# A New Hybrid Classical-Quantum Algorithm for Continuous Global Optimization Problems

Pedro C. S. Lara[1], Renato Portugal[1], and Carlile Lavor[2]

[1]Laboratório Nacional de Computação Científica, Petrópolis, Brazil,
{pcslara,portugal}@lncc.br
[2]IMECC/Unicamp, Campinas, Brazil,
clavor@ime.unicamp.br

**Abstract**

Grover's algorithm can be employed in global optimization methods providing, in some cases, a quadratic speedup over classical algorithms. This paper describes a new method for continuous global optimization problems that uses a classical algorithm for finding a local minimum and Grover's algorithm to escape from this local minimum. Simulations with testbed functions and comparisons with algorithms from the literature are presented.

**Keywords:** Global optimization, Quantum computing, Continuous functions, Grover's algorithm.

## 1    Introduction

In general, we could say that global optimization is about finding the best solution for a given problem. Global optimization algorithms play an important role in many practical problems. A recent review of different methods in global optimization can be found in [4].

Recently, some papers addressed the problem of finding the global minimum of discrete [2, 7, 8] and continuous functions [9, 10] using quantum algorithms, where the method used in the discrete case is an extension of Dürr and Høyer's (DH) algorithm [3], which in turn is based on the quantum search algorithm developed by Boyer *et. al.* (BBHT) [1]. This was made possible after Lov Grover has discovered the seminal algorithm for searching one item in an unsorted database with $N$ distinct elements [5]. Grover's algorithm finds the position of the desired element querying the database $O(\sqrt{N})$ times, which is a quadratic improvement with respect to the number of times classical algorithms query the database.

The goal of this work is to present a new method to solve continuous global optimization problems. It is a hybrid method that uses an efficient classical algorithm for finding a local minimum and a quantum algorithm to escape from that, in order to reach the global minimum.

The article is organized as follows. In Sec. 2, we describe the relationship between quantum algorithms and global optimization problems, and we review the DH and Baritompa *et. al* (BBW) algorithms. In Sec. 3, we describe the new method proposed in this paper. In Sec. 4, we present the simulations and discuss the results of this work. Finally, in Sec. 5, we present our conclusions.

## 2    Quantum Search and Global Optimization Problems

We start by describing the problem that Grover's algorithm addresses. Consider a boolean function $h : \{0, \cdots, N-1\} \to \{0, 1\}$, where $N = 2^n$ and $n$ is some positive integer, such that

$$h(x) = \begin{cases} 1, & \text{if } x \in M; \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

where $M \subseteq \{0, \cdots, N-1\}$. The goal is to find an element $x_0 \in M$ by querying function $h(x)$ the least number of times. Grover [5] described a quantum algorithm that finds $x_0$ when $|M| = 1$, that is, the only element in $M$ is $x_0$. Grover's algorithm finds $x_0$ with probability greater than or equal to $1 - 1/N$ by querying $h$ around $\frac{\pi}{4}\sqrt{N}$ times. Internally, the algorithm uses an initial vector in a Hilbert space that undergoes $\frac{\pi}{4}\sqrt{N}$ rotations of small angles $\theta$, such that $\sin(\theta/2) = 1/\sqrt{N}$. The initial vector is the normalized vector obtained by adding all vectors of the canonical basis (also known by computational basis) of the $N$-dimensional Hilbert space, which yields a vector of equal entries, the value of which are $1/\sqrt{N}$. After $\frac{\pi}{4}\sqrt{N}$ Grover's rotations, the final vector has a large overlap with the vector that represents the solution. This is the guarantee one needs to be sure that a measurement of the quantum computer in this final state will yield $x_0$ with high probability. One rotation in the algorithm is also called a Grover iteration. In each Grover iteration, function $h$ is queried one time. The number of times $h$ is queried is the main parameter to measure the efficiency of hybrid classical-quantum algorithms.

Boyer *et. al.* generalized Grover's algorithm in two directions [1]. Firstly, they considered the case $|M| > 1$ and showed that the number of rotations required to find one element in $M$ with probability greater than or equal to $1 - 1/N$ is

$$\frac{\pi}{4}\sqrt{\frac{N}{|M|}}.$$

The number of Grover iterations decreases when $|M| > 1$ because the dimension of the subspace of the Hilbert space spanned by elements in $M$ is $|M|$. One obtains a large overlap between the final vector state of the quantum computer with the solution-subspace with less Grover rotations. That is a straightforward generalization of Grover's algorithm.

Secondly, the authors addressed the problem of finding one element in $M$ without knowing a priori the number of elements in $M$. The main problem in this case is to know what is the best number of rotations. If the algorithm performs too few or too many rotations, the probability to find the correct results becomes small. Their strategy is to start the algorithm by performing a small number of Grover rotations followed by a measurement, which yields an element $x \in \{0, \cdots, N-1\}$. One checks whether $h(x) = 1$. If that fails, start over again and increase the number of Grover rotations. The key point is to determine the increment rate. Formally, the BBHT algorithm [1], which finds a marked element when the number of solutions is not known in advance (unknown $|M|$), can be written in a pseudo-code as Algorithm 1. Boyer *et. al.* proved that the expected running time of the algorithm is $O(\sqrt{N}/|M|)$ if each query to function $h$ is evaluated in unit time. They observed that any value of $\lambda$ in the range $1 < \lambda < 4/3$ is allowed.

---

**Algorithm 1:** BBHT Algorithm

---

1 **begin**
2      Initialize $m = 1$ and set $\lambda = 8/7$;
3      Choose an integer $j$ uniformly at random such that $0 \le j < m$;
4      Apply $j$ Grover's iterations starting from the initial vector;
5      Perform a measurement (let $x$ be the outcome);
6      If $h(x) = 1$, return the result $x$;
7      Otherwise, set $m$ to $\min\{\lambda m, \sqrt{N}\}$ and go to line 3;
8 **end**

---

Using the BBHT algorithm, Dürr and Høyer [3] proposed an algorithm to find the minimum element of a finite list $L$, which can be seen either as finding the index of the smallest element in a database or as a discrete global optimization problem. At the beginning, the algorithm selects at random one element $y$ in $L$ and searches for elements in set $M = \{y' \in L | y' < y\}$ using the BBHT algorithm. If it succeeds, $y'$ is the new candidate for minimum value and the BBHT algorithm will be used again and again until the minimum is found with probability greater than $1/2$. In order to use the BBHT algorithm in the way we have described, we have to suppose that the number of elements of the list is $N = 2^n$ and we have to search for the indices of the elements (instead for the elements themselves), because the domain of function $h$ in Grover's algorithm

is $\{0, \cdots, N-1\}$. The details can be found in Ref. [3]. Dürr and Høyer showed that the running time of the algorithm is $O(\sqrt{N})$ and the probability of finding the minimum is at least $1/2$. Their analysis is valid when all elements of the list are distinct.

Baritompa *et. al* (BBW) [2] used the DH algorithm to propose a generic structure of a quantum global optimization algorithm for a discrete function $f : \{0, \cdots, N-1\} \to L$, where $L$ is a list of $N$ numbers, such that $f(x)$ is the $(x+1)$-th element in $L$. We use the notation $h$ for the oracle function and $f$ for the function the minimum of which we want to find. Note that in the DH algorithm, the BBHT algorithm is used as a black box. BBW uses the BBHT algorithm explicitly, and can be written in a pseudo-code as Algorithm 2. GAS is the basis for the quantum algorithm used in this work.

---

**Algorithm 2:** Grover Adaptive Search (GAS)

---

1 **begin**
2      Generate $x_0$ uniformly in $\{0, \cdots, N-1\}$ and set $y_0 = f(x_0)$.;
3      **for** $i = 1, 2, \cdots$, *until a termination condition is met* **do**
4          Perform $r_i$ Grover's rotations marking points with image $\leq y_{i-1}$. Denote outputs by $x$ and $y$;
5          **if** $y < y_{i-1}$ **then**
6              set $x_i = x$ and $y_i = y$;
7          **else**
8              set $x_i = x_{i-1}$ and $y_i = y_{i-1}$;
9          **end**
10      **end**
11 **end**

---

GAS reduces to the DH algorithm if: (1) the integer $r_i$ is chosen uniformly at random in the range $0 \leq r_i < m$, (2) $m$ is incremented as $m = \lambda m$ if $y \geq y_i$ and $m = 1$ otherwise, where $\lambda = 8/7$, as in the BBHT algorithm, and (3) the termination condition is that the accumulated number of Grover's rotations is greater than $22.5\sqrt{N} + 1.4 \log^2 N$.

Baritompa *et. al* improved the prefactor that describes the running time of the BBHT algorithm. They prove that for $\lambda = 1.34$, the expected number of oracle queries for the BBHT algorithm to find and verify a marked element repeated $t$ times is at most $1.32\sqrt{N/|M|}$ $\left(\text{BBHT uses the threshold } 8\sqrt{N/|M|}\right)$. They also provide a detailed proof of the quadratic speedup of the DH algorithm when there are repeated elements. They have proposed a new version of the minimization algorithm by changing the method of choosing the number of Grover's rotations in each round of the algorithm. Instead of selecting $r_i$ at random, they have proposed a deterministic method in such way that the number of rotations follows a pre-computed sequence of integers. The BBW version avoids to set $m = 1$ each time the algorithm finds a new candidate for minimum. This is also proposed in Ref. [6], which shows that the number of measurement reduces from $O(\log^2 N)$ in the DH algorithm to $O(\log N)$ in the version that set $m = 1$ only once at the beginning of the algorithm. