VXQR: Derivative-free unconstrained optimization based on QR factorizations

Arnold Neumaier, Hannes Fendl,
Harald Schilly, Thomas Leitner

Fakultät für Mathematik, Universität Wien
Nordbergstr. 15, A-1090 Wien, Austria
email: Arnold.Neumaier@univie.ac.at
WWW: http://www.mat.univie.ac.at/~neum/

June 20, 2010

Abstract. This paper presents basic features of a new family of algorithms for unconstrained derivative-free optimization, based on line searches along directions generated from QR factorizations of past direction matrices. Emphasis is on fast descent with a low number of function values, so that the algorithm can be used for fairly expensive functions. The theoretical total time overhead needed per function evaluation is of order $O(n^2)$, where n is the problem dimension, but the observed overhead is much smaller.

Numerical results are given for a particular algorithm VXQR1 from this family, implemented in Matlab, and evaluated on the scalability test set of Herrera et al. [14] for problems in dimensions $n \in \{50, 100, 200, 500, 1000\}$.

Performance depends a lot on the graph $\{(t, f(x+th)) \mid t \in [0,1]\}$ of the function along line segments. The algorithm is typically very fast on smooth problems with not too rugged graphs, and on problems with a roughly separable structure. It typically performs poorly on problems where the graph along many directions is highly multimodal without pronounced overall slope (e.g., for smooth functions with superimposed oscillations of significant size), where the graphs along many directions are piecewise constant (e.g., for problems minimizing a maximum norm), or where the function overflows on the major part of the search region and no starting point with finite function value is known.

Keywords: Derivative-free optimization, black box optimization, scalability, high-dimensional, global optimization, line search, expensive objective function

2010 MSC Classification: primary 90C56

1 Introduction

Background. In derivative-free optimization (also known as black-box optimization), the goal is to optimise a function defined on a subset of $\mathbb{R}^n$ for which derivative information is neither symbolically available nor numerically computable, and bounds on Lipschitz constants are not known. Since the beginning of the field in 1961 with the algorithm by Hooke & Jeeves [15], numerous algorithms and software implementations have been proposed. Recently the interest in derivative-free optimization grew rapidly, motivated by practical applications in physics, chemistry, biology, engineering design, medical science, finance, and operations research.

Many approaches and algorithms exist already, and the field undergoes rapid growth. Currently, there are various approaches to derivative-free and black-box optimization coming from three different research communities that in the past interacted only loosely. These approaches have led to three main classes of algorithms:

Direct methods are deterministic but make no model assumptions. They include the simplex method by Nelder & Mead [24] (cf. Kelley [20]), pattern search (e.g., Torczon [33]), and generating set search (e.g., Kolda et al. [21]).

Indirect or model-based methods are also deterministic, and include methods using quadratic model fits (e.g., SNOBFIT by Huyer & Neumaier [17]), trust-region methods based on model updates (e.g., NEWUOA by Powell [28]), other response surface methods (e.g. DACE by Sacks et al. [31]) and surrogate function methods (e.g., EGO by Jones et al. [18]), Lipschitz constant based method (e.g., DIRECT by Jones et al. [19]), implicit filtering (e.g., Gilmore & Kelley [10], and *multilevel coordinate search* (MCS) by Huyer & Neumaier [16]. Created in 1999 at the University of Vienna, it is the only global optimization code (added in 2009; see [23]) in the well-known NAG library of routines for the solution of numerical and statistical problems.

Stochastic methods use random choices in their strategy and include, e.g., simulated annealing (Van Laarhoven & Aarts et al. [35], genetic algorithms (e.g., real-coded CHC by Eshelman & Schaffer [8]) and their modern evolutionary improvements (e.g., DE by Storn & Price [32] and (G_)CMA-ES by Auger & Hansen [2, 12]), stochastic clustering algorithms (e.g., GLOBAL by Csendes et al. [5]), particle swarm methods (e.g., Eberhart & Kennedy [7]), ant colony optimization (e.g., Dorigo & Stützle [6]), tabu search (e.g., Glover [11], and hit-and-run algorithms (e.g., Bélisle [3]). There are also various hybrid methods, e.g. between direct search and particle swarms (Fan & Zahara [9]).

For details on black box optimization methods and further references, see the book by Conn et al. [4] for deterministic methods and for stochastic methods the books by Törn & Žilinskas [34], Ashlock [1] or Michalewicz & Fogel [22]; see also the thesis by Rios [29].

Recent benchmarking of a large array of black box algorithms for unconstrained optimization has been done by two independent teams. The 2009 PhD thesis by Rios [29], presents a review and systematic comparison of 23 derivative-free algorithms using a test set of 505 problems in dimensions ranging from 1 to 300 (see also Rios & Sahinidis [30]). The algorithms were tested under the same conditions and ranked under several criteria, including their ability to find global solutions to nonconvex problems. It was found that the ability of all the solvers tested to obtain good solutions diminished with increasing problem size. For the problems used in this study, the algorithms LGO (Pintér [27]) and MCS were on average much better than other derivative-free solvers in terms of solution quality within 2500 function evaluations, but LGO was significantly better on the high-dimensional problems of the test set when the budget (the limit on the number of function evaluations allowed) allowed more than 1000 function evaluations. The solver NEWUOA was found best when good startings points were already available.

A second battery of tests of derivative-free optimization algorithms primarily aimed at high accuracy using a huge number of function evaluations. It was performed by Hansen et al. [13] and involved 31 algorithms on a testbed of functions in dimensions up to 40. MCS was ranked together with NEWUOA and GLOBAL best for a function evaluation budget of up to $500n$ function values, but was no longer competitive when the budget was significantly larger, where variants of CMA-ES performed best. (LGO was not tested in this setting, while GLOBAL was not present in the other study mentioned.)

Goals of the paper. The preceding results both confirmed the advantages of MCS and pointed to its limitations. This motivated our search for an algorithm that preserves the advantages of MCS in case of a low budget or low dimensions, while performing better in case of a generous budget and in high dimensions. The main features of MCS involved a very successful initialization strategy, followed by a global multilevel search and local searches building quadratic models. The relative poor performance of MCS when a generous budget was available can be traced to its deterministic nature, which does not allow MCS to recover from getting stuck when the multilevel search happened to rate the region containing the global minimum as too unpromising for further search. The main reason for performing poorly in high dimensions is that the stage where local quadratic models are built is never reached within the budget limit: Even with a very generous budget, each model requires $O(n^2)$ function values, and generally a number of quadratic models must be built to reach high accuracy.

We therefore aimed for creating a stochastic algorithm that uses the ideas leading to the initialization strategy of MCS and the idea of quadratic model building – the two ingredients that made MCS superior in the cases it performed best. The initialization strategy developed into the scout phase of our new algorithm, the quadratic models are applied in the new subspace phase, in adaptively constructed subspaces of low dimension only, and the deterministic multilevel strategy was replaced by stochastic techniques.

Section 2 gives a formal description of the problem solved by VXQR1, and describes the techniques used in VXQR1. Section 3 explains the testing environment used, and Section 4 gives our test results. Section 5 compares our results with those of three reference solvers, taken from HERRERA et al. [14]. Section 6 provides some conclusions. Two appendices give progress figures and a comparative of median results for VXQR1 and the reference solvers from [14].

2 A family of algorithms

We consider the unconstrained optimization problem of minimizing a real-valued function f defined on a subset of $\mathbb{R}^n$ by an oracle that returns for a given $x \in \mathbb{R}^n$ the function value $f(x)$ if it is defined, and otherwise one of the surrogate values `inf` or `NaN`. In addition, scaling information is assumed to be available in the form of a search box

$$\mathbf{x} = [\underline{x}, \overline{x}] = \{x \in \mathbb{R}^n \mid \underline{x} \le x \le \overline{x}\}.$$

The expectation (which may or may not be true) is that the minimizer of interest lies in the interior of the search box; but the search may lead out of the box. The best point found is therefore not guaranteed to lie in the box.

The algorithms to be described are designed for the case when a call to the oracle is fairly expensive, so that one wants to keep the number of function evaluations needed as small as possible, aiming for a rapid decrease of the objective function rather than for necessarily reaching the global minimum. Such a fast decrease is achievable only if the dimension of the problem is small, or if the function to be optimized has an appropriate structure that can be exploited by the algorithms. In particular, our algorithms are based on the expectation that either
• the objective function resembles a separable function, so that the variation of single coordinates is a reasonable option, or
• that the function is smooth (twice continuously differentiable) and not strongly oscillating along typical rays. This enables one to make fast progress using line searches, a basic tool from traditional deterministic optimization algorithms.
However, we also exploit the robustness that can be gained from a stochastic approach, and thus combine deterministic and stochastic features in a novel way. Thus none of the structural expectations is essential for the algorithm to work, although the performance of problems not matching the stated expectations may be erratic or poor.

VXQR stands for *valley exploration based on QR factorizations*, emphasizing the linear algebra tool (QR factorization) that dominates the part of the execution cost of the algorithms that is independent of the cost of the function evaluation routine. The VXQR class of algorithms we consider proceed in several phases. After the initial scaling phase 1, scout phase 2 and subspace phase 3 alternate (according to a tunable strategy) until a stopping criterion is reached. All phases crucially depend on well-implemented line searches; it also matters for performance which kind of line search is employed at which stage.

VXQR1 is a particular realization of this general scheme, chosen in a trial-and-error fashion through experimenting with various strategies and tuning parameters. For our experiments, we used random test problems of various forms and dimensions to check the quality of particular algorithms, with the number of function values bounded by 2500 or 25000. The Matlab code of VXQR1 is available online [26]. The details of the VXQR1 implementation and their mathematical justification will be discussed elsewhere.

Line searches. Line searches are done by function evaluation from the best point along a specified direction. This is the place where the algorithm expects that the objective function is smooth; performance is degraded if this is not the case. We use two versions of the line search: a global one and a local one. A **global line search** evaluates the function at a (tunable) number of points uniformly distributed on a random line segment along this direction containing the best point. This is followed by safeguarded parabolic interpolation steps according to standard formulas (NEUMAIER [25]) at the local minima of the grid function so determined, and possibly by piecewise-linear interpolation steps, to approximately locate local minima along the direction searched. A **local line search** evaluates the function at a random point along the search direction, then reflects the worse point at the best point to get a third point on the line; this is repeated as long as the function value improves. We then perform a single safeguarded parabolic interpolation. If this did not lead to an improvement, a few further points on the line are evaluated according to heuristic criteria.

Because all line searches start from the best point available at the time, the VXQR algorithms have a greedy tendency. This makes them efficient for a low number of function values, but puts them at a disadvantage for highly oscillating functions where the greedy strategy often confines progress to a small neighborhood of the best point, with escape to a different valley often being the result of pure chance rather than strategy.

Phase 1 (scaling phase): Search for a well-scaled initial point x with finite $f(x)$. This is simply done by a uniform random search in the specified search box, followed by a line search in the direction of the point in the search region with the absolutely smallest components to get an initial point with an appropriate order of magnitude.

Phase 2 (scout phase): Search for a direction of good expected progress. This is done by a sequence of line searches from the best point, either in coordinate directions (to be efficient for approximately separable problems), or (to be efficient for nonseparable smooth problems) in a direction chosen from an orthonormal basis that adapts itself during the course of the algorithm. The orthonormal basis is created at the beginning of each scout phase by taking the columns of the orthogonal factor Q of a QR factorization of the matrix formed by the differences of the best point from the results of the scout line searches of the previous scout phase, but chosen randomly before the first scout phase.

Phase 3 (subspace phase): The direction generated by the scout phase is used to extend a low-dimensional search subspace until a saturation dimension (in VXQR1, this dimension is 10 when $n > 20$) is reached; in this case, the scout direction replaces an old subspace direction. In the new subspace, a local quadratic model is created and minimized, subject to some safeguards. Then a line search is performed from the best point found so far to the model minimizer.

The work outside the function evaluations is dominated by the cost of the QR factorizations, the only expensive step in the whole procedure. Since a QR factorization needs $O(n^3)$ arithmetic operations, and since after each QR factorization the scout search takes $O(n)$ but at least $3n$ function evaluations, the total time overhead needed per function evaluation is of order $O(n^2)$. But the observed overhead is much smaller; see Section 4.

3 The testing environment

VXQR1 is primarily designed for giving good results in high dimensions with a very small number nf of function evaluations; it is tuned for $nf = 2500$ and $nf = 25000$. However, to fit into the scope of the present special issue, we test VXQR1 on the problems from the scalability test set of Herrera et al. [14] in high dimension $n \in \{50, 100, 200, 500, 1000\}$, using an upper bound of $5000n$ function evaluations.

For the comfortable development and testing of VXQR variants in high dimensions, we created in Python a parallel solving environment PARAEXEC that allows us to solve on the otherwise unused processors of the workstations of our department many test problems many times with different algorithmic variants, to collect the results in a single place for further inspection and analysis. This means that our tests are done on computers with different performance characteristics. Times in seconds (in the progress figures) are always averages over problem solution times, without taking into account the computer on which the problem was solved, and hence can be taken to be those of an average computer accessible by PARAEXEC at the time the corresponding experiment was performed.

For the present paper, we performed large-scale experiments on the problems from the SCAL benchmark. The SCAL benchmark consists of 19 functions F1–F19*, defined in [14] by explicit dimension-dependent expressions f_k ($k = 1 : 19^*$) in an arbitrary dimension n.

The SCAL benchmark functions F1–F11 are traditional test functions shifted so that their global minimum is attained at a particular point specified by the benchmark for each of these six functions. In particular, F1 is the function defined by

$$f_1(x) := \sum_{i=1}^{n} (x_i - s_i)^2, \tag{1}$$

where s is fixed. F12-F19* are artificial hybrid functions created from F1-F11.

In interpreting the results of a solver obtained on the SCAL benchmark, we shall consider a result to be of **high accuracy** if the difference to the global minimum value is $< 10^{-10}$, of **medium accuracy** if the difference to the global minimum value is $\geq 10^{-10}$ but $< 10^{-2}$, and of **low accuracy** if the difference to the global minimum value is $\geq 10^{-2}$. On the SCAL benchmark, low accuracy usually means that an algorithm got stuck far from the global minimum position, while high accuracy usually means that the global minimum position (and not only the optimal function value) was accurately located. Medium accuracy usually means that there are several nearly global local minima, and that the algorithm found one such non-global minimum only.

Tuning parameters. Many variations are possible both in the strategy and in the details, so there are many tuning parameters whose choice affects the behavior of a VXQR algorithm. However, VXQR1 has a fixed choice of strategy and parameters; so the user need not spend any time on tuning. The strategy and the tuning parameters for VXQR1 were selected to work well for the more realistic limit of 2500 or 25000 function evaluations, since this is already quite generous for many practical optimization problems where a function evaluation takes minutes. In this range, our algorithm should beat most current alternatives with respect to the function values reached, though it rarely comes close to the global minimum with such a tight budget.

With the much larger budget of $5000n$ function evaluations, VXQR1 reaches the global minimizer for many problems that respect the structural assumptions on which the algorithm is based. But it does not make enough progress for many problems that are nonsmooth or highly oscillatory. This can be seen on the results presented below for the test functions from the benchmark "Scalability of Evolutionary Algorithms and other Metaheuristics for Large Scale Continuous Optimization Problems" (referred to in the following as the SCAL benchmark), defined by Herrera et al. [14]. It consists of 19 functions with different structural characteristics, parameterized by the dimension n.

4 Test results

We used VXQR1 to solve all (modified) SCAL benchmark problems 25 times for each dimension $n \in \{50, 100, 200, 500, 1000\}$. Selected results for the individual runs are visualized in the figures in Appendix A. (Conforming to the requested protocol, values below 10^{-14} were treated as zero. In cases where VXQR1 stopped early due to our setting of a target function value, the best function value was assumed not to improve later.)

Good cases. Within the allotted budget of $5000n$ function evaluations, VXQR1 solved at least half of the 25 solution calls to high accuracy 7 problems F1, F4–F8 and F10 in dimensions $n \in \{50, 100, 200, 500, 1000\}$, and F19* in dimension $n = 50$. As expected, this includes the smooth problems without large oscillations, the separable problems, and a few others. An exception was the smooth function F3 (extended Rosenbrock), which performed reasonably in the *best* runs in all dimensions except $n = 1000$, but not frequently enough. We therefore ran F3 again with $50000n$ function values to check the scaling behavior in this case. It turned out that VXQR1 reached for all n considered in at least half the cases the global minimum within $40n^2 + 250n$ function evaluations with an accuracy of better than 10^{-6}. (One would expect the need for $O(n^2)$ function values for any method applied to general nonseparable functions, since not even a general convex quadratic function is determined by fewer than $(n + 1)(n + 2)/2$ function values. For large n, a budget of $O(n)$ function values is adequate only if the function has special structure that can be exploited by the solver to find the solution quickly.)

The following table exhibits the number of function values sufficient to reach in at least half the cases the global minimum to an accuracy close to the limit accuracy, but at least 10^{-6} and thus gives an idea of how VXQR1 scales with the dimension when it is successful. (In view of the limited numerical evidence we did not fit polynomials to the data but give simple upper bounds for the median function values, found by visual comparisons of appropriate plots.)

test function	F1	F3	F4	F5	F6	F7	F8	F10
function values	$10n$	$40n^2 + 250n$	$2000n$	$150n$	$400n$	$300n$	$1200n$	$800n$

Poor cases. On the remaining 11 problems F2, F9, F11–F19*, the performance of VXQR1 was fairly poor in all dimensions $n \in \{50, 100, 200, 500, 1000\}$.

On the eight highly oscillatory problems F9, F11–F14, and F16*–F18*, the greedy strategy of VXQR1 forced the progress path to take numerous tiny turns, resulting in slow but steady progress, but far too slow to come close to the global minimizer within the budget available.

On the problem F2, which required to minimize

$$f_2(x) := \|x - z'\|_\infty = \max_{i=1:n} |x_i - z'_i|, \tag{2}$$

many line searches were completely inefficient since they searched for improvements on a constant piece. Attempts with the p-norm in place of the maximum norm – which corresponds to $p = \infty$ – showed that success could be achieved for increasing p up to $p = 20$, with deteriorating tendency.

The hybrid functions F15 and F19* (created from F10 and F7) were solved well in dimensions $n = 50$ and $n = 100$ but poorly in higher dimensions. Since the nonsmoothness of F7 combines with the nonseparability of F10, the assumptions made in developing VXQR1 are no longer satisfied well enough to ensure scalability.

Running times. The evaluation of each test function takes $O(n)$ operations, and the number of function values was bounded by $5000n$, leading to an expected running time of $O(n^2)$ for the function evaluation part. VXQR1 uses approximately $O(n^2)$ linear algebra operation per function value, suggesting an expected running time of $O(n^3)$. However, since we didn't optimize the administrative part of our programs, and also record in our Matlab testing environment auxiliary statistical information with each function call, the dominant time consumption in the range of dimensions considered was the overhead in calling a function value. Therefore, the total observed run-time was essentially linear in n. Indeed, inspection of the actual times showed that, in all test problems and independent of the dimension, the total running time of the 25 runs divided by the total number of function values computed was always between 0.7ms and 4ms. Of course, different implementations or different test functions may give different running times, and the higher order effects might then be visible.

5 Comparisons

The organizers of the SCAL contest provided results on the average, minimum, maximum, and median function value gaps reached by the three reference algorithms DE, CHC, and G_CMA-ES after $5000n$ function evaluations on all SCAL problems in the dimensions $n \in \{50, 100, 200, 500, 1000\}$ (except for G_CMA-ES and $n = 1000$), each repeated 25 times. Appendix B contains the final median values for DE, CHC, and G_CMA-ES from [14] and those for VXQR1 from our own runs.

In view of the severe heterogeneity of the results, a detailed statistical comparative analysis seems out of place. However, to satisfy the formal requirements for the special issue, we provide a web supplement [26] containing details about the performance of each individual run (summarized in progress figures and tables for minimum, maximum, and mean performance), together with the results for some statistical tests comparing our results with the results of the reference solvers.

We note that the worst case and best case cannot be reproduced with high confidence under repetition of a statistical simulation study. Averages are not very meaningful for numbers ranging over many magnitudes; moreover, these averages are large already when a single run got stuck in a nonglobal local minimum, hence are not representative for the average behavior of an algorithm. We shall therefore concentrate on the median results (achieved in half of the runs).

The results for CHC are almost always inferior to those of either DE or G_CMA-ES, so that we shall compare our results only to DE and G_CMA-ES. We classify problems according to the behavior of these two algorithms.

Two problems, namely F1 and F5, may be classified as easy. They are solved both by DE and G_CMA-ES to high accuracy in the majority of cases, and also VXQR1 solves them to high accuracy. Remarkably, it does so for F1 with less than $10n$ function values, and for F5 with less than $150n$ function values.

Nine problems, namely F2, F4, F6–F8, F10, F12, F15, and F19*, may be classified as accurately solvable. In the majority of cases, they are solved by either DE and G_CMA-ES (but not by both) to high accuracy. Here VXQR1 solves to high accuracy five of these, namely F4, F6–F8, F10. For F15 and F19* (both solved by DE), VXQR1 also ranks second, but achieves only medium accuracy (except for F15 in dimension $n \geq 500$, where the accuracy is low). For F2 (solved by G_CMA-ES) and F12 (solved by DE), VXQR1 achieves only low accuracy, but ranks second except for F2 with $n = 50$.

Four problems of an oscillating nature, namely F11, F14, F16*, and F18*, may be classified as hard but solvable. In the majority of cases, they are solved to medium accuracy (median between 2 and 8 digits) by DE, while G_CMA-ES and VXQR1 only get low accuracy (which therefore should be counted as failure). The VXQR1 results are almost always significantly better than those of G_CMA-ES, but much worse than those of DE.

The remaining four problems, F3, F9, F13, and F17* should be classified as hard problems. Both DE nor G_CMA-ES solves them to low accuracy only (except that G_CMA-ES solves F3 for $n = 50$ and in a few trials for $n \in \{100, 200\}$). VXQR1 is there generally best (for F17* only second but competitive), but the accuracy is also low (again except for F3 in the smaller dimensions).

Ranking DE, G_CMA-ES, and VXQR1 simultaneously, we find that, in the more demanding dimensions $n \geq 100$, both DE and VXQR1 (and sometimes G_CMA-ES) are of high accuracy for F1, F5–F7, and F10. From the other functions, DE is best for the oscillating functions F11, F12, F14-F19*, G_CMA-ES for the minimax function F2, and VXQR1 for the smooth or neamrly separable functions F3, F4, F7–F9, and F13.

6 Conclusion

We introduced a number of new techniques for black box optimization: Local and global line searches, orthogonal search in evolving orthonormal frames, and local quadratic models in adaptively constructed affine subspaces. We tuned and tested a particular strategy called VXQR1 involving these techniques, and found that, even in the highest dimension tested ($n = 1000$), it works quite well for a number of nontrivial test functions.

Performance depends a lot on the graph $\{(t, f(x + th)) \mid t \in [0, 1]\}$ of the function along line segments. The algorithm is typically very fast on smooth problems with not too rugged graphs, and on problems with a roughly separable structure. It typically performs poorly on problems where the graph along many directions is highly multimodal without pronounced overall slope (e.g., for smooth functions with superimposed oscillations of significant size), where the graphs along many directions are piecewise constant (e.g., for problems minimizing a maximum norm), or where the function overflows on the major part of the search region and no starting point with finite function value is known.

