Les Cahiers du GERAD ISSN: 0711–2440

NOMAD version 4: Nonlinear optimization with the MADS algorithm

C. Audet, S. Le Digabel, V. Rochon Montplaisir, C. Tribes

G–2021–23

April 2021

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2021
– Bibliothèque et Archives Canada, 2021

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2021
– Library and Archives Canada, 2021

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

NOMAD version 4: Nonlinear optimization with the MADS algorithm

Charles Audet
Sébastien Le Digabel
Viviane Rochon Montplaisir
Christophe Tribes

GERAD & Département de Mathématiques et de Génie Industriel, Polytechnique Montréal, Montréal (Québec), Canada H3C 3A7

charles.audet@gerad.ca
sebastien.le.digabel@gerad.ca
viviane.rochon.montplaisir@gerad.ca
christophe.tribes@gerad.ca

April 2021
Les Cahiers du GERAD
G–2021–23

Abstract : NOMAD is software 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 from 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 is 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.

Keywords: Optimization software, blackbox optimization, derivative-free optimization, mesh adaptive direct search

Acknowledgements: 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.

1 Introduction

NOMAD is software 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 they 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 [45]. 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 and has proven its usefulness in scientific papers as well as in established companies. Our own work includes contributions in hydrology [2, 51, 57], pharmacology [57], metamaterial design [15], alloy design [34, 36, 37], chemical engineering [8, 40] and bioinformatics [33]. Many other researchers use NOMAD in a variety of fields. In astrophysics for example, NOMAD is used for black hole observation [49], for tracking the interstellar object 1I/'Oumuamua [41], for kinematics analysis of galaxies [54], and for gravitational wave detection, in a paper with more than 700 co-authors [1]. Hundreds of applications are reported in the surveys [5, 7, 35], 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 improvement.** 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, 58, 59, 60].
- **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 [3, 6, 9, 18, 27]. Sub-algorithms may also be tailored to exploit surrogate functions [10, 20, 59]. 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, ensure faster convergence [56]. Quadratic models approximate the problem to rapidly find better points [30]. 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 3 [45] were developed. However, these strategies are not well adapted to fully utilize the new abundance of computing resources, with some computer clusters counting cores in thousands.

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 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 paper 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 is 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 ∞ 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 paper [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

$$\min_{x \in \mathcal{X} \subseteq \mathbb{R}^n} \{f(x) \ : \ c(x) \leq 0\}, \tag{1}$$

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 $\mathbb{R}^n_+$.

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 $\mathcal{X}$ called the *mesh*. At iteration k, let x^k denote the current best-known solution. The mesh is defined as $M^k := \left\{x^k + \delta^k D y \ : \ y \in \mathbb{N}^p\right\} \subset \mathbb{R}^n$

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:

$$M^k \quad := \quad \left\{x^k + \delta^k y \ : \ y \in \mathbb{Z}^n\right\} \ \subset \ \mathbb{R}^n.$$

The search step is flexible, and allows the user to explore any finite number of mesh points in the set named S^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, 30]. Additional search strategies include basic Latin hypercube sampling [48] and others based on advanced statistical surrogates [20, 59, 60]. The *Variable Neighbourhood Search* search step [9, 52] available in NOMAD 3 will be added in the future. 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 a so-called frame around x^k whose dimension is set by a frame size parameter Δ^k which is always greater than or equal to the mesh size parameter δ^k. The key elements of the poll step is that the poll set P^k must lie within the frame, and $\left\{x - x^k : x \in P^k\right\}$ must be 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 [31], generalized pattern search [63], ORTHOMADS with $2n$ [3] and $n+1$ [18] directions. Of these, NOMAD 4 currently implements ORTHOMADS with $2n$ directions; some simple direction strategies are also included; other types of poll steps will be included in the future.

Algorithm 2.1 shows MADS with the extreme barrier to handle constraints. It is close to the one presented in the textbook [17].

Algorithm 2.1. Mesh adaptive direct search (MADS)

Given $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$, starting point $x^0 \in \Omega$, and barrier function $f_\Omega(x)$

0. Initialization

$\Delta^0 \in (0, \infty)$	initial frame size parameter
$\tau \in (0, 1)$, with τ rational	mesh-size adjustment parameter
$\varepsilon_{\texttt{stop}} \in [0, \infty)$	stopping tolerance
$k \leftarrow 0$	iteration counter

1. Parameter Update

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

2. Search

   if $f_\Omega(t) < f_\Omega(x^k)$ for some t in a finite subset S^k of the mesh M^k
   set $x^{k+1} \leftarrow t$ and $\Delta^{k+1} \leftarrow \tau^{-1}\Delta^k$ and go to 4
   otherwise go to 3

3. Poll

   let P^k be a poll set constructed around x^k using a positive spanning set
   if $f_\Omega(t) < f_\Omega(x^k)$ for some $t \in P^k$
   set $x^{k+1} \leftarrow t$ and $\Delta^{k+1} \leftarrow \tau^{-1}\Delta^k$
   otherwise
   set $x^{k+1} \leftarrow x^k$ and $\Delta^{k+1} \leftarrow \tau\Delta^k$

4. Termination

   if $\Delta^{k+1} \geq \varepsilon_{\texttt{stop}}$
   increment $k \leftarrow k + 1$ and go to 1
   otherwise stop and return $x^* \leftarrow x^{k+1}$

The MADS algorithm with the progressive barrier to handle constraints is slightly more complex. The rules for accepting a new incumbent solution x^{k+1} are based on both the objective function value and a second function that aggregates the constraint violations. The rules also depend on whether x^{k+1} is feasible or not. A high-level description 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. The pseudo-code from Algorithm 2.1 offers flexibility and may be coded in different ways. The MADS algorithm takes as input an initial point x^0 which 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 use of generic components and tasks during the design of an optimization program favors maintainability, which is a requisite for the revised version of `NOMAD`. Hence, the components can be readily reused in different optimization programs, and can be upgraded when new features are introduced.

The program consists of a hierarchy of components, which are executed one at a time, 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. *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 iterating or testing. 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 [61]. 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 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.

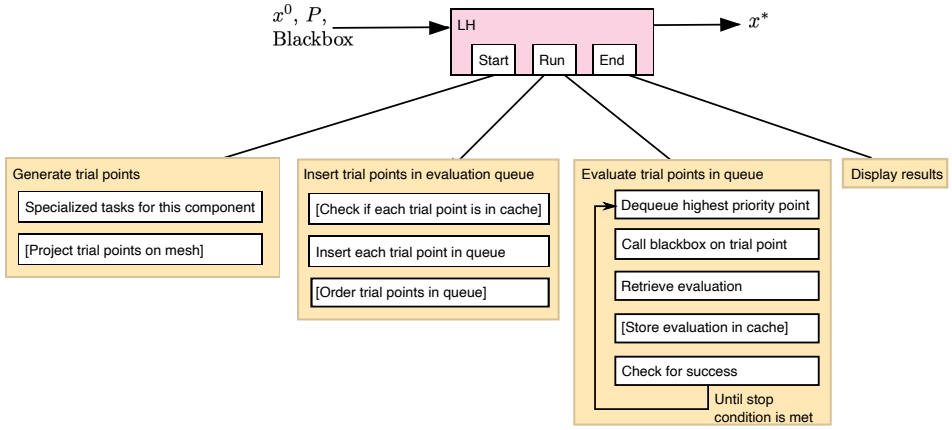


Figure 1: The LH component (Latin hypercube sampling algorithm) with its connected tasks. Tasks within brackets are optional.

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 not found, then 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 $f_\Omega(t)$ and $f_\Omega(x^k)$ seen in the search and poll steps corresponds to the tasks `Call blackbox on trial point`, `Retrieve evaluation`, and `Check for success`.

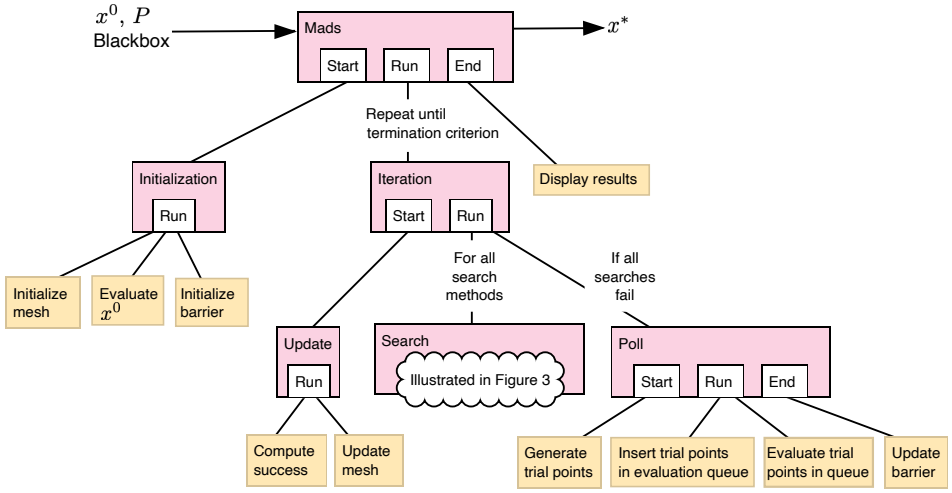


Figure 2: The Mads component with its nested tasks and algorithmic components.

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` components. 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, that they are projected on the current mesh, and that 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` [30] 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 which, 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.

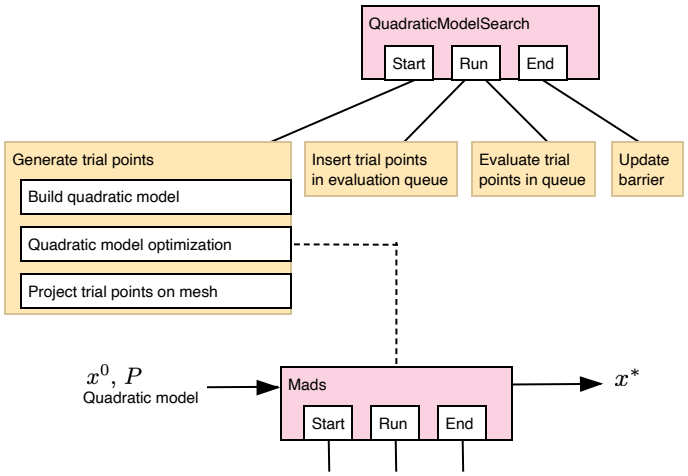


Figure 3: The `QuadraticModelSearch` component with its nested tasks and components. The nested components and tasks of the `Mads` component are not presented.

Various components can be used during a search with some control other 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. Version 3 of `NOMAD` has a

`VariableNeighborhoodSearch` component [9, 39, 52] to attempt escaping local solutions and will be integrated in `NOMAD` 4 in the future.

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 [46] where `NOMAD` is hybridized with a mesh-free linesearch method. It may also be executed repeatedly to solve a biobjective optimization problem through a series of single-objective formulations [25]. Version 3 of `NOMAD` already has the corresponding `BiObjective` component that will be integrated in `NOMAD` 4 in the future.

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 available 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 one 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 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. Nevertheless, grouping points for evaluation is not ideal because there may be an insufficient number of trial points in the queue to fill a group to its maximum size, resulting in unexploited cores. Exploratory work on filling groups of poll sets appears in [44] and will eventually be incorporated into `NOMAD`.

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 criteria 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.

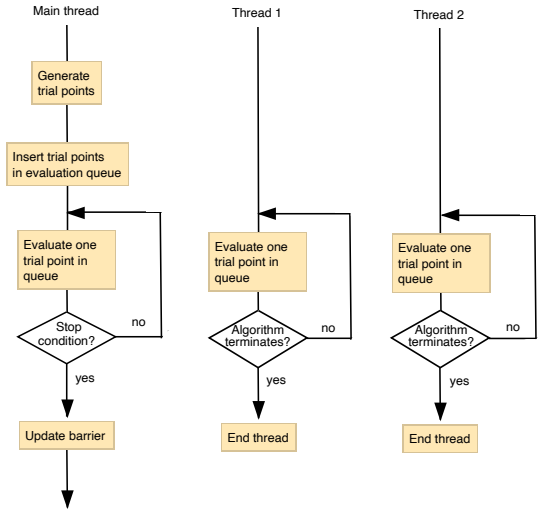


Figure 4: Workflow of the tasks for the evaluation queue using parallel threads.

The management of threads is currently done using `OpenMP`. The user may provide the number of threads n_t to efficiently access the computer cores; otherwise, `OpenMP` computes the number of available threads.

4.4 Grouping trial points generation

In every `Search` and `Poll` components 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 like the speculative search were reworked to provide more points. Different strategies to enlarge the number of points generated by the poll step are also proposed and examined in [44].

4.5 PSDMADS: MADS with parallel space decomposition

The sequential implementation of MADS is recommend from 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

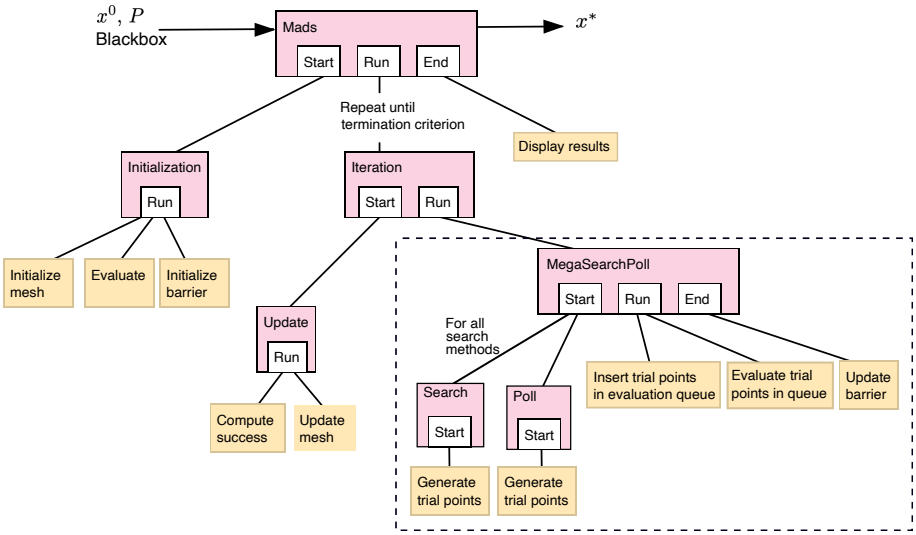


Figure 5: The `MegaSearchPoll` version of MADS with its nested tasks and algorithmic components. A `MegaSearchPoll` component is added to the `Mads` component of Figure 2. The `MegaSearchPoll` component and its dependant tasks and components are presented in a dashed box. The `Run` and `End` tasks of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

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.

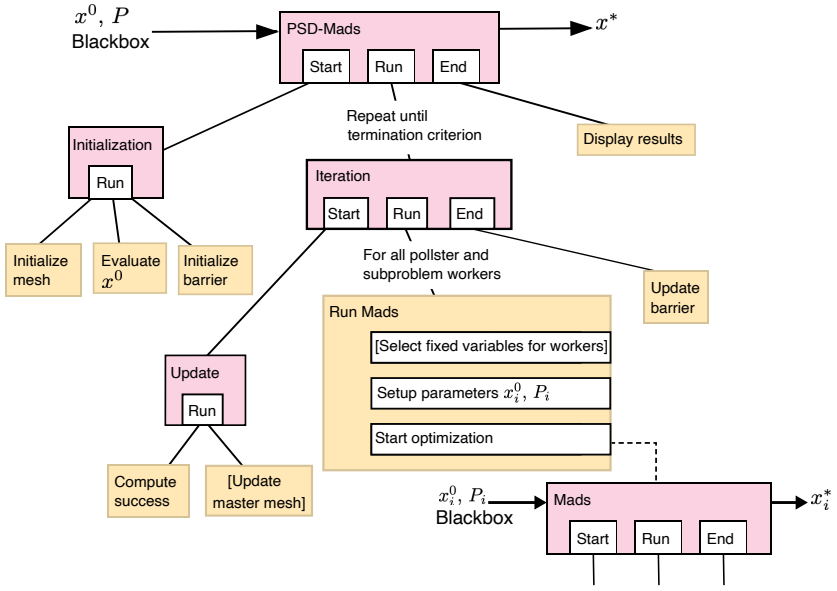


Figure 6: Description of PSDMADS using algorithmic components. The POLLSTER and subproblem WORKERS are run in parallel. The nested components and tasks of `Mads` are not presented.

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.

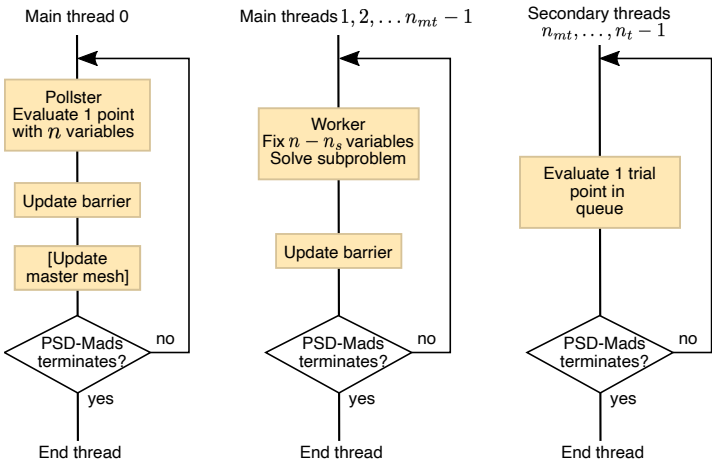


Figure 7: Thread usage in PSDMADS. n_s is the number of variables handled by each WORKER.

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 placed in the development of a modular and reusable architecture. 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 in 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`. Efforts are made to ensure that the code is clear and generic enough to be easily understood and modified, for example by implementing a new search method.

The code is organized into libraries that provide a range of functionalities for programming 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.

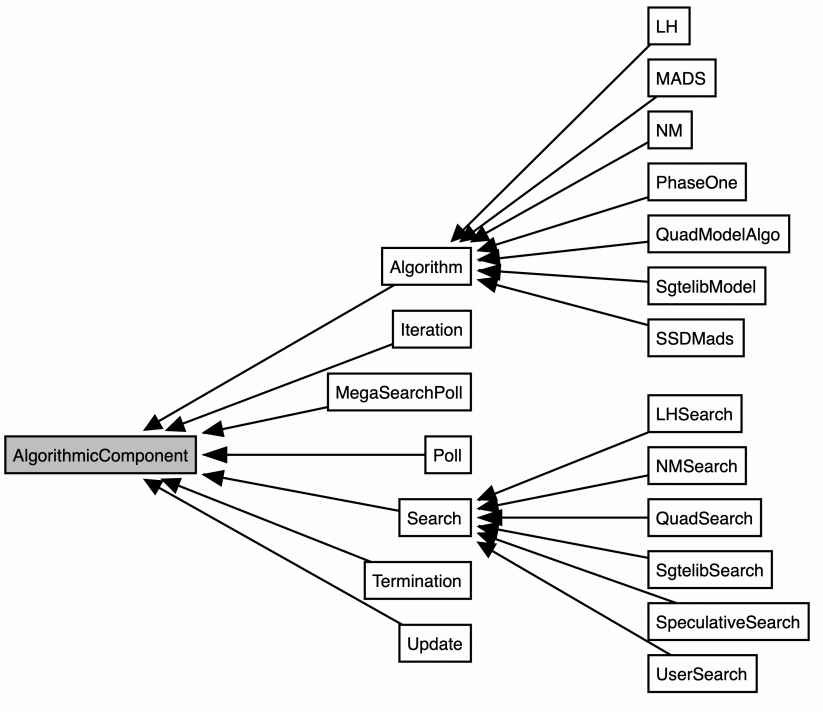


Figure 8: Inheritance graph for the main `AlgorithmicComponent` classes.

- `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` [58], a library containing many surrogate models developed by B. Talgorn.

5.2 Software development

The development process of `NOMAD` 4 is inspired by the Agile software development values. The team meets over daily scrums and biweekly group meetings. They work closely with students and business partners. Features and issues are discussed and added timely to the code. 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.

Customer collaboration is key to development. For instance, one of our key users asked for *hot/warm restart*. 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. This feature was discussed with the user, and added to `NOMAD` 4. The user could test it promptly.

`NOMAD` 4 is a standalone program coded in `C++14` using `OpenMP` when available, on `Linux` and `macOS`. A `Windows` version will be available soon. `CMake` is used for compilation. `Google Test` is used for unit tests. Stable code is available and updated frequently at `github.com/bbopt/nomad`.

6 Computational results

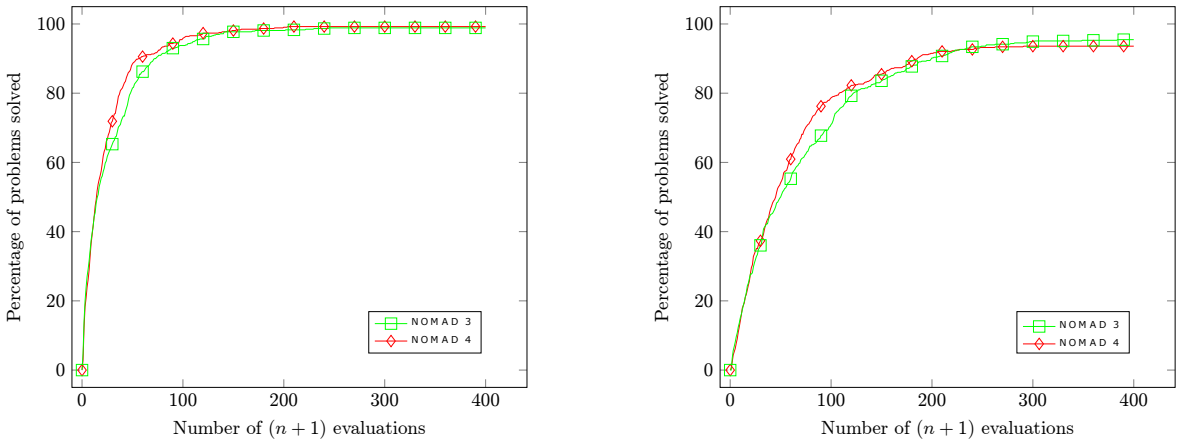
The code of NOMAD 4 differs significantly from that of NOMAD 3; only a few base classes were preserved. Comparing the performance of the two versions is crucial to validate that algorithms have been correctly coded.

Tests are conducted using an in-house application called the Runner. The Runner is designed to compare the performance of different optimization software 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, as well as tests involving parallelism.

Comparisons are made through data profiles [53]. The vertical axis shows the proportion of problem solved within a prescribed tolerance of a parameter τ 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 all default parameters, except that the direction type is set to ORTHO 2N (NOMAD 3 has a default called ORTHO N+1 QUAD direction type that is not yet implemented in NOMAD 4) and the ordering of the points before evaluation is set to the last direction of success (NOMAD 3 has a default ordering based on quadratic models that is not yet implemented in NOMAD 4).

6.1 Validation on analytical problems

Figure 9 compares NOMAD 3 and 4 on a collection of 53 unconstrained smooth problems [53] 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. The profiles on the left use a tolerance of $\tau = 10^{-2}$ (the reader is invited to consult [53] for the precise description of the role of τ) and the ones on the right use a smaller tolerance of $\tau = 10^{-4}$. In both cases, the two versions exhibit comparable performance, and there is no clear dominance of one over the other.

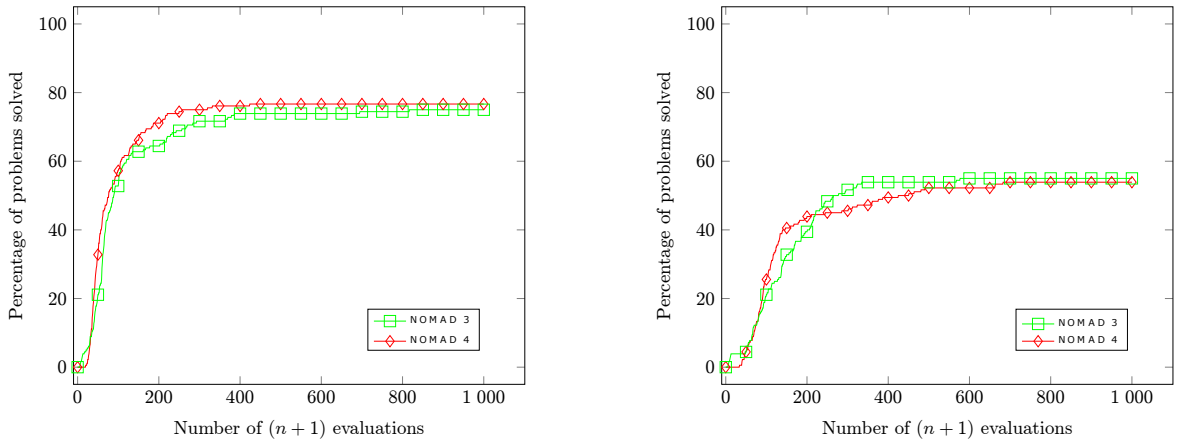


(a) $\tau = 10^{-2}$. **(b)** $\tau = 10^{-4}$.

Figure 9: Data profiles obtained on 53 smooth unconstrained problems. The precision to detect if a problem is solved is set to $\tau = 10^{-2}$ (left) and $\tau = 10^{-4}$ (right).

Figure 10 compares NOMAD 3 and NOMAD 4 on the collection of 18 constrained problems listed in Table 1 with a budget of $1000(n+1)$ 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 $\tau = 10^{-2}$ and the ones on the right use a smaller tolerance of $\tau = 10^{-4}$. As with the unconstrained case, NOMAD 3 and 4 have a similar performance.



(a) $\tau = 10^{-2}$. **(b)** $\tau = 10^{-4}$.

Figure 10: Data profiles obtained on 18 problems with constraints. The precision to detect if a problem is solved is set to $\tau = 10^{-2}$ (left) or $\tau = 10^{-4}$ (right).

Table 1: Description of the set of 18 analytical problems with constraints.

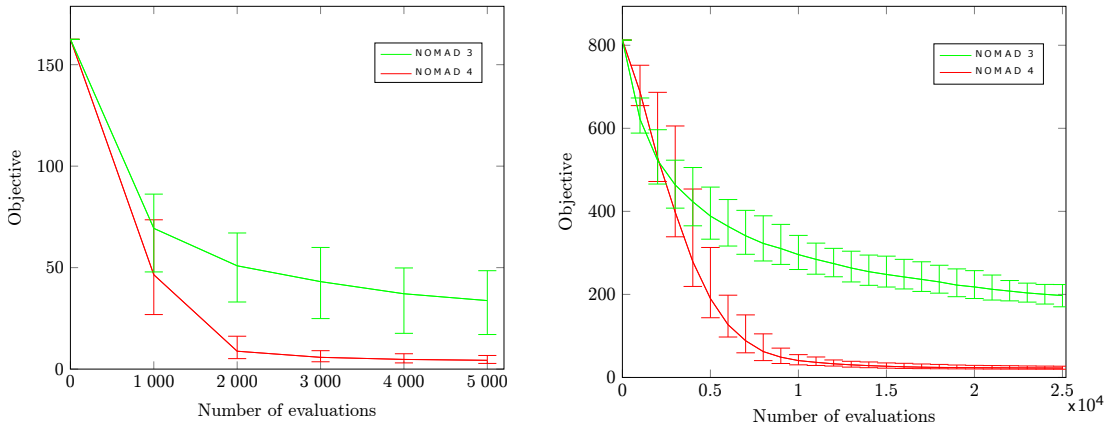
#	Name	Source	n	m	Bnds
1	CHENWANG_F2	[29]	8	6	yes
2	CHENWANG_F3	[29]	10	8	yes
3	CRESCENT	[13]	10	2	no
4	DISK	[13]	10	1	no
5	G210	[14]	10	2	yes
6	G220	[14]	20	2	yes
7	HS19	[42]	2	2	yes
8	HS83	[42]	5	6	yes
9	HS114	[47]	9	6	yes
10	MAD6	[47]	5	7	no
11	MDO	[64]	10	10	yes
12	MEZMONTES	[50]	2	2	yes
13	OPTENG_RBF	[43]	3	4	yes
14	PENTAGON	[47]	6	15	no
15	SNAKE	[13]	2	2	no
16	SPRING	[55]	3	4	yes
17	TAOWANG_F2	[62]	7	4	yes
18	ZHAOWANG_F5	[65]	13	9	yes

