Algorithm xxx: NOMAD: Nonlinear Optimization with the MADS algorithm

Sébastien Le Digabel [*]

August 27, 2010

Abstract

NOMAD is software that implements the MADS algorithm (Mesh Adaptive Direct Search) for blackbox optimization under general nonlinear constraints. Blackbox optimization is about optimizing functions that are usually given as costly programs with no derivative information and no function values returned for a significant number of calls attempted. NOMAD is designed for such problems and aims for the best possible solution with a small number of evaluations. The objective of this paper is to describe the underlying algorithm, the software's functionalities, and its implementation.

Keywords: Optimization software, MADS (Mesh Adaptive Direct Search), constrained optimization, nonlinear optimization, blackbox optimization.

1 Introduction and motivation

NOMAD is a software package designed to solve optimization problems of the form

$$\min_{x\in\Omega} \quad f(x) \tag{1}$$

or biobjective optimization problems as

$$\min_{x\in\Omega} \quad \big(f_1(x), f_2(x)\big) \tag{2}$$

where $\Omega = \{x \in X : c_j(x) \leq 0, j \in J\} \subset \mathbb{R}^n$, $f, f_1, f_2, c_j : X \to \mathbb{R} \cup \{\infty\}$ for all $j \in J = \{1, 2, \ldots, m\}$, and X is a subset of $\mathbb{R}^n$. Constraints may be of several types

[*]GERAD and Département de mathématiques et de génie industriel, Ecole Polytechnique de Montréal, C.P. 6079, Succ. Centre-ville, Montréal, Québec H3C 3A7 Canada, www.gerad.ca/Sebastien.Le.Digabel, Sebastien.Le.Digabel@gerad.ca.

(blackboxes, nonlinear inequalities, yes/no or hidden constraints) and their treatment is discussed in 2.3.2. Variables x may also be integer, binary, or categorical (such variables are defined in Section 2.3.6).

The functions f, f_1, f_2, and c_j's defining the problem are typically simulations which possess no exploitable properties such as derivatives, and they may fail to evaluate at some trial points, even feasible ones. They may also take a long time to evaluate, and be noisy. In this paper, the term *blackbox* problem is used to denote this problem class.

Direct search methods are meant for such a context since they use only function evaluations to drive their exploration in the space of variables and they can deal with missing function values. Several methods are available and the reader is invited to consult the recent book [26] for an in-depth description of derivative-free optimization methods.

NOMAD is coded in C++ and implements the Mesh Adaptive Direct Search algorithm (MADS) [15], a direct search method with a rigorous convergence theory based on the nonsmooth calculus [25]. A MATLAB version developed by Mark Abramson is also available. MATLAB users may also try the MADS algorithm directly from the GADS package [36].

NOMAD has been under development since 2000. Its first version implemented the Generalized Pattern Search (GPS) algorithm [42, 13], a particular case of MADS, which was developed in 2006 and integrated into NOMAD since then. The most recent change occurred in 2008 with the release of version 3, which was a complete restructuring of the code. This document focuses on versions 3 and above. Previous algorithms like GPS are still options in NOMAD.

NOMAD can handle general nonlinear constraints in a computationally effective and theoretically sound way. It is at its best for blackbox problems with less than 50 variables. For larger problems, the PSD-MADS [17] algorithm launching several parallel MADS instances on subproblems should be considered. PSD-MADS is integrated in the NOMAD package as a separated program and is described in Section 2.3.5. Biobjective optimization is treated with the BiMADS algorithm [21]. The present paper focuses more on MADS than on BiMADS and PSD-MADS since these two methods use MADS in subproblems.

Examples of applications solved by NOMAD may be consulted in [3, 6, 9, 11, 18, 20, 19]. In addition to academic researchers, NOMAD users include companies such as Airbus, ExxonMobil, GM, and recently Hydro-Québec. Other codes are available for derivative-free optimization and lists may be consulted at [37] or in the recent study [39]. However, NOMAD is the only available program that combines a wide range of features including the use of the MADS algorithm for single-objective and biobjective problems, under general inequality constraints, with variables that can be continuous, integer, binary, or categorical, and with scalar and parallel versions. Numerical tests comparing several codes including NOMAD may be consulted in [31] for one difficult blackbox problem, although only unconstrained variants of the problem were considered because most of the codes could not deal with them. Other tests are given in [39] for a set of academic problems, and in [28] for a series of multi-objective problems. One conclusion of these comparisons is that no algorithm is dominant over the others and that their efficiency differ dependently of the problem to solve. However, [39] and [28] consider only unconstrained or bound constrained problems with continuous variables and scalar algorithms, and only [28] focuses on more than one objective. The test problems in [39] are also exempt of noise and have no hidden constraints, situations typically encountered in blackboxes for which NOMAD is designed.

Source code documentation generated with Doxygen [43] is available in the HTML format on the NOMAD website. A user guide [34] may also be found inside the NOMAD package. It explains in details how to install and use NOMAD and complements the present paper intended to explain the internal mechanics of the software and its links to the MADS algorithm. Moreover the user guide is a dynamic document that will evolve with the different versions of NOMAD, whereas the present paper should stay consistent throughout the software's life.

This paper is organized as follows: Section 2 gives a general overview of the MADS algorithm and its most recent developments. Section 3 describes the NOMAD implementation details and its functionalities and Section 4 proposes future developments and conclusions.

2 The MADS algorithm

The MADS algorithm [15] extends the GPS algorithm [42, 13], which is itself an extension of the coordinate (or compass) search [30]. This section first gives a description of the algorithm, then summarizes the convergence analysis, and finally lists the features and different extensions of the method.