Our tentative conclusion (as far as the limited testing discussed here allows one) is that VXQR1 compares well with the reference solvers, though not uniformly over all problem types. For best results, it seems that VXQR1 should be used on smooth problems and on nearly separable problems, DE on problems with large oscillations, and G_CMA-ES on problems that minimize the maximum of many functions.

7 Appendices

The appendices contain figures and tables recording more details about the behavior of VXQR1 on the test problems. Appendix A displays for dimensions $n = 100$ and $n = 1000$ the history of best function values for the test problems that were solved successfully for $n = 100$. Results for the other test problems and for all dimensions tested can be found in the web supplement [26].

Appendix B gives median function values after $5000n$ function values together with the corresponding results for the reference algorithms DE, CHC, and (for $n \neq 1000$) G_CMA-ES, taken from Herrera et al. [14]. Corresponding tables for minimum, maximum, and mean function values after $5000n$ function values can also be found in the web supplement [26].

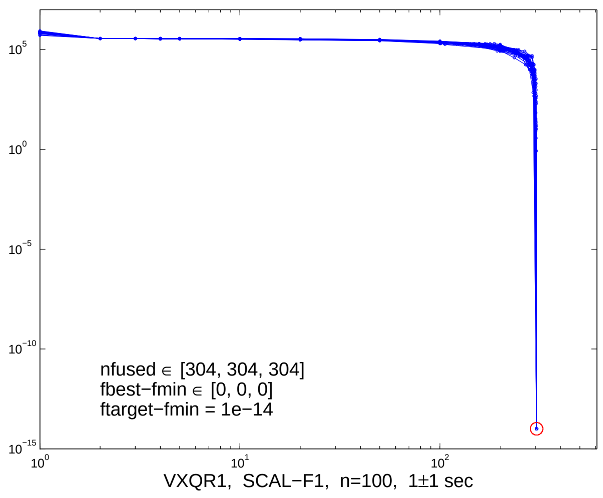
Appendix A: Progress figures

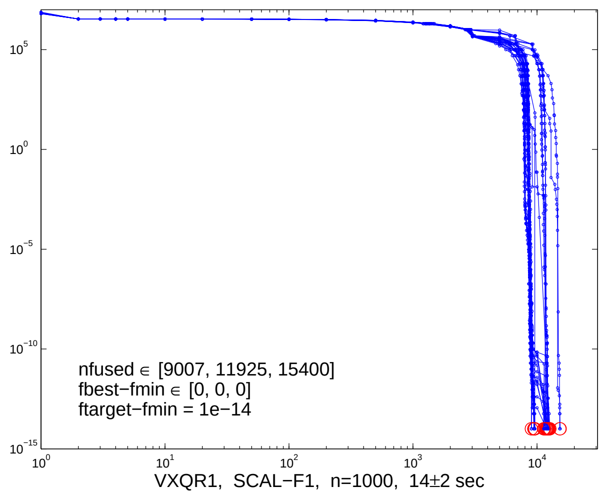
The following figures give a dynamical view of the difference between the best function value in each run and the global minimum, as a function of the number of function values. The final value of each run, rounded upwards towards the target (if needed) is indicated by a circled dot. A nearly vertical dropdown of a curve indicates that the close neighborhood of the global minimum was reached. To avoid the figure scaling to be determined by huge initial function values (sometimes of the order of $>> 10^{200}$), the cut-off value 10^{14} is used in the upper part of the figures.

The captions describe the function used, the dimension, and the median time needed to solve the resulting problem – rounded upwards to full seconds, ± the median of the absolute deviation. (Since some machines in our parallel environment were much slower than the others, mean values and standard deviations were not meaningful.)

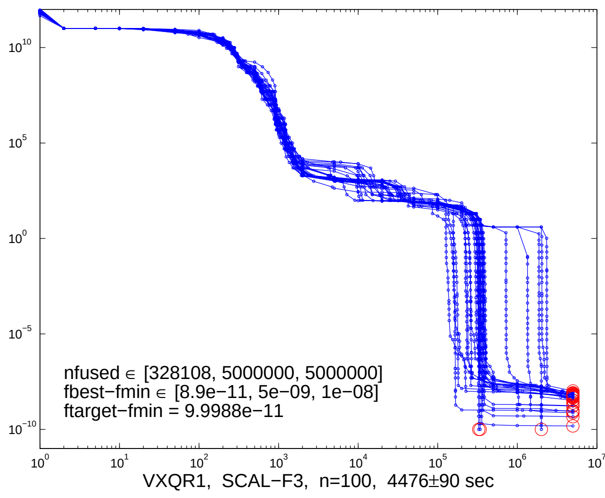
The lower left corner reports the minimum, median, and maximum number of function values needed to reach the target value indicated, and the minimum, median, and maximum of the difference between the best function value and the global minimum.

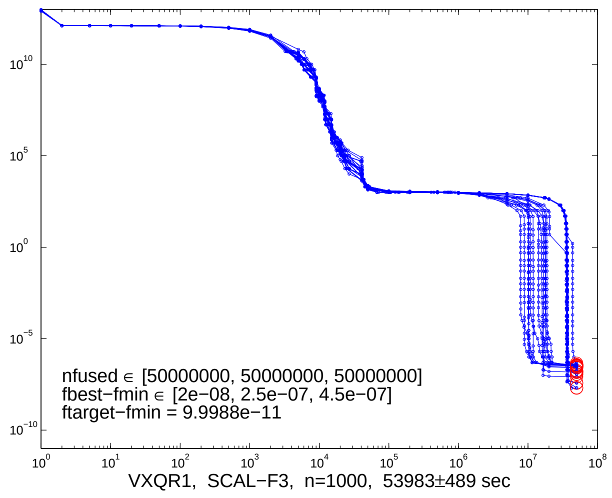
F1





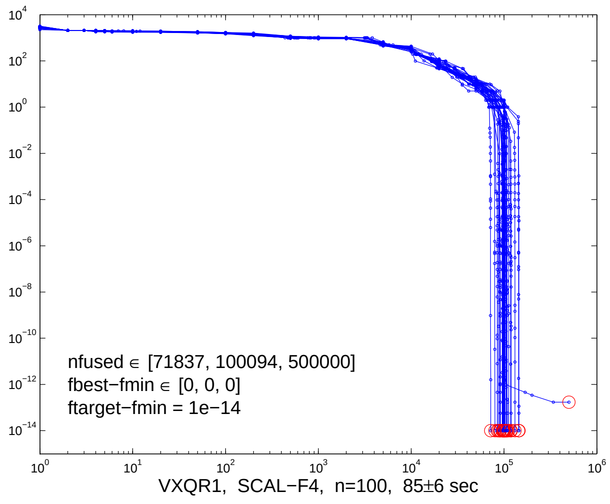
F3

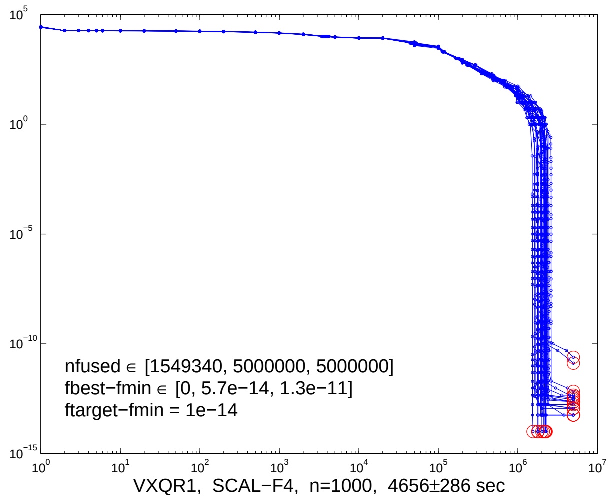




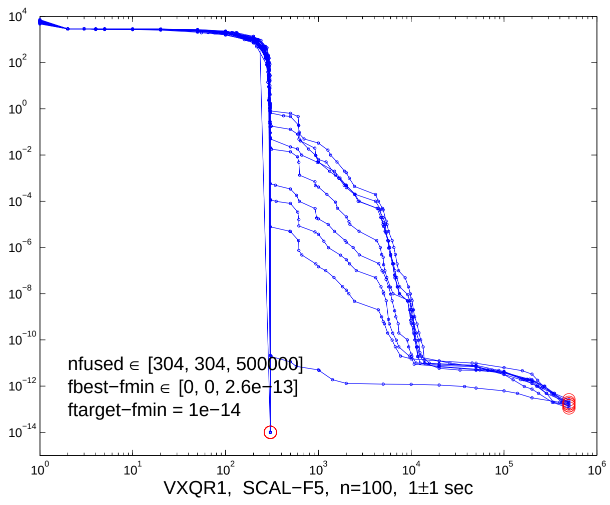
F3 was run twice, first with $5000n$ function values (for the tables), then with $50000n$ function values (to see the scaling behavior). The present figures record the latter, more informative runs.