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. We present a comparison of the two PsdMads implementations on two bound constrained variants of the Rosenbrock test problem [38]. 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 are all set to -10 and the upper bounds are set to 10.

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 4 workers to perform subproblem optimizations with 2 randomly selected variables among the 60 available. NOMAD 3 uses MPI with 6 processes (1 process for



(a) SRosenbr50. (b) SRosenbr250.

Figure 11: Convergence plot for 30 runs of PSDMADS on SRosenbr50 (left) and SRosenbr250 (right). The solid line is for the best average objective function value and the error bars represent the minimum and maximum objective function values after 1, 1000, 2000, . . . , 100n times the number of blackbox evaluations.

the pollster, 3 processes for the regular workers, 1 process for the cache server and 1 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).

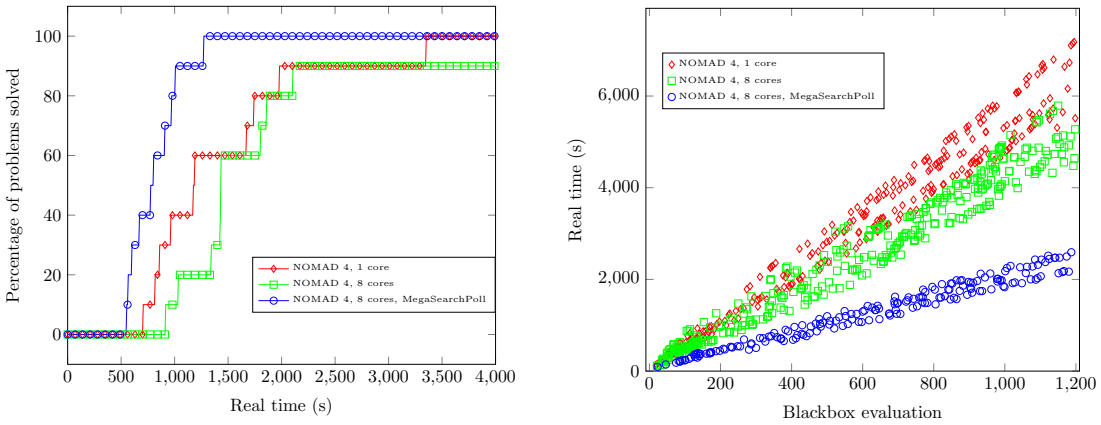
For both variants, the PSDMADS version in `NOMAD` 4 outperforms that of `NOMAD` 3. The worst performance of the 30 PSDMADS runs using `NOMAD` 4 is significantly better than the best run of `NOMAD` 3.

6.3 Improvements in solution times

The previous section compared the performance of PSDMADS in terms of 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 [32], which requires approximately 5 seconds for each evaluation. This problem simulates the operation of a solar thermal power plant. It has 7 variables, one of which is integer, 6 constraints, and variables are bounded.

Figure 12a shows data profiles, for different parameter settings of `NOMAD` 4, 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 12b 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 8 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, 8 cores are used ($n_t = 8$). The third case combines 8 cores with the `MegaSearchPoll` component described in Section 4.4. Each case is launched ten times with different random seeds.

Without surprise, 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 8 cores as well as the `MegaSearchPoll` component leads to an overall computational time up to 3.3 times faster than using a single core.



(a) Data profiles with respect to time; $\tau = 10^{-2}$. **(b) Speed-up.**

Figure 12: Effect of using multiple cores for NOMAD 4, for constrained problem Solar 7, where a single evaluation takes about five seconds.

7 Conclusion

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, 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:

- BiMads [25], MultiMads [26] and DMulti-Mads [28] for multiobjective optimization;
- RobustMads [19] and StoMads [16] for robust and stochastic optimization;
- Variable Neighbourhood Search [9, 52] to escape locally optimal solutions;
- Categorical [11] and periodical variables [21].

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, that may result in a new generic feature within NOMAD. Close collaborations with industry users is crucial for the development of NOMAD. Contact us for projects, we look forward working with you.

References

[1] J. Aasi et al. Einstein@Home all-sky search for periodic gravitational waves in LIGO S5 data. Phys. Rev. D, 87:042001, February 2013.

[2] M.A. Abramson, C. Audet, J.W. Chrissis, and J.G. Walston. Mesh Adaptive Direct Search Algorithms for Mixed Variable Optimization. Optimization Letters, 3(1):35–47, 2009.

[3] M.A. Abramson, C. Audet, J.E. Dennis, Jr., and S. Le Digabel. OrthoMADS: A Deterministic MADS Instance with Orthogonal Directions. SIAM Journal on Optimization, 20(2):948–966, 2009.