2.1 Description of the algorithm

MADS is an iterative method where the blackbox functions are evaluated at some trial points lying on a *mesh* whose discrete structure is defined at iteration k by:

$$M_k = \bigcup_{x \in V_k} \{x + \Delta_k^m D z : z \in \mathbb{N}^{n_D}\}, \tag{3}$$

where $\Delta_k^m \in \mathbb{R}^+$ is the *mesh size parameter*, V_k is described in the next paragraph, and D is a $n \times n_D$ matrix representing a fixed finite set of n_D directions in $\mathbb{R}^n$. More precisely D is called the set of *mesh directions* and is constructed so that $D = GZ$ where G is a nonsingular $n \times n$ matrix and Z is a $n \times n_D$ integer matrix. The matrix D is often taken as $D = [I_n \; -I_n]$ where I_n is the identical matrix in dimension n.

V_k is the set of points where the objective function and constraints have been evaluated by the start of iteration k. V_0 contains the starting points (providing several starting points is allowed). In NOMAD, the set V_k corresponds to the *cache* data structure memorizing all previously evaluated points so that no double evaluations occur. The cache is the fundamental data structure of NOMAD and will be detailed in Section 3.3.

Each MADS iteration is composed of three steps: the *poll* , the *search* , and updates. The search step is flexible and allows the creation of trial points anywhere on the mesh. That liberty is due to the fact that the convergence analysis does not rely on this step. Specific search steps may be tailored for a specific application or one may define generic strategies such as Latin-Hypercube (LH) sampling [41] or variable neighborhood search (VNS) [9]. Indeed the flexibility of the search step is exploited by NOMAD and users may define their own search strategy.

The poll step is more rigidly defined since the convergence analysis relies on it. It explores the mesh near the current iterate x_k with the following set of *poll trial points*:

$$P_k \quad = \quad \{x_k + \Delta_k^m d : d \in D_k\} \subset M_k \ , \tag{4}$$

where D_k is the set of poll directions. Each column of D_k is an integer combination of the columns of D and D_k is such that its columns form a positive spanning set. Points of P_k are generated so that their distance to the *poll center* x_k is bounded below by the *poll size parameter* $\Delta_k^p \in \mathbb{R}^+$. The MADS algorithm defines the links between the mesh and poll size parameters: Δ_k^m is always smaller than Δ_k^p and Δ_k^m is reduced faster than Δ_k^p after failures. Figure 1 illustrates this relation between the mesh and poll size parameters. As poll trial points are generated on the mesh at distance Δ_k^p from the poll center, this implies that the number of possible choices for the trial points is larger and larger. The MADS algorithm defines types of directions that, once normalized, become asymptotically dense in the unit sphere. The way that these directions may be generated will be discussed in 2.3.

The poll and search steps generate trial points on the mesh. The blackbox functions are evaluated at these points and the algorithm determines if each evaluation is a success or not, according to certain rules that will be described later on.

At the end of iteration k, an update step determines the iteration status (success or failure) and the next iterate x_{k+1} is chosen. It corresponds to the most promising success or stays at x_k. The mesh size parameter is also updated with

$$\Delta_{k+1}^m \quad = \quad \tau^{w_k} \Delta_k^m \tag{5}$$

where $\tau > 1$ is a fixed rational number and w_k is a finite integer, positive or null if iteration k is a success, or strictly negative if the iteration failed. In other words the mesh size parameter is decreased after failures and possibly increased after successes.

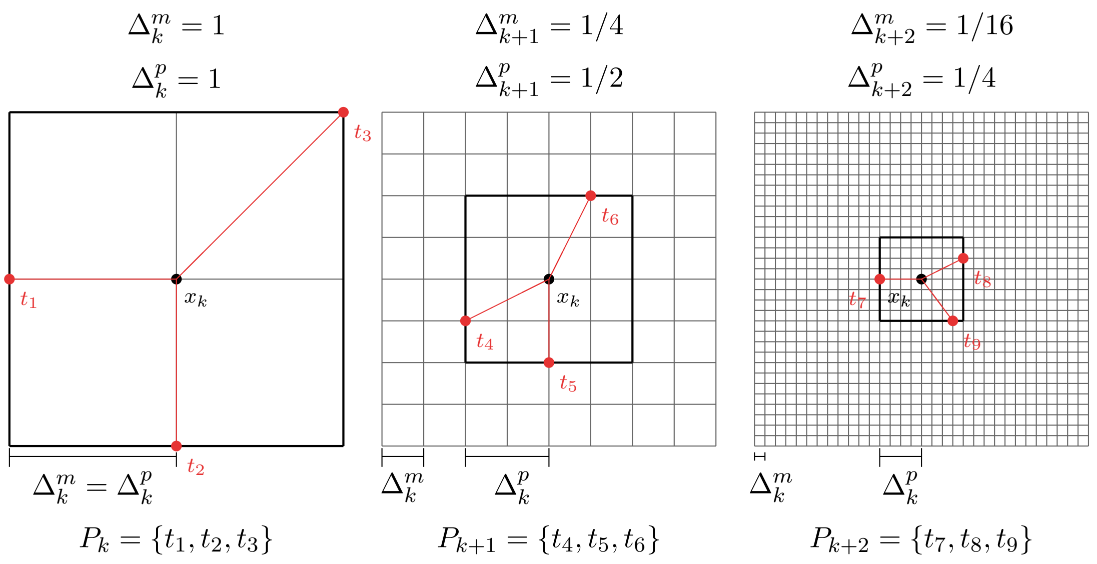

Figure 1: Example of different mesh configurations with $n = 2$. Thin lines represent the mesh of size Δ_k^m, and thick lines the points at distance Δ_k^p from x_k in norm L_∞. Poll trial points are illustrated with random LT-MADS $n + 1$ directions (see Section 2.3.1). The mesh is reduced between the situations to show that Δ_k^m is reduced faster than Δ_k^p. As the poll trial points lie at the intersection of the thick and thin lines, the number of possible locations grows larger and larger.

The poll size parameter is also updated in this step according to rules depending on the implementation. For example, with the LT-MADS method described in 2.3.1, $\Delta_k^m \leq \Delta_k^p \leq 1$ and $\Delta_k^p = \sqrt{\Delta_k^m}$.

Figure 2 concludes this MADS overview by giving a high-level description of the algorithm.

THE MADS ALGORITHM

- INITIALIZATION: Let $x_0 \in V_0$ be a starting point, G and Z be the matrices used to define D, and τ be the rational number used to update the mesh size parameter. Set Δ_0^m, Δ_0^p, and the iteration counter $k \leftarrow 0$.
- SEARCH AND POLL STEPS: Perform the search and poll steps (or only part of them) until an improved mesh point x_{k+1} is found on the mesh M_k (3) or until all trial points are visited.
- UPDATES: Update Δ_{k+1}^m, Δ_{k+1}^p, and V_{k+1}.
  Set $k \leftarrow k + 1$ and go back to the search and poll steps.

Figure 2: High-level description of the MADS algorithm.

2.2 Summary of the convergence analysis

The MADS convergence analysis [15, 10] relies on the Clarke calculus [25] for nonsmooth functions. Under mild hypotheses, the algorithm globally converges (i.e. independently of the starting point) to a point $\hat{x}$ satisfying local optimality conditions based on local properties of the functions defining the problem. The basic assumptions are that at least one initial point in X is provided, but not required to be in Ω, and that all iterates belong to some compact set. If nothing about f is known then $\hat{x}$ is the limit of mesh local optimizers on meshes that get infinitely fine. This result is called the *zeroth order result*. If in addition f is Lipschitz near $\hat{x}$ and if the constraint qualification that the hypertangent cone of Ω at $\hat{x}$ is nonempty, then the Clarke generalized directional derivatives of f in all of the directions in the Clarke tangent cone at $\hat{x}$ are nonnegative. A corollary to this result is that if f is strictly differentiable near $\hat{x}$, then $\hat{x}$ is a Clarke KKT stationary point, and in the unconstrained case $\nabla f(\hat{x}) = 0$. Some examples and counter-examples illustrating these results are presented in [7].

In [2], the convergence analysis of MADS is extended to the second order under more hypotheses on the smoothness of the functions defining the problem. In [44], MADS and other directional direct search methods are analyzed for discontinuous functions. Finally, note that analyses have also been developed for the BiMADS and PSD-MADS algorithms [17, 21].

2.3 Main features and extensions of MADS

This section summarizes various features and extensions brought to MADS since its release in 2006. This features have been integrated into NOMAD.

2.3.1 Polling strategies

The NOMAD software implements both the GPS and MADS algorithms. The way of choosing the poll directions is what makes the difference between the two methods. In GPS, the directions D_k must be chosen from the finite set D fixed over all iterations. In MADS, D_k is not restricted to be a subset of D and the normalized directions may be chosen to be asymptotically dense in the unit sphere, allowing a better coverage of the search space.

From a practical point of view, in NOMAD, both GPS and MADS are coded. GPS uses the coordinate directions (called simply the GPS directions), and two families of directions are used for MADS: First the LT-MADS directions, as defined by the original 2006 paper, which are based on the random generation of lower-triangular matrices (thus the name LT), and then the OrthoMADS directions, introduced in 2008 [3], where no more randomness is involved and which are orthogonal (hence their name).

Figure 3 illustrates the differences between the GPS, LT-MADS, and OrthoMADS directions. OrthoMADS is now considered as the default type of directions both in research papers and in NOMAD. This decision is motivated by the fact that OrthoMADS directions have been proved generally more efficient than GPS or LT-MADS in [3] on a set of 45 problems. However, there may be some problems on which LT-MADS or GPS give similar or better results. This is the reason why the three types of directions may be chosen in NOMAD.

2.3.2 Constraints handling

The two sets X and Ω are involved in the definition of the feasible domain. X defines unrelaxable constraints that cannot be violated by any trial point. The c_j functions of the set Ω define the relaxable constraints that can possibly be satisfied only at the final solution $\hat{x}$. For infeasible points, the c_j functions provide a measure of the constraint violations. A third type of constraint is considered, hidden constraints [24], which can be violated even for points of Ω. Such constraints arise when the blackbox function to optimize is a computer code that fails to evaluate at some points.

MADS and NOMAD can deal with these three types of constraints with different strategies. First, the extreme barrier (EB) approach consists in rejecting infeasible points. The original MADS paper of 2006 uses this technique. Other approaches include the filter technique [14] and more recently the progressive barrier (PB) [16]. These two last strategies are not penalty methods and exploit the fact that the constraint viola-