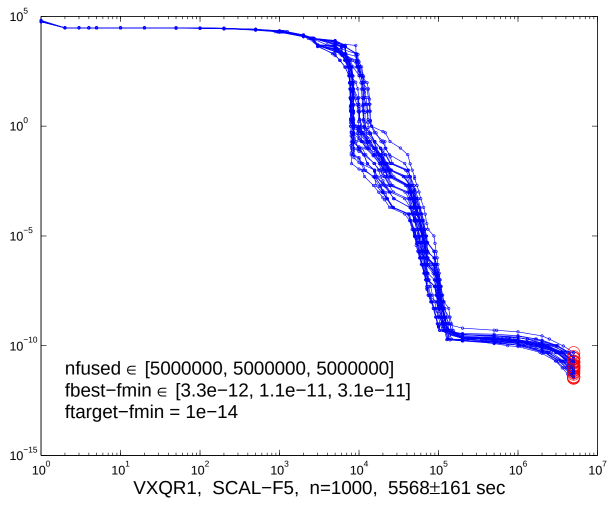
F4


nfused ∈ [71837, 100094, 500000]
fbest−fmin ∈ [0, 0, 0]
ftarget−fmin = 1e−14
VXQR1, SCAL−F4, n=100, 85±6 sec

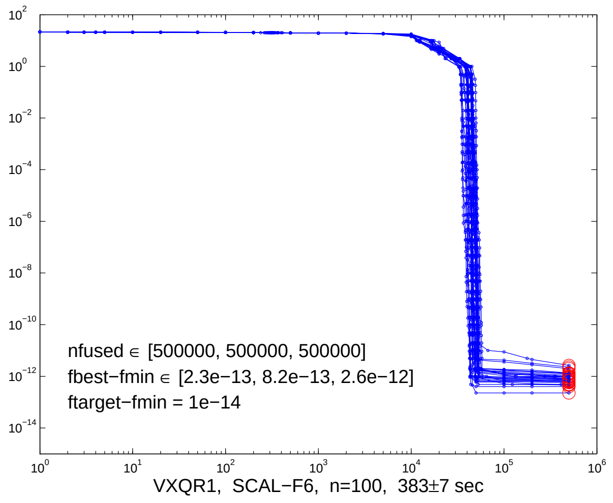

nfused ∈ [1549340, 5000000, 5000000]
fbest−fmin ∈ [0, 5.7e−14, 1.3e−11]
ftarget−fmin = 1e−14
VXQR1, SCAL−F4, n=1000, 4656±286 sec

F5

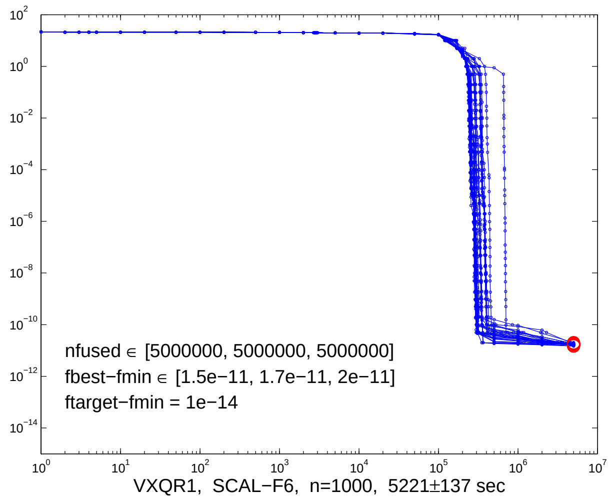

nfused ∈ [304, 304, 500000]
fbest−fmin ∈ [0, 0, 2.6e−13]
ftarget−fmin = 1e−14
VXQR1, SCAL−F5, n=100, 1±1 sec


nfused ∈ [5000000, 5000000, 5000000]
fbest−fmin ∈ [3.3e−12, 1.1e−11, 3.1e−11]
ftarget−fmin = 1e−14
VXQR1, SCAL−F5, n=1000, 5568±161 sec

F6

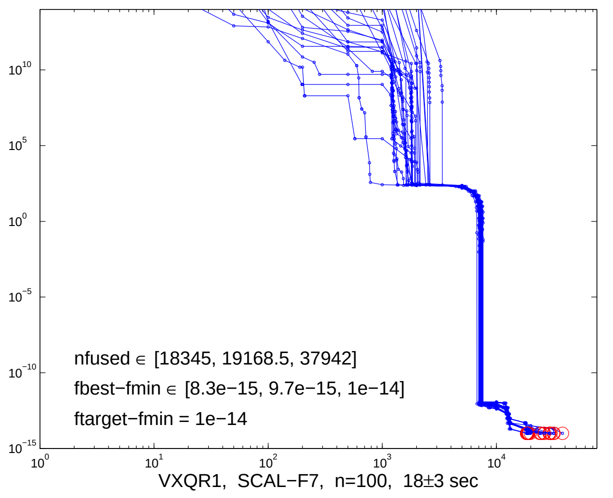


10^{2}
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}
nfused ∈ [500000, 500000, 500000]
fbest−fmin ∈ [2.3e−13, 8.2e−13, 2.6e−12]
ftarget−fmin = 1e−14
VXQR1, SCAL−F6, n=100, 383±7 sec

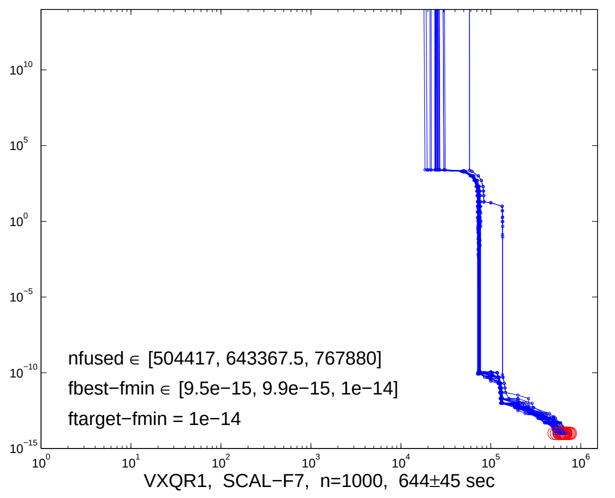


10^{2}
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}
10^{7}
nfused ∈ [5000000, 5000000, 5000000]
fbest−fmin ∈ [1.5e−11, 1.7e−11, 2e−11]
ftarget−fmin = 1e−14
VXQR1, SCAL−F6, n=1000, 5221±137 sec