[4] L. Adjengue, C. Audet, and I. Ben Yahia. A variance-based method to rank input variables of the Mesh Adaptive Direct Search algorithm. Optimization Letters, 8(5):1599–1610, 2014.

[5] S. Alarie, C. Audet, A.E. Gheribi, M. Kokkolaras, and S. Le Digabel. Two decades of blackbox optimization applications. Technical Report G–2020–58, Les cahiers du GERAD, 2020.

[6] N. Amaioua, C. Audet, A.R. Conn, and S. Le Digabel. Efficient solution of quadratically constrained quadratic subproblems within a direct-search algorithm. European Journal of Operational Research, 268(1):13–24, 2018.

[7] C. Audet. A survey on direct search methods for blackbox optimization and their applications. In P.M. Pardalos and T.M. Rassias, editors, Mathematics without boundaries: Surveys in interdisciplinary research, chapter 2, pages 31–56. Springer, New York, NY, 2014.

[8] C. Audet, V. Béchard, and J. Chaouki. Spent potliner treatment process optimization using a MADS algorithm. Optimization and Engineering, 9(2):143–160, 2008.

[9] C. Audet, V. Béchard, and S. Le Digabel. Nonsmooth optimization through Mesh Adaptive Direct Search and Variable Neighborhood Search. Journal of Global Optimization, 41(2):299–318, 2008.

[10] C. Audet and J. Côté-Massicotte. Dynamic improvements of static surrogates in direct search optimization. Optimization Letters, 13(6):1433–1447, 2019.

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

[12] C. Audet and J.E. Dennis, Jr. Mesh Adaptive Direct Search Algorithms for Constrained Optimization. SIAM Journal on Optimization, 17(1):188–217, 2006.

[13] C. Audet and J.E. Dennis, Jr. A Progressive Barrier for Derivative-Free Nonlinear Programming. SIAM Journal on Optimization, 20(1):445–472, 2009.

[14] C. Audet, J.E. Dennis, Jr., and S. Le Digabel. Parallel Space Decomposition of the Mesh Adaptive Direct Search Algorithm. SIAM Journal on Optimization, 19(3):1150–1170, 2008.

[15] C. Audet, K. Diest, S. Le Digabel, L.A. Sweatlock, and D.E. Marthaler. Metamaterial Design by Mesh Adaptive Direct Search. In Numerical Methods for Metamaterial Design, volume 127 of Topics in Applied Physics, pages 71–96. Springer, Dordrecht, 2013.

[16] C. Audet, K.J. Dzahini, M. Kokkolaras, and S. Le Digabel. Stochastic mesh adaptive direct search for blackbox optimization using probabilistic estimates. Technical Report G–2019–30, Les cahiers du GERAD, 2021. To appear in Computational Optimization and Applications.

[17] C. Audet and W. Hare. Derivative-Free and Blackbox Optimization. Springer Series in Operations Research and Financial Engineering. Springer, Cham, Switzerland, 2017.

[18] C. Audet, A. Ianni, S. Le Digabel, and C. Tribes. Reducing the Number of Function Evaluations in Mesh Adaptive Direct Search Algorithms. SIAM Journal on Optimization, 24(2):621–642, 2014.

[19] C. Audet, A. Ihaddadene, S. Le Digabel, and C. Tribes. Robust optimization of noisy blackbox problems using the Mesh Adaptive Direct Search algorithm. Optimization Letters, 12(4):675–689, 2018.

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

[21] C. Audet and S. Le Digabel. The mesh adaptive direct search algorithm for periodic variables. Pacific Journal of Optimization, 8(1):103–119, 2012.

[22] C. Audet, S. Le Digabel, and M. Peyrega. Linear equalities in blackbox optimization. Computational Optimization and Applications, 61(1):1–23, 2015.

[23] C. Audet, S. Le Digabel, and C. Tribes. Dynamic scaling in the mesh adaptive direct search algorithm for blackbox optimization. Optimization and Engineering, 17(2):333–358, 2016.

[24] C. Audet, S. Le Digabel, and C. Tribes. The Mesh Adaptive Direct Search Algorithm for Granular and Discrete Variables. SIAM Journal on Optimization, 29(2):1164–1189, 2019.

[25] C. Audet, G. Savard, and W. Zghal. Multiobjective Optimization Through a Series of Single-Objective Formulations. SIAM Journal on Optimization, 19(1):188–210, 2008.

[26] C. Audet, G. Savard, and W. Zghal. A mesh adaptive direct search algorithm for multiobjective optimization. European Journal of Operational Research, 204(3):545–556, 2010.

[27] C. Audet and C. Tribes. Mesh-based Nelder-Mead algorithm for inequality constrained optimization. Computational Optimization and Applications, 71(2):331–352, 2018.

[28] J. Bigeon, S. Le Digabel, and L. Salomon. DMulti-MADS: Mesh adaptive direct multisearch for bound-constrained blackbox multiobjective optimization. Technical Report G–2020–25, Les cahiers du GERAD, 2021. To appear in Computational Optimization and Applications.

[29] X. Chen and N. Wang. Optimization of short-time gasoline blending scheduling problem with a DNA based hybrid genetic algorithm. Chemical Engineering and Processing: Process Intensification, 49(10):1076–1083, 2010.

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

[31] E. Fermi and N. Metropolis. Numerical solution of a minimum problem. Los Alamos Unclassified Report LA–1492, Los Alamos National Laboratory, Los Alamos, USA, 1952.

[32] M. Lemyre Garneau. Modelling of a solar thermal power plant for benchmarking blackbox optimization solvers. Master's thesis, Polytechnique Montréal, 2015. Code available at https://github.com/bbopt/solar.

