060677513

[26] C. Audet, G. Savard, and W. Zghal. 2010. A mesh adaptive direct search algorithm for multiobjective optimization. *European Journal of Operational Research* 204, 3 (2010), 545–556. https://doi.org/10.1016/j.ejor.2009.11.010

[27] C. Audet and C. Tribes. 2018. Mesh-based Nelder–Mead algorithm for inequality constrained optimization. *Computational Optimization and Applications* 71, 2 (2018), 331–352. https://doi.org/10.1007/s10589-018-0016-0

[28] J. Bigeon, S. Le Digabel, and L. Salomon. 2021. DMulti-MADS: Mesh adaptive direct multisearch for bound-constrained blackbox multiobjective optimization. *Computational Optimization and Applications* 79, 2 (2021), 301–338. https://doi.org/10.1007/s10589-021-00272-9

[29] A. J. Booker, E. J. Cramer, P. D. Frank, J. M. Gablonsky, and J. E. Dennis, Jr. 2007. MoVars: Multidisciplinary Optimization Via Adaptive Response Surfaces. AIAA Paper 2007–1927. (2007). https://doi.org/10.2514/6.2007-1927

[30] X. Chen and N. Wang. 2010. Optimization of short-time gasoline blending scheduling problem with a DNA based hybrid genetic algorithm. *Chemical Engineering and Processing: Process Intensification* 49, 10 (2010), 1076–1083. https://doi.org/10.1016/j.cep.2010.07.014

[31] A. R. Conn and S. Le Digabel. 2013. Use of quadratic models with mesh-adaptive direct search for constrained black box optimization. *Optimization Methods and Software* 28, 1 (2013), 139–158. https://doi.org/10.1080/10556788.2011.623162

[32] L. Dagum and R. Menon. 1998. OpenMP: An industry standard API for shared-memory programming. *IEEE Computational Science and Engineering* 5, 1 (1998), 46–55. https://doi.org/10.1109/99.660313
Numerical Solution of a Minimum Problem. Los Alamos Unclassified Report LA–1492. Los Alamos National Laboratory, Los Alamos, NM.

E. M. Gertz, T. Hiekkalinna, S. Le Digabel, C. Audet, J. D. Terwilliger, and A. A. Schaeffer. 2014. PSEUDOMARKER 2.0: Efficient computation of likelihoods using NOMAD. BMC Bioinformatics 15, 47 (2014), 8. https://doi.org/10.1186/1471-2105-15-47

A. E. Gheribi, C. Audet, S. Le Digabel, E. Bélisle, C. W. Bale, and A. D. Pelton. 2012. Calculating optimal conditions for alloy and process design using thermodynamic and properties databases, the FactSage software and the mesh adaptive direct search algorithm. CALPHAD: Computer Coupling of Phase Diagrams and Thermochemistry 36 (2012), 135–143. https://doi.org/10.1016/j.calphad.2011.06.003

A. E. Gheribi, J.-P. Harvey, E. Bélisle, C. Robelin, P. Chartrand, A. D. Pelton, C. W. Bale, and S. Le Digabel. 2016. Use of a biobjective direct search algorithm in the process design of material science applications. Optimization and Engineering 17, 1 (2016), 27–45. https://doi.org/10.1007/s11081-015-9301-2

A. E. Gheribi, S. Le Digabel, C. Audet, and P. Chartrand. 2013. Identifying optimal conditions for magnesium based alloy design using the mesh adaptive direct search algorithm. Thermochimica Acta 559 (2013), 107–110. https://doi.org/10.1016/j.tca.2013.02.004

A. E. Gheribi, C. Robelin, S. Le Digabel, C. Audet, and A. D. Pelton. 2011. Calculating all local minima on liquidus surfaces using the FactSage software and databases and the mesh adaptive direct search algorithm. Journal of Chemical Thermodynamics 43, 9 (2011), 1323–1330. https://doi.org/10.1016/j.jct.2011.03.021

N. I. M. Gould, D. Orban, and Ph. L. Toint. 2003. CUTeR (and SifDec): A constrained and unconstrained testing environment, revisited. ACM Trans. Math. Software 29, 4 (2003), 373–394. https://doi.org/10.1145/962437.962439

P. Hansen and N. Mladenović. 2001. Variable neighborhood search: Principles and applications. European Journal of Operational Research 130, 3 (2001), 449–467. https://doi.org/10.1016/S0377-2217(00)00100-4

R. E. Hayes, F. H. Bertrand, C. Audet, and S. T. Kolaczkowski. 2003. Catalytic combustion kinetics: Using a direct search algorithm to evaluate kinetic parameters from light-off curves. Canadian Journal of Chemical Engineering 81, 6 (2003), 1192–1199. https://doi.org/10.1002/cjce.5450816068

A. Hibberd, A. M. Hein, and T. M. Eubanks. 2020. Project Lyra: Catching 1I/’Oumuamua - Mission opportunities after 2024. Acta Astronautica 170 (2020), 136–144. https://doi.org/10.1016/j.actaastro.2020.01.018

W. Hock and K. Schittkowski. 1981. Test Examples for Nonlinear Programming Codes. Lecture Notes in Economics and Mathematical Systems, Vol. 187. Springer, Berlin, Germany.

S. Kitayama, M. Arakawa, and K. Yamazaki. 2011. Sequential approximate optimization using radial basis function network for engineering optimization. Optimization and Engineering 12, 4 (2011), 535–557. https://doi.org/10.1007/s11081-010-9118-y

G. Lameynardie. 2020. Sondes locales intensives lors de l’exécution de l’algorithme MADS dans un environnement parallèle. Master’s Thesis. Polytechnique Montréal. https://publications.polymtl.ca/1996/.

S. Le Digabel. 2011. Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm. ACM Trans. Math. Software 37, 4 (2011), 44:1–44:15. https://doi.org/10.1145/1916461.1916468

M. Lemyre Garneau. 2015. https://doi.org/10.3847/1538-4357/ab5856

L. Lukjan and J. Vlček. 2000. Test Problems for Nonsmooth Unconstrained and Linearly Constrained Optimization. Technical Report V-798. ICS AS CR.

M. D. McKay, R. J. Beckman, and W. J. Conover. 1979. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 21, 2 (1979), 239–245. http://www.jstor.org/stable/1268522.

K. Mehran, J. Thomas, R. Saglia, X. Mazzalay, P. Erwin, R. Bender, M. Kluge, and M. Fabricius. 2019. A 40 billion solar-mass black hole in the extreme core of Holm 85. Astrophysical Journal 887, 2 (2019), 195. https://doi.org/10.3847/1538-4357/ab5856

E. Mezura-Montes and C. A. Coello. 2005. Useful infeasible solutions in engineering optimization with evolutionary algorithms. In Proceedings of the 4th Mexican International Conference on Advances in Artificial Intelligence (MICAi ’05). Springer-Verlag, Berlin, 652–662. https://doi.org/10.1007/11579427_66

Z. Michalewicz and M. Schoenauer. 1996. Evolutionary algorithms for constrained parameter optimization problems. Evolutionary Computation 4, 1 (1996), 1–32. https://doi.org/10.1162/evco.1996.4.1.1

M. Minville, D. Cartier, C. Guay, L.-A. Leclaire, C. Audet, S. Le Digabel, and J. Merleau. 2014. Improving process representation in conceptual hydrological model calibration using climate simulations. Water Resources Research 50, 6 (2014), 5044–5073. https://doi.org/10.1002/2013WR013857
[55] N. Mladenović and P. Hansen. 1997. Variable neighborhood search. *Computers and Operations Research* 24, 11 (1997), 1097–1100. https://doi.org/10.1016/S0305-0548(97)00031-2

[56] J. J. Moré and S. M. Wild. 2009. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization* 20, 1 (2009), 172–191. https://doi.org/10.1137/080724083

[57] V. Ramakrishnan, N. M. Nagar, C. Finlez, T. Storchi-Bergmann, R. Slater, A. Schnorr-Muller, R. A. Riffel, C. G. Mundell, and A. Robinson. 2019. Nuclear kinematics in nearby AGN - I. An ALMA perspective on the morphology and kinematics of the molecular CO(2–1) emission. *Monthly Notices of the Royal Astronomical Society* 487, 1 (2019), 444–455. https://doi.org/10.1093/mnras/stz1244

[58] J. F. Rodríguez, J. E. Renaud, and L. T. Watson. 1998. Trust region augmented Lagrangian methods for sequential response surface approximation and optimization. *Journal of Mechanical Design* 120, 1 (1998), 58–66. https://doi.org/10.1115/1.2826677

[59] L. A. Sarrazin-Mc Cann. 2018. *Optimisation et ordonnancement en optimisation sans dérivées*. Master’s Thesis. Polytechnique Montréal. https://publications.polymtl.ca/3099/>

[60] B. Talgorn. 2019. sgteleb: Surrogate Model Library for Derivative-Free Optimization. https://github.com/bbopt/sgteleb

[61] B. Talgorn, C. Audet, M. Kokkolaras, and S. Le Digabel. 2018. Locally weighted regression models for surrogate-assisted design optimization. *Optimization and Engineering* 19, 1 (2018), 213–238. https://doi.org/10.1007/s10589-017-9370-5

[62] B. Talgorn, S. Le Digabel, and M. Kokkolaras. 2015. Statistical surrogate formulations for simulation-based design optimization. *Journal of Mechanical Design* 137, 2 (2015), 021405–1–021405–18. https://doi.org/10.1115/1.4028756

[63] B. Tang. 1993. Orthogonal array-based Latin hypercubes. *J. Amer. Statist. Assoc.* 88, 424 (1993), 1392–1397. https://doi.org/10.2307/2291282

[64] J. Tao and N. Wang. 2008. DNA double helix based hybrid GA for the gasoline blending recipe optimization problem. *Chemical Engineering and Technology* 31, 3 (2008), 440–451. https://doi.org/10.1002/ceat.200700322

[65] V. Torczon. 1997. On the convergence of pattern search algorithms. *SIAM Journal on Optimization* 7, 1 (1997), 1–25. https://doi.org/10.1137/S1052623493250780

[66] C. Tribes, J.-F. Dubé, and J.-Y. Trépanier. 2005. Decomposition of multidisciplinary optimization problems: Formulations and application to a simplified wing design. *Engineering Optimization* 37, 8 (2005), 775–796. https://doi.org/10.1080/03052100500289305

[67] J. Xu, C. Audet, C. E. DiLiberti, W. W. Hauck, T. H. Montague, A. F. Parr, D. Potvin, and D. J. Schuirmann. 2016. Optimal adaptive sequential designs for crossover bioequivalence studies. *Pharmaceutical Statistics* 15, 1 (2016), 15–27. https://doi.org/10.1002/pst.1721

[68] J. Zhao and N. Wang. 2011. A bio-inspired algorithm based on membrane computing and its application to gasoline blending scheduling. *Computers and Chemical Engineering* 35, 2 (2011), 272–283. https://doi.org/10.1016/j.compchemeng.2010.01.008

Received April 2021; revised February 2022; accepted June 2022