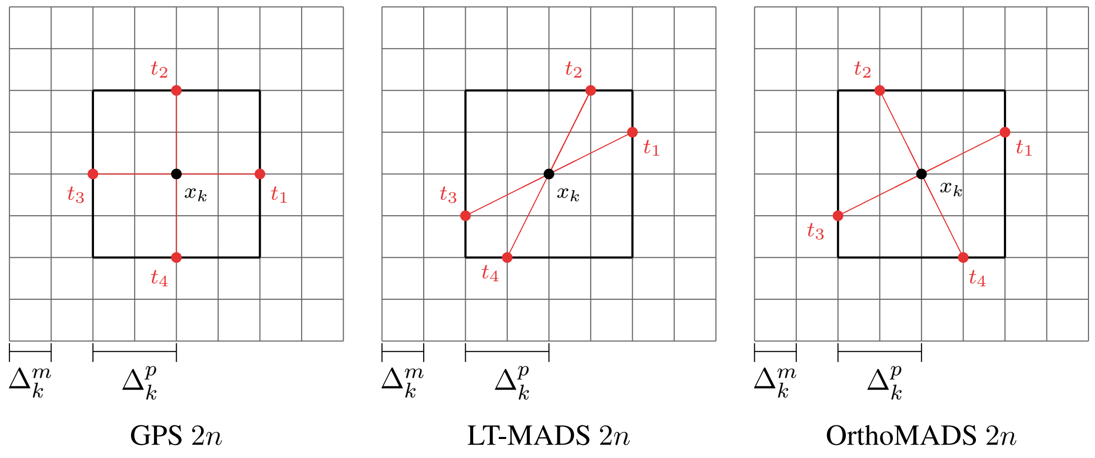

Figure 3: Illustration of the differences between the types of directions in the case $n = 2$. Thin lines represent the mesh of size Δ_k^m and thick lines the points at distance Δ_k^p from x_k in norm L_∞. In the three cases, $\Delta_k^p = 2\Delta_k^m$, and $2n$ directions are considered. GPS directions are orthogonal but correspond always to the coordinate directions. LT-MADS and OrthoMADS directions have greater flexibility: trial points can be anywhere at the intersection of thin and thick lines. Moreover, OrthoMADS directions are orthogonal.

tion of relaxable constraints can be measured. An hybrid method of PB and EB, called Progressive-to-Extreme Barrier (or PEB), is introduced in [18].

These strategies define how the algorithm determines if an evaluation is a success or not. With the EB approach, a success is simply achieved when a new best feasible solution is found. This is more elaborated with the filter and PB approaches, as infeasible points are not ignored, and the reader may consult the cited papers for details.

2.3.3 Surrogates

Surrogate functions are functions that share some similarities with the blackbox functions defining problem (1), but are cheaper to compute. They are useful when the true blackbox functions are costly to evaluate.

Two types of surrogates may be distinguished: Non-adaptive surrogates are defined by the user at the beginning of an optimization and do not evolve during the algorithm execution. Adaptive surrogates, on the contrary, are automatically computed and updated during the execution from past evaluations. Model-based derivative-free methods use models that can be compared to adaptive surrogates (see the description of derivative-free optimization methods in [26]).

NOMAD currently exploits only the non-adaptive surrogates but adaptive surrogates will be studied in future research. Surrogates are used in the poll step: Before the poll trial points are evaluated on the true functions, they are evaluated on the surrogates. This list of points is sorted according to the surrogate values so that the most promising points are evaluated first during the true evaluations. Surrogates are also used in the VNS search described in 2.3.7.

For a more complete review on surrogates, the interested reader may consult [23].

2.3.4 Biobjective optimization

Biobjective optimization is performed with the BiMADS algorithm of [21], consisting of running a series of single-objective MADS runs.

The objective of the method is to construct a list of undominated solutions that gives an approximation of the Pareto front. The single-objective reformulations are not based on weights, but rather on nonsmooth functions for which MADS is well suited. This allows to find Pareto fronts that may be discontinuous or non-convex.

Future research include the handling of more than two objective functions with the algorithm described in [22].

2.3.5 Parallelism

There are several ways of parallelizing the MADS algorithm. The most simple one is to evaluate in parallel the different lists of trials points that can be generated during the algorithm.

This method is called the p-MADS algorithm and is available in the NOMAD package using the Message Passing Interface MPI [40]. In particular, two versions of the p-MADS method may be used: a synchronous and an asynchronous one. The synchronous version consists in evaluating the points in parallel and to wait for all evaluations in progress to terminate an iteration and to decide which new incumbent to consider. The asynchronous version, which is in fact a simplified version of the asynchronous parallel pattern search algorithm APPSPACK [32], allows to terminate an iteration as soon as a new success is achieved. Evaluations in progress are not canceled and if an old evaluation terminates after the incumbent has been changed, and if this evaluation gives a new best point, then the algorithm will backtrack and consider this point.

A more evolved parallel algorithm applied to MADS have been studied in [17]. This method, called PSD-MADS for parallel space decomposition of MADS, allows to treat larger problems than the ones NOMAD usually considers.

The principle of PSD-MADS is to distribute subproblems to the different processes. These subproblems correspond to the original problem with a certain number of fixed variables, resulting in smaller problems that MADS can tackle. A master process decides the groups of variables, and a cache server centralizes the evaluations so that no trial point is evaluated twice.

Another parallel version using MADS has been developed. This version is called COOP-MADS and consists in running several MADS algorithms on the same problem, with the same cache server used in PSD-MADS. The different MADS parallel executions are performed with different seeds so that they have a different behavior.

Inside the NOMAD package, PSD-MADS and COOP-MADS are given as separated programs using the NOMAD library, while p-MADS is directly integrated in the NOMAD code. Numerical results comparing these different parallel versions may be consult at [35]. The results presented in this reference, in addition to other tests, suggest that PSD-MADS is well suited for the larger problems (up to $\simeq$500 variables) and that p-MADS gives in general equivalent results than MADS and much faster, but that COOP-MADS is usually more efficient than p-MADS.

2.3.6 Categorical variables

The MADS algorithm has been developed for continuous variables. The handling of integer or binary variables is easy in practice with the use of minimal mesh sizes of one, and is available in NOMAD.

Categorical variables are variables that can take only a finite and predetermined set of values, without ordering properties. Moreover, for some problems, the values of the categorical variables have an influence on the dimension of the problem. Hence categorical variables cannot be represented by integers and are totally problem-dependent. Extentions to the MADS algorithm have been developed in [1, 4, 5, 12, 33] to treat such problems. The MADS algorithm for categorical variables is included into the NOMAD package and executes the *extended poll* during the poll step of the method. The extended poll consists in polling around points where new values for the categorical variables are tested. In NOMAD, the extended poll may only be performed in library mode where the user defines the neighbors related to the considered problem via C++ virtual functions.

2.3.7 Other features

A generic search strategy based on the VNS metaheuristic (Variable Neighborhood Search) [38] has been integrated into MADS in [9] and is included in NOMAD. This search strategy consists in perturbing the current iterate and to conduct poll-like descents from the perturbed point. This allows to escape from local optima on which the algorithm may be trapped. These VNS descents are performed on surrogates if they are provided.

Finally, support of periodic (or cyclic) variables has been recently added to both the software and to the theory [20]. This may appear for some problem where the functions are periodic with respect to some of the variables. Angles are the most common periodic variables. These variables are marked as *periodic* by the user and NOMAD treats them by mapping their value into the interval defined by their period.

3 Implementation

NOMAD is a stand-alone C++ code. The parallel implementations p-MADS COOP-MADS, and PSD-MADS have been developed with MPI and tested with the major implementations of the library (openMPI, LAM, MPICH and the Microsoft HPC package). The C++ language has been chosen for several reasons. First, in the context of CPU expensive blackbox optimization problems, the MADS algorithm requires no significant computation in the body of NOMAD as most of the computational load is performed by the user-provided functions. These functions, though, can be coded with fast numerical languages such as FORTRAN and their interface with NOMAD is simple and documented in the user guide. In addition, using an object oriented language allows an easy customization of the software due to the definition of various virtual functions. Moreover, NOMAD types have been put inside a `NOMAD` namespace for a better integration with other codes. For this reason, the types described in this section have the format `NOMAD::Type`. Finally, C++ provides exception handling facilities that make it easier to handle errors.

The NOMAD installation produces two executables (scalar and parallel) that take a parameters file as input (batch mode), and two static libraries with which users can link their own code (library mode). The source code may be consulted directly from the files in the package, or consulted via the Doxygen documentation on the website. It is a good additional resource to understand the program design.

The remaining of this section summarizes the NOMAD principles and describes the key points of the software.

3.1 General principle

The principle of NOMAD as a direct search algorithm is given in Figure 4. It consists in generating lists of trial points that are submitted to the blackbox functions to be evaluated. Results of these evaluations are then examined and used to generate new trial points.

One NOMAD iteration is rigorously similar to the algorithm shown in Figure 2 and consists of the three steps: search, poll, and updates.

By default, the search step first executes the *speculative search* consisting of one trial point generated alongside the last successful direction. The speculative search is only used with MADS directions. With GPS directions, this is replaced by sorting the poll trial points relatively to the last success. The user may also activate the Latin-Hypercube (LH) and VNS searches or define a custom search strategy.

The poll step is executed only when no success was obtained in the search step. It first selects a poll center x_k that may be the best feasible point so far or the best infeasible as in [14, 16]. Then the directions D_k are generated to construct the set P_k of poll trial points (details on the directions generation is given in 3.6). Three families

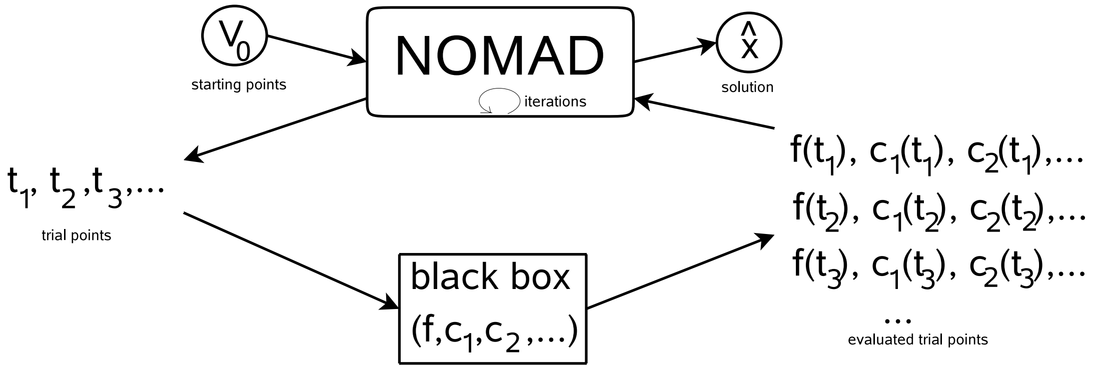

Figure 4: General principle of NOMAD and of any direct search method. The algorithm iteratively constructs lists of trial points that are evaluated by the blackbox. The parallel method p-MADS consists in evaluating theses lists of points in parallel.

of directions may be chosen: GPS, LT-MADS, or OrthoMADS, which is the default. Hybrid combinations of these directions are possible.

The way in which NOMAD evaluates the list of points generated by the search and poll steps is detailed in Section 3.7.

In biobjective optimization, things are quite different. Once NOMAD detects that the problem has two objectives, it launches the BiMADS algorithm.

3.2 Basic data structures

NOMAD is based on two basic data structures to represent reals and vectors. The class of reals is `NOMAD::Double` and is based on C++ double-precision reals. What this class additionally brings is the handling of undefined values and the redefinition of all comparison operators relative to a specific tolerance (set by default to 10^{-13}). The undefined status that reals can possess allows the throwing of exceptions every time an undefined value is used. Vectors are represented by the class `NOMAD::Point` which is a simple array of `NOMAD::Double` objects with some additional operators. The user should inspect the header files `Double.hpp` and `Point.hpp` for details.

Points that are going to be evaluated are referred to as *evaluation points*. The corresponding data structure is `NOMAD::Eval_Point`, a derived class of `NOMAD::Point`. This class allows the evaluation points to store themselves the results of the evaluations. Objects of these class are memorized in the cache described in the next section. They are also used to store the criteria with which trial points are sorted, ranging from surrogate values to user-defined criteria (see 3.7).