F7

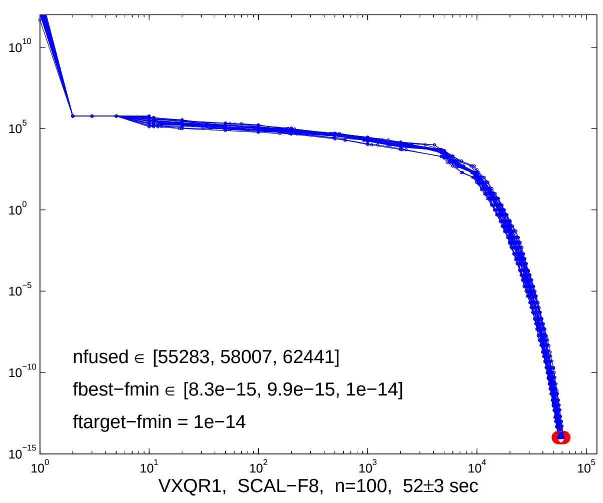


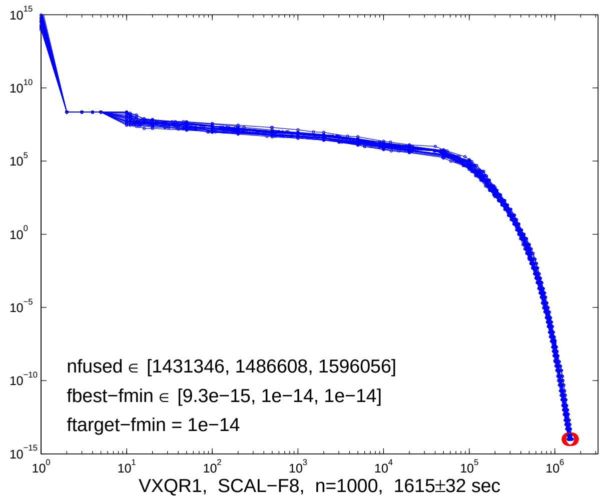
10^{10}
10^{5}
10^{0}
10^{-5}
10^{-10}
10^{-15}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
nfused ∈ [18345, 19168.5, 37942]
fbest−fmin ∈ [8.3e−15, 9.7e−15, 1e−14]
ftarget−fmin = 1e−14
VXQR1, SCAL−F7, n=100, 18±3 sec



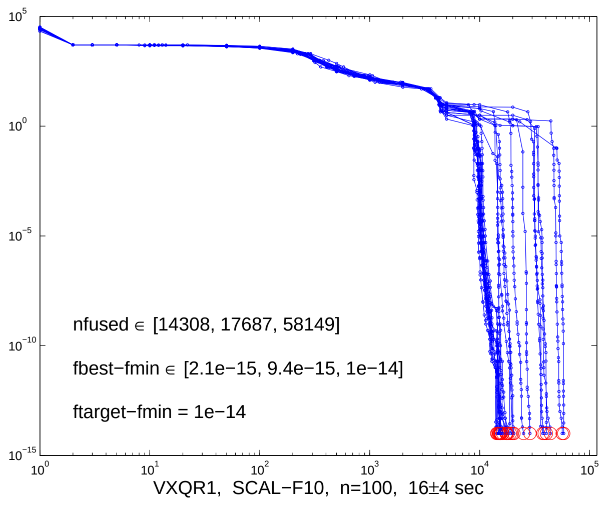
10^{10}
10^{5}
10^{0}
10^{-5}
10^{-10}
10^{-15}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}
nfused ∈ [504417, 643367.5, 767880]
fbest−fmin ∈ [9.5e−15, 9.9e−15, 1e−14]
ftarget−fmin = 1e−14
VXQR1, SCAL−F7, n=1000, 644±45 sec

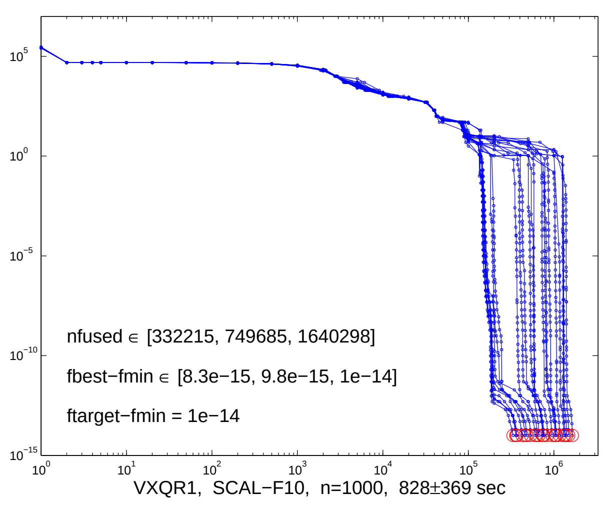
F8


nfused ∈ [55283, 58007, 62441]
fbest−fmin ∈ [8.3e−15, 9.9e−15, 1e−14]
ftarget−fmin = 1e−14
VXQR1, SCAL−F8, n=100, 52±3 sec

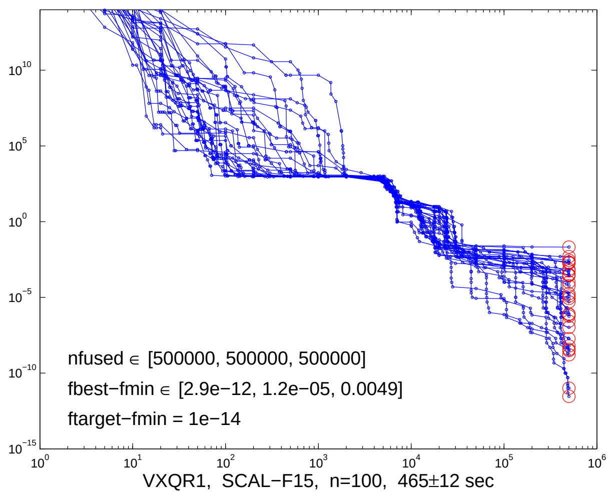

nfused ∈ [1431346, 1486608, 1596056]
fbest−fmin ∈ [9.3e−15, 1e−14, 1e−14]
ftarget−fmin = 1e−14
VXQR1, SCAL−F8, n=1000, 1615±32 sec

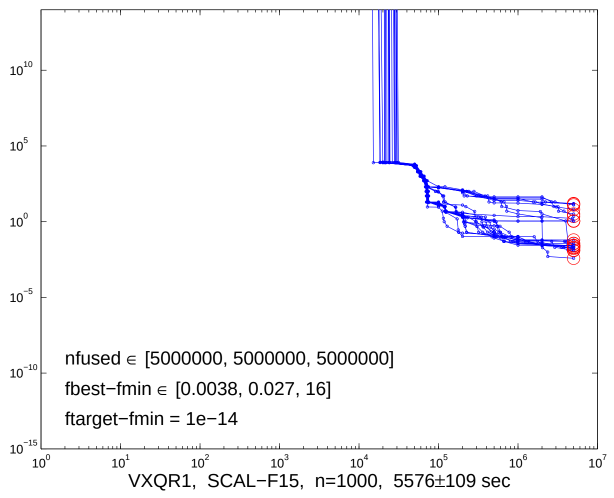
F10


nfused ∈ [14308, 17687, 58149]
fbest−fmin ∈ [2.1e−15, 9.4e−15, 1e−14]
ftarget−fmin = 1e−14
VXQR1, SCAL−F10, n=100, 16±4 sec

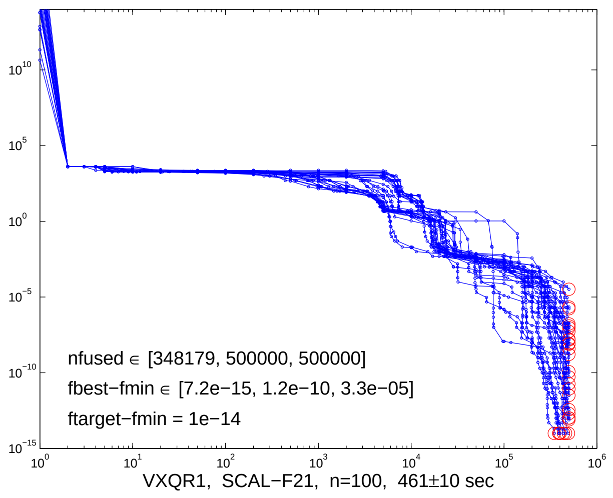

nfused ∈ [332215, 749685, 1640298]
fbest−fmin ∈ [8.3e−15, 9.8e−15, 1e−14]
ftarget−fmin = 1e−14
VXQR1, SCAL−F10, n=1000, 828±369 sec