[33] E.M. Gertz, T. Hiekkalinna, S. Le Digabel, C. Audet, J.D. Terwilliger, and A.A. Schaffer. PSEUDOMARKER 2.0: efficient computation of likelihoods using NOMAD. BMC Bioinformatics, 15(47):8, 2014.

[34] A.E. Gheribi, C. Audet, S. Le Digabel, E. Bélisle, C.W. Bale, and A.D. Pelton. 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, 36:135–143, 2012.

[35] A.E. Gheribi, J.-P. Harvey, E. Bélisle, C. Robelin, P. Chartrand, A.D. Pelton, C.W. Bale, and S. Le Digabel. Use of a biobjective direct search algorithm in the process design of material science applications. Optimization and Engineering, 17(1):27–45, 2016.

[36] A.E. Gheribi, S. Le Digabel, C. Audet, and P. Chartrand. Identifying optimal conditions for magnesium based alloy design using the Mesh Adaptive Direct Search algorithm. Thermochimica Acta, 559(0):107–110, 2013.

[37] A.E. Gheribi, C. Robelin, S. Le Digabel, C. Audet, and A.D. Pelton. Calculating all local minima on liquidus surfaces using the FactSage software and databases and the Mesh Adaptive Direct Search algorithm. The Journal of Chemical Thermodynamics, 43(9):1323–1330, 2011.

[38] N.I.M. Gould, D. Orban, and Ph.L. Toint. CUTEr (and SifDec): A Constrained and Unconstrained Testing Environment, revisited. ACM Transactions on Mathematical Software, 29(4):373–394, 2003.

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

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

[41] A. Hibberd, A.M. Hein, and T.M. Eubanks. Project Lyra: Catching 1I/'Oumuamua - Mission opportunities after 2024. Acta Astronautica, 170:136–144, 2020.

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

[43] S. Kitayama, M. Arakawa, and K. Yamazaki. Sequential approximate optimization using radial basis function network for engineering optimization. Optimization and Engineering, 12(4):535–557, 2011.

[44] G. Lameynardie. Sondes locales intensives lors de l'exécution de l'algorithme MADS dans un environnement parallèle. Master's thesis, Polytechnique Montréal, 2020.

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

[46] G. Liuzzi and K. Truemper. Parallelized hybrid optimization methods for nonsmooth problems using NOMAD and linesearch. Computational and Applied Mathematics, 37(3):3172–3207, 2018.

[47] L. Lukšan and J. Vlček. Test problems for nonsmooth unconstrained and linearly constrained optimization. Technical Report V–798, ICS AS CR, 2000.

[48] 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.

[49] K. Mehrgan, J. Thomas, R. Saglia, X. Mazzalay, P. Erwin, R. Bender, M. Kluge, and M. Fabricius. A 40 Billion Solar-mass Black Hole in the Extreme Core of Holm 15A, the Central Galaxy of Abell 85. The Astrophysical Journal, 887(2):195, 2019.

[50] E. Mezura-Montes and C.A. Coello. Useful Infeasible Solutions in Engineering Optimization with Evolutionary Algorithms. In Proceedings of the 4th Mexican International Conference on Advances in Artificial Intelligence, MICAI'05, pages 652–662, Berlin, Heidelberg, 2005. Springer-Verlag.

[51] M. Minville, D. Cartier, C. Guay, L.-A. Leclaire, C. Audet, S. Le Digabel, and J. Merleau. Improving process representation in conceptual hydrological model calibration using climate simulations. Water Resources Research, 50:5044–5073, 2014.

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

[53] J.J. Moré and S.M. Wild. Benchmarking Derivative-Free Optimization Algorithms. SIAM Journal on Optimization, 20(1):172–191, 2009.

[54] V. Ramakrishnan, N.M. Nagar, C. Finlez, T. Storchi-Bergmann, R. Slater, A. Schnorr-Muller, R.A. Riffel, C.G. Mundell, and A. Robinson. 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):444–455, 2019.

[55] J.F. Rodríguez, J.E. Renaud, and L.T. Watson. Trust Region Augmented Lagrangian Methods for Sequential Response Surface Approximation and Optimization. Journal of Mechanical Design, 120(1):58–66, 1998.

[56] L.A. Sarrazin-Mc Cann. Optimisation et ordonnancement en optimisation sans dérivées. Master's thesis, Polytechnique Montréal, 2018.

[57] S. Séguin, P. Côté, and C. Audet. Scenario tree modeling for stochastic short-term hydropower operations planning. Journal of Water Resources Planning and Management, 143(12):04017073–1–12, 2017.

[58] B. Talgorn. sgtelib: Surrogate model library for Derivative-Free Optimization, 2019.

[59] 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.

[60] 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.

[61] B. Tang. Orthogonal array-based latin hypercubes. Journal of the American Statistical Association, 88(424):1392–1397, 1993.

[62] J. Tao and N. Wang. DNA Double Helix Based Hybrid GA for the Gasoline Blending Recipe Optimization Problem. Chemical Engineering and Technology, 31(3):440–451, 2008.

[63] V. Torczon. On the convergence of pattern search algorithms. SIAM Journal on Optimization, 7(1):1–25, 1997.

[64] C. Tribes, J.-F. Dubé, and J.-Y. Trépanier. Decomposition of multidisciplinary optimization problems: formulations and application to a simplified wing design. Engineering Optimization, 37(8):775–796, 2005.

[65] J. Zhao and N. Wang. A bio-inspired algorithm based on membrane computing and its application to gasoline blending scheduling. Computers and Chemical Engineering, 35(2):272–283, 2011.