Finally, several simple types are represented with enumeration types, bringing more security and comprehension to the code by replacing the use of integers and special nomenclatures. These types are visible in `defines.hpp`.

3.3 The cache

The cache is the central data structure of NOMAD. It corresponds to the set V_k from the mesh definition (3). All points that are been evaluated are stored in the cache which is consulted before each new evaluation to avoid double evaluations. The points are compared using the custom comparison operators included in `NOMAD::Double`, meaning that the real coordinates are compared with a default tolerance of 10^{-13}.

The cache is based on the STL `std::set` data structure which consists of a binary tree with unique entries. No other specialized data structure has been conceived as it was totally non-relevant: For the target class of problems that NOMAD considers, the largest problem have roughly a maximum of 500 variables, and for these extremely large problems, it is expected that no more than 500,000 blackbox evaluations will be made. Based on these extreme values, tests showed that access times in the cache were still extremely fast (for this example, a cumulated time of approximately 8 seconds has been observed for 500,000 consecutive insertions and search operations on a 2003 machine).

The only issue that may be encountered with long NOMAD executions on large problems is the memory occupation of the cache. To avoid such difficulties, NOMAD includes a parameter to limit the size of the cache in memory (`MAX_CACHE_MEMORY`). If this size is reached, the algorithm terminates (and the user may launch a new execution from the last success).

The cache can be saved in a binary file in order for the evaluations to be memorized from a NOMAD run to another. Note that with the largest caches, writing the file can take some time. This is why the parameter `CACHE_SAVE_PERIOD` may be specified to define the period at which the cache is saved, in terms of iteration numbers.

When surrogate functions are used, a second cache is created specially for the surrogate evaluations. For the parallel algorithms COOP-MADS and PSD-MADS, a special parallel version of the cache has been designed, called the cache server. It is in fact a derived class of the class `NOMAD::Cache` which overrides the `find` and `insert` methods. This allows a process to centralize all the evaluations and to communicate cache points to other processes.

3.4 Scaling

Users can scale their variables using the `SCALING` parameter. However, NOMAD has a natural way of automatically dealing with the scaling of bounded variables: For each variable, NOMAD considers one mesh size parameter Δ_k^m, and one poll size parameter Δ_k^p. In other words, in the code, mesh and poll size parameters are vectors represented by `NOMAD::Point` objects.

Recall that the matrix of mesh directions D is the product of matrices G and Z. From a practical point of view, the matrices D, G, and Z are never explicitly expressed in NOMAD, but the vectors Δ_k^m and Δ_k^p that the program considers are in fact the product of the matrix G with the scalars Δ_k^m and Δ_k^p defined by the MADS algorithm. Scaling is performed with initial values for these vectors which are equal by default to 10% of the variable ranges:

$$\Delta_0^m \quad = \Delta_0^p \quad = 0.1(u-l) \quad \text{for all } i = 1, 2, \ldots, n, \tag{6}$$

where $l \in \mathbb{R}^n$ and $u \in \mathbb{R}^n$ are the variable bounds. Parameters allow to change these defaults. A value of 1 is considered for variables with no bounds, but then scaling may be an issue. In that case, users must use the scaling parameter or perform directly their scaling into the blackbox.

3.5 Constraints handling

NOMAD can handle several types of inequality constraints in addition to bounds. Equality constraints are not yet supported though one may replace one equality constraint with two inequalities. This strategy is in general not efficient without a more specialized treatment.

Extreme Barrier (EB) constraints correspond to unrelaxable constraints that must be satisfied by all trial points. Relaxable constraints can also be treated by EB constraints but it will be less efficient than to use the filter or PB approaches. If EB constraints are violated at a given point, this point is simply rejected by NOMAD as if the evaluation failed. From a user point of view, if such constraints are cheap to evaluate, they should be computed first so that the remaining computations are not made in case of violation. When EB constraints are used, the user should provide a starting point at which these constraints are satisfied. If this is not the case, NOMAD will run a first phase in which the objective value is changed to the minimization of the values displayed by the blackbox for these constraints. If the corresponding constraints are unrelaxable constraints without information on the constraint violation, this phase will fail to generate a feasible solution.

Progressive Barrier (PB) constraints or filter constraints can be violated during the execution. For such constraints, of the form $c_j \leq 0$, $j = 1, 2, \ldots, m$, the user must output the c_j functions from its blackbox. For a trial point x, NOMAD will compute a global measure of the constraints violation with the *constraint violation function*

$$h(x) \quad = \| \max\{0, c_j(x)\} \|^2 \tag{7}$$

(a parameter allows to change the default L_2 norm used for the computation of h). For a feasible point x, h is equal to zero, and $h(x) > 0$ means that x violates at least one constraint. The constraint violation function is used by the progressive barrier and the filter strategies to define the notion of dominance of a point and to determine if an infeasible point is good enough to be considered as a success and as a potential future poll center. Through its execution, NOMAD will try to generate trial points with smaller and smaller values of h with the hope that the proposed solution $\hat{x}$ satisfies $h(\hat{x}) = 0$. See [14, 16, 18] for explanations about these strategies, and the `NOMAD::Barrier` class for their implementation. Finally, note that NOMAD can deal with several types of constraints at the same time: For example the user may define EB alongside PB constraints.

3.6 Generation of directions

Directions of D_k are constructed during the poll step. They are used to generate P_k the set of poll trial points, from a poll center x_k and from the mesh size parameter Δ_k^m.

NOMAD considers three families of directions: LT-MADS, OrthoMADS, and GPS. Each of them have options (for example the number of directions), leading to 14 different possibilities which are described in the user guide. The user may choose one or several types of directions with the parameter `DIRECTION_TYPE`, but NOMAD uses OrthoMADS with $2n$ directions as default, as already explained in Section 2.3.1. For integer variables, directions are simply rounded and special directions are used for binary variables.

From a practical point of view, the NOMAD directions already include the mesh size parameter Δ_k^m, reducing the risk of numerical errors. When using the directions to generate the trial points, the directions are then not multiplied with Δ_k^m. If the trial points are generated outside the bounds, NOMAD projects them to the boundary (the parameter `SNAP_TO_BOUNDS` allows to disable that feature). Finally, directions are memorized by the `NOMAD::Eval_Point` objects representing the trial points so that the successful directions can be used for the speculative search.

3.7 Evaluations

Evaluations are supervised by the `NOMAD::Evaluator_Control` class and the `eval_lop()` function (`lop` stands for *list of points*). This class has a link to a `NOMAD::Evaluator` object that may be user-defined in library mode.

When the trial points are generated in the search and poll steps, the function `add_eval_point()` is called to add an evaluation point to the list of points considered later by `eval_lop()`. Several priorities are used to sort this list of points and to decide in which order they will be evaluated. For example, points may be ordered according to surrogate values. If the user defines its own search step, user-defined priorities may also be defined.

The `eval_lop()` function first calls two virtual functions that the user may override: `NOMAD::Eval_Point::set_user_eval_priority()` and `NOMAD::Evaluator::list_of_points_preprocessing()`. They allow the user to preprocess the trial points to be evaluated. This was useful, for example, for a location problem in which the variables represented objects on a map with a lot of infeasible locations [8]. An user with the knowledge of this map can then program a routine to modify the trial points so that they become feasible.

When the `eval_lop()` loops through the list of points to evaluate, it first checks if the points are in the cache. If so, the evaluation point is taken from the cache and no additional evaluation is performed. If the point is not in the cache, the method `eval_x()` of the associated `NOMAD::Evaluator` object is invoked. If this latter has been user-defined in library mode, then the user `eval_x()` function is used. If not, the default `eval_x()` is called which evaluates the functions via system calls.

The evaluations may be opportunistic or not, i.e. they can be interrupted after the first success or entirely performed. In this case, the previous ordering of the trial points was unnecessary. The opportunistic strategy applies to both the poll and search steps, but it does not apply at the iteration level: If the search step is successful, the poll will never be performed at the same iteration.

With the p-MADS parallel version of the method, it is the `eval_lop()` function which is parallelized.

3.8 Additional functionalities

This section about the NOMAD implementation concludes by exposing a list of functionalities that are not expressed elsewhere in this paper. The objective here is to indicate some practical aspects of NOMAD.

- Several starting points: In the MADS algorithm, the set V_0 may contain several elements. With NOMAD, the user may also specify not just one, but several starting points that are not necessarily feasible. The algorithm will start from the most promising one. A cache file can also be specified as a starting point.

- Types of variables: NOMAD treats problems with continuous, binary, integer, and categorical variables. Variables can be bounded or free, and defined to be periodic.

- Fixed variables: The user may decide to fix the value of some variables. This feature is useful for post-analysis tests and is used by the COOP-MADS and PSD-MADS programs.

- Groups of variables: The user can define groups of variables, which are kept constant during the run. The directions are generated according to these groups, meaning that, from the current iterate, no trial point can be generated with changes in two different groups at the same time. One usage example applies to location problems where the variables correspond to objects positions. It is possible to define one group for each object. Another example is to create groups including only the important variables, if the user possesses a strategy to determine them.

- User search: NOMAD allows the user to define its own search strategy. This is motivated by the fact that often blackbox designers know some specific features of their problem and thus may exploit this knowledge to design a tailored search method. To be used with NOMAD, such a strategy must be coded and linked to the NOMAD library. The user search strategy will then be called at each search step. An example is available in the NOMAD package, corresponding to a case studied in [20].

- Seeds: If LT-MADS directions or LH search are used, the algorithm becomes stochastic, since it uses randomness. It is possible to change the course of a run by selecting a random seed. Otherwise, without LH search and with GPS or OrthoMADS directions, the algorithm is deterministic. However, it is possible to have different executions and results by setting the OrthoMADS Halton seed. Changing the seeds is illustrated in the code of COOP-MADS.

4 Discussion

This paper gives an overview of the NOMAD software for blackbox optimization with general nonlinear constraints. Implementation details as well as the links with the MADS algorithm have been described. NOMAD has various features that no other software possesses all together. The most important of these features are biobjective optimization, evolved and various strategies to handle inequality blackbox constraints, handling of continuous, integer, binary and categorical variables, and parallelism. NOMAD is freely available as a stand-alone C++ code, with a complete documentation, and may be easily integrated into other applications. NOMAD is easy to use as all the algorithmic parameters have defaults. In order to verify the NOMAD efficiency, references [3, 18, 19, 28, 39] use the version 3 of NOMAD to generate all their numerical results on various blackbox problems.

Future developments include the use of simplex gradients [27, 29] and multi-objective optimization with more than two objectives [22]. Adaptive surrogates will also be introduced according to the surrogate management framework of [23].

Acknowledgments

I would like to thank Charles Audet and John Dennis for their support and helpful remarks. I also wish to thank the anonymous referee who helped a lot in improving both the present paper and the software.

References

[1] M. A. Abramson. Mixed variable optimization of a load-bearing thermal insulation system using a filter pattern search algorithm. *Optimization and Engineering*, 5(2):157–177, 2004.

[2] M. A. Abramson and C. Audet. Convergence of mesh adaptive direct search to second-order stationary points. *SIAM Journal on Optimization*, 17(2):606–619, 2006.

[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] 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, January 2009.

[5] M.A. Abramson, C. Audet, and J. E. Dennis, Jr. Filter pattern search algorithms for mixed variable constrained optimization problems. *Pacific Journal on Optimization*, 3(3):477–500, 2007.

[6] H. Aoudjit. *Planification de la maintenance d'un parc de turbines-alternateurs par programmation mathématique*. PhD thesis, Ecole Polytechnique de Montréal, 2010.

[7] C. Audet. Convergence Results for Pattern Search Algorithms are Tight. *Optimization and Engineering*, 5(2):101–122, 2004.

[8] C. Audet, S. Alarie, S. Le Digabel, Q. Lequy, M. Sylla, and O. Marcotte. Localisation de stations de mesure automatisée du couvert nival. Technical Report CRM-3277, Centre de Recherches Mathématiques, 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, June 2008.

[10] C. Audet, A. L. Custódio, and J. E. Dennis, Jr. Erratum: Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 18(4):1501–1503, 2008.

[11] C. Audet, C.-K. Dang, and D. Orban. Algorithmic parameter optimization of the dfo method with the opal framework. Technical Report G-2010-02, Les cahiers du GERAD, 2010. To appear in Software Automatic Tuning, Springer.

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

[13] C. Audet and J. E. Dennis, Jr. Analysis of generalized pattern searches. *SIAM Journal on Optimization*, 13(3):889–903, 2003.

[14] C. Audet and J. E. Dennis, Jr. A pattern Search Filter Method for Nonlinear Programming without Derivatives. *SIAM Journal on Optimization*, 14(4):980–1010, 2004.

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

[16] C. Audet and J. E. Dennis, Jr. A Progressive Barrier for Derivative-Free Nonlinear Programming. *SIAM Journal on Optimization*, 20(4):445–472, 2009.

[17] 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.

[18] C. Audet, J. E. Dennis, Jr., and S. Le Digabel. Globalization strategies for mesh adaptive direct search. *Computational Optimization and Applications*, 46(2):193–215, June 2010.

[19] C. Audet, X. Fournier, P. Hansen, S. Perron, and F. Messine. A note on diameters of point sets. Technical Report Les Cahiers du GERAD G-2009-85, Les cahiers du GERAD, 2009. To appear in Optimization Letters.

[20] C. Audet and S. Le Digabel. The mesh adaptive direct search algorithm for periodic variables. Technical Report G-2009-23, Les cahiers du GERAD, 2009.

[21] 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.

[22] 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.

[23] A. J. Booker, J. E. Dennis, Jr., P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural Optimization*, 17(1):1–13, February 1999.

[24] T. D. Choi and C. T. Kelley. Superlinear convergence and implicit filtering. *SIAM Journal on Optimization*, 10(4):1149–1162, 2000.

[25] F. H. Clarke. *Optimization and Nonsmooth Analysis*. Wiley, New York, 1983. Reissued in 1990 by SIAM Publications, Philadelphia, as Vol. 5 in the series Classics in Applied Mathematics.

[26] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization*. MPS/SIAM Book Series on Optimization. SIAM, Philadelphia, 2009.

[27] A. L. Custódio, J. E. Dennis, Jr., and L. N. Vicente. Using simplex gradients of nonsmooth functions in direct search methods. *IMA Journal of Numerical Analysis*, 28(4):770–784, 2008.

[28] A. L. Custódio, J. F. A. Madeira, A. I. F. Vaz, and L. N. Vicente. Direct multisearch for multiobjective optimization. Technical Report Preprint 10-18, Dept. of Mathematics, Univ. Coimbra, May 2010. Paper available at http://www.mat.uc.pt/~lnv/papers/dms.pdf.

[29] A. L. Custódio and L. N. Vicente. Using sampling and simplex derivatives in pattern search methods. *SIAM Journal on Optimization*, 18(2):537–555, May 2007.

[30] W. C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1(1):1–17, February 1991.

[31] K.R. Fowler, J.P. Reese, C.E. Kees, J.E. Dennis Jr., C.T. Kelley, C.T. Miller, C. Audet, A.J. Booker, G. Couture, R.W. Darwin, M.W. Farthing, D.E. Finkel, J.M. Gablonsky, G. Gray, and T.G. Kolda. Comparison of derivative-free optimization methods for groundwater supply and hydraulic capture community problems. *Advances in Water Resources*, 31(5):743–757, May 2008.

[32] G. A. Gray and T. G. Kolda. Algorithm 856: APPSPACK 4.0: Asynchronous parallel pattern search for derivative-free optimization. *ACM Transactions on Mathematical Software*, 32(3):485–507, September 2006.

[33] M. Kokkolaras, C. Audet, and J. E. Dennis, Jr. Mixed variable optimization of the number and composition of heat intercepts in a thermal insulation system. *Optimization and Engineering*, 2(1):5–29, 2001.

[34] S. Le Digabel. NOMAD user guide. Technical Report G-2009-37, Les cahiers du GERAD, 2009.

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

[36] Inc. MathWorks. MATLAB GADS toolbox. http://www.mathworks.com/products/gads/, 2005.

[37] H. D. Mittelmann. Decision tree for optimization software. http://plato.asu.edu/guide.html, 2007.

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

[39] L.M. Rios and N.V. Sahinidis. Derivative-free optimization: A review of algorithms and comparison of software implementations. Technical report, Carnegie Mellon University, May 2010. Paper available at http://thales.cheme.cmu.edu/dfo.

[40] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI: The Complete Reference*. The MIT Press, Cambridge, Massachusetts, 1995.

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

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

[43] D. van Heesch. Doxygen manual. http://www.doxygen.org/manual.html, 1997.

[44] L. N. Vicente and A. L. Custódio. Analysis of direct searches for discontinuous functions. Technical report, Dept. of Mathematics, Univ. Coimbra, May 2010. Paper available at http://www.mat.uc.pt/~lnv/papers/discon.pdf.