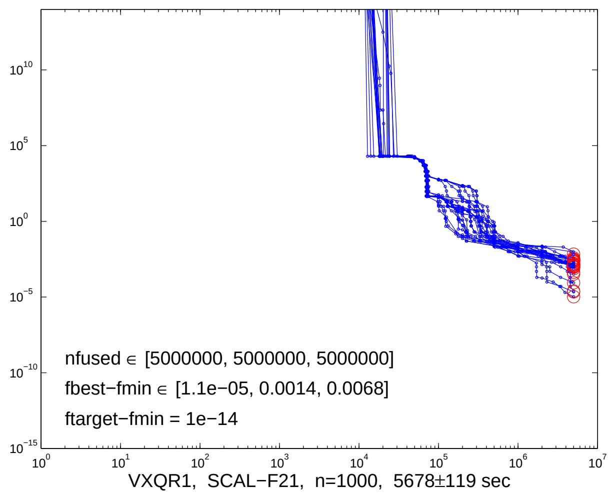
F15


10^10
10^5
10^0
10^−5
10^−10
10^−15
10^0
10^1
10^2
10^3
10^4
10^5
10^6
nfused ∈ [500000, 500000, 500000]
fbest−fmin ∈ [2.9e−12, 1.2e−05, 0.0049]
ftarget−fmin = 1e−14
VXQR1, SCAL−F15, n=100, 465±12 sec


10^10
10^5
10^0
10^−5
10^−10
10^−15
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
nfused ∈ [5000000, 5000000, 5000000]
fbest−fmin ∈ [0.0038, 0.027, 16]
ftarget−fmin = 1e−14
VXQR1, SCAL−F15, n=1000, 5576±109 sec

F19*


10^10
10^5
10^0
10^−5
10^−10
10^−15
10^0
10^1
10^2
10^3
10^4
10^5
10^6
nfused ∈ [348179, 500000, 500000]
fbest−fmin ∈ [7.2e−15, 1.2e−10, 3.3e−05]
ftarget−fmin = 1e−14
VXQR1, SCAL−F21, n=100, 461±10 sec


10^10
10^5
10^0
10^−5
10^−10
10^−15
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
nfused ∈ [5000000, 5000000, 5000000]
fbest−fmin ∈ [1.1e−05, 0.0014, 0.0068]
ftarget−fmin = 1e−14
VXQR1, SCAL−F21, n=1000, 5678±119 sec

Appendix B: Comparison of final median results

$n = 50$	DE	CHC	G-CMA-ES	VXQR1
F1	0.00e+00	1.67e−11	0.00e+00	0.00e+00
F2	3.29e−01	6.19e+01	2.64e−11	1.66e+00
F3	2.90e+01	1.25e+06	0.00e+00	1.83e−09
F4	1.51e−13	7.43e+01	1.08e+02	0.00e+00
F5	0.00e+00	1.67e−03	0.00e+00	5.68e−14
F6	1.42e−13	6.15e−07	2.11e+01	2.84e−13
F7	0.00e+00	2.66e−09	7.67e−11	0.00e+00
F8	3.54e+00	2.24e+02	0.00e+00	0.00e+00
F9	2.73e+02	3.10e+02	1.61e+01	9.21e+00
F10	0.00e+00	7.30e+00	6.71e+00	0.00e+00
F11	5.60e−05	2.16e+00	2.83e+01	1.01e+01
F12	5.27e−13	9.57e−01	1.87e+02	1.34e+00
F13	2.44e+01	2.08e+06	1.97e+02	2.51e+00
F14	2.58e−08	6.17e+01	1.05e+02	1.37e−01
F15	0.00e+00	3.98e−01	8.12e−04	8.01e−08
F16*	1.51e−09	2.95e−09	4.22e+02	4.33e+00
F17*	6.83e−01	2.26e+04	6.71e+02	1.04e+01
F18*	1.20e−04	1.58e+01	1.27e+02	4.75e−01
F19*	0.00e+00	3.59e+02	4.03e+00	0.00e+00

$n = 100$	DE	CHC	G-CMA-ES	VXQR1
F1	0.00e+00	3.56e−11	0.00e+00	0.00e+00
F2	4.34e+00	8.58e+01	1.62e−10	5.17e+01
F3	7.81e+01	4.19e+06	2.27e+00	1.86e−08
F4	4.23e−13	2.19e+02	2.50e+02	0.00e+00
F5	0.00e+00	3.83e−03	0.00e+00	0.00e+00
F6	3.13e−13	4.10e−07	2.13e+01	8.24e−13
F7	0.00e+00	1.40e−02	6.98e−07	0.00e+00
F8	3.47e+02	1.69e+03	0.00e+00	0.00e+00
F9	5.06e+02	5.86e+02	1.06e+02	1.85e+01
F10	0.00e+00	3.30e+01	1.68e+01	0.00e+00
F11	1.29e−04	7.32e+01	1.51e+02	1.74e+01
F12	6.18e−11	1.03e+01	4.20e+02	4.56e+00
F13	6.17e+01	2.70e+06	4.12e+02	1.08e+01
F14	1.30e−07	1.66e+02	2.52e+02	3.59e−01
F15	0.00e+00	8.13e+00	4.13e−01	1.21e−05
F16*	3.53e−09	2.23e+01	8.48e+02	8.34e+00
F17*	1.28e+01	1.47e+05	1.52e+03	2.92e+01
F18*	2.86e−04	7.00e+01	3.13e+02	1.34e+00
F19*	0.00e+00	5.45e+02	1.47e+01	1.16e−10

$n = 200$	DE	CHC	G-CMA-ES	VXQR1
F1	0.00e+00	8.34e−01	0.00e+00	0.00e+00
F2	1.93e+01	1.03e+02	9.91e−10	8.68e+01
F3	1.77e+02	2.01e+07	8.95e+01	2.87e+01
F4	3.58e−12	5.40e+02	6.68e+02	0.00e+00
F5	0.00e+00	8.76e−03	0.00e+00	7.96e−13
F6	6.54e−13	1.23e+00	2.14e+01	2.70e−12
F7	0.00e+00	2.59e−01	2.61e−02	0.00e+00
F8	5.33e+03	9.38e+03	0.00e+00	0.00e+00
F9	1.01e+03	1.19e+03	3.81e+02	4.32e+01
F10	0.00e+00	7.13e+01	4.41e+01	0.00e+00
F11	2.59e−04	3.85e+02	7.93e+02	3.89e+01
F12	9.36e−10	7.44e+01	9.08e+02	2.71e+01
F13	1.36e+02	5.75e+06	9.34e+02	6.21e+01
F14	2.71e−07	4.29e+02	6.24e+02	8.56e−01
F15	0.00e+00	2.14e+01	2.10e+00	5.66e−03
F16*	7.26e−09	1.60e+02	1.90e+03	2.94e+01
F17*	3.70e+01	1.75e+05	3.33e+03	5.66e+01
F18*	4.70e−04	2.12e+02	6.88e+02	4.18e+00
F19*	0.00e+00	2.06e+03	5.74e+02	7.87e−07

$n = 500$	DE	CHC	G-CMA-ES	VXQR1
F1	0.00e+00	2.84e−12	0.00e+00	0.00e+00
F2	5.33e+01	1.29e+02	3.31e−04	9.43e+01
F3	4.74e+02	1.14e+06	3.55e+02	2.11e+02
F4	9.22e−03	1.91e+03	2.07e+03	0.00e+00
F5	0.00e+00	6.98e−03	0.00e+00	5.32e−12
F6	1.65e−12	5.16e+00	2.15e+01	8.98e−12
F7	0.00e+00	1.27e−01	2.14e+143	0.00e+00
F8	6.11e+04	7.22e+04	2.31e−06	0.00e+00
F9	2.52e+03	3.00e+03	1.76e+03	5.95e+01
F10	0.00e+00	1.86e+02	1.27e+02	0.00e+00
F11	6.71e−04	1.81e+03	4.18e+03	7.32e+01
F12	6.98e−09	4.48e+02	2.59e+03	1.08e+02
F13	3.58e+02	3.22e+07	2.87e+03	2.40e+02
F14	9.01e−07	1.46e+03	1.95e+03	2.81e+00
F15	0.00e+00	6.01e+01	5.57e+258	1.19e−02
F16*	2.05e−08	9.55e+02	5.43e+03	3.69e+01
F17*	1.11e+02	8.40e+05	9.50e+03	1.47e+02
F18*	1.22e−03	7.32e+02	2.06e+03	8.33e+00
F19*	0.00e+00	1.76e+03	2.50e+06	8.48e−04

$n = 1000$	DE	CHC	VXQR1
F1	0.00e+00	1.36e−11	0.00e+00
F2	8.44e+01	1.44e+02	9.64e+01
F3	9.69e+02	8.75e+03	6.05e+02
F4	1.32e+00	4.76e+03	5.68e−14
F5	0.00e+00	7.02e−03	1.09e−11
F6	3.30e−12	1.38e+01	1.67e−11
F7	0.00e+00	3.52e−01	0.00e+00
F8	2.46e+05	3.11e+05	0.00e+00
F9	5.13e+03	6.11e+03	9.99e+01
F10	0.00e+00	3.83e+02	0.00e+00
F11	1.35e−03	4.82e+03	1.12e+02
F12	1.70e−08	1.05e+03	2.24e+02
F13	7.29e+02	6.66e+07	5.39e+02
F14	9.95e−01	3.62e+03	5.64e+00
F15	0.00e+00	8.37e+01	2.70e−02
F16*	4.19e−08	2.32e+03	6.35e+01
F17*	2.35e+02	2.04e+07	2.97e+02
F18*	2.37e−03	1.72e+03	1.57e+01
F19*	0.00e+00	4.20e+03	1.37e−03

References

[1] D. Ashlock, Evolutionary Computation for Modeling and Optimization, Springer, 2006.

[2] A. Auger and N. Hansen, A restart CMA evolution strategy with increasing population size. pp. 1769–1776 in: The 2005 IEEE Congress on Evolutionary Computation, vol 2 (2005).

[3] C.J. Bélisle, H.E. Romeijn, and R.L. Smith, Hit-and-run algorithms for generating multivariate distributions, Mathematics of Operations Research 18 (1993), 255–266.

[4] A.R. Conn, K. Scheinberg, and L.N. Vicente, Introduction to derivative-free optimization. SIAM, Philadelphia, PA, 2009.

[5] T. Csendes, L. Pál, J.O.H. Sendin, and J.R. Banga, The GLOBAL Optimization Method Revisited, Optimization Letters, 2 (2008), 445–454.

[6] M. Dorigo and T. Stützle, Ant Colony Optimization, MIT Press 2004.

[7] R. Eberhart and J. Kennedy, A new optimizer using particle swarm theory, Proc. Sixth Int. Symp. Micro Machine and Human Science. Nagoya, Japan, 1995, 39–43.

[8] L.J. Eshelman and J. D. Schaffer, Real-coded genetic algorithm and interval schemata. pp. 187–202 in: Foundations of Genetic Algorithms Workshop (FOGA-92) (D. Whitley, ed.), Vail, CO, 1993.

[9] S.S. Fan and E. Zahara, A hybrid simplex search and particle swarm optimization for unconstrained optimization. European Journal of Operational Research 181(2007), 527–548.

[10] P. Gilmore and C.T. Kelley, An implicit filtering algorithm for optimization of functions with many local minima. SIAM Journal on Optimization 5 (1995), 269–285.

[11] F. Glover, Tabu thresholding: Improved search by nonmonotonic trajectories. ORSA Journal on Computing 7 (1995), 426–442.

[12] N. Hansen, The CMA evolution strategy: A comparing review. pp. 75–102 in: Towards a new evolutionary computation. Advances on estimation of distribution algorithms (J.A. Lozano et al., eds.), Springer 2006.

[13] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Pošik, Comparing Results of 31 Algorithms from the Black-Box Optimization Benchmarking BBOB-2009, Manuscript (2010). `http://www.lri.fr/~hansen/gecco09-results-2010.pdf`

[14] F. Herrera, M. Lozano, and D. Molina, Test Suite for the Special Issue of Soft Computing on Scalability of Evolutionary Algorithms and other Metaheuristics for Large Scale Continuous Optimization Problems. Web document (2010), `http://sci2s.ugr.es/eamhco/CFP.php`

[15] R. Hooke and T.A. Jeeves. "Direct Search" Solution of Numerical and Statistical Problems, Journal of the ACM 8 (1961), 212–229.

[16] W. Huyer and A. Neumaier. Global Optimization by Multilevel Coordinate Search. Journal of Global Optimization 14 (1999), 331–355.

[17] W. Huyer and A. Neumaier. SNOBFIT – Stable Noisy Optimization by Branch and Fit. ACM Transactions on Mathematical Software 35 (2008), Article 9.

[18] D.R. Jones, M. Schonlau and W.J. Welch: Efficient Global Optimization of Expensive Black-Box Functions. Journal of Global Optimization, 13:455–492, 1998.

[19] D.R. Jones, C.D. Perttunen and B.E. Stuckman: Lipschitzian optimization without the Lipschitz constant. J. Optimization Th. Appl. 79 (1993), pp. 157–181.

[20] C.T. Kelley, Detection and remediation of stagnation in the Nelder-Mead algorithm using a sufficient decrease condition. SIAM Journal on Optimization 10 (1999), 43–55.

[21] T.G. Kolda, R.M. Lewis and V.J. Torczon, Optimization by direct search: New perspectives on some classical and modern methods, SIAM Review 45 (2003), 385–482.

[22] Z. Michalewicz and D.B. Fogel, How To Solve It: Modern Heuristics, Springer, 2004.

[23] NAG Library Chapter Contents. E05 – Global Optimization of a Function. NAG Library, Mark 22 (2009). `http://www.nag.co.uk/numeric/FL/nagdoc_fl22/xhtml/E05/e05conts.xml`

[24] J.A. Nelder and R. Mead, A simplex method for function minimization, Computer Journal 7 (1965), 308–313.

[25] A. Neumaier, Introduction to Numerical Analysis, Cambridge Univ. Press, Cambridge 2001.

[26] A. Neumaier, VXQR1. Web document, `http://http://www.mat.univie.ac.at/~neum/software/vxqr1/`

[27] J.D. Pintér, Global Optimization in Action: Continuous and Lipschitz Optimization. Algorithms, Implementations and Applications. Kluwer, Dordrecht 1995.

[28] M.J.D. Powell, Developments of NEWUOA for minimization without derivatives. IMA Journal of Numerical Analysis 28 (2008), 649–664.

[29] L.M. Rios. *Algorithms for derivative-free optimization.* PhD thesis, University Of Illinois At Urbana-Champaign, 2009.

[30] L.M. Rios and N.V. Sahinidis. Derivative-free optimization: A review of algorithms and comparison of software implementations. *"Advances in Optimization II"* (AIChE 2009).

[31] J. Sacks, W.J. Welch, T.J. Mitchell and H.P. Wynn, Design and analysis of computer experiments (with discussion), Statistical Science 4, pp. 409–435, 1989.

[32] R. Storn and K. Price, Differential Evolution – a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, Journal of Global Optimization 11 (1997), 341–359.

[33] V. Torczon, On the convergence of multidirectional search algorithms, SIAM Journal on Optimization 1 (1991), 123–145.

[34] A. Törn and A. Žilinskas. Global Optimization, Springer, New York 1989.

[35] P.J.M. Van Laarhoven and E.H.L. Aarts: Simulated Annealing: Theory and Applications. Kluwer, Dordrecht, 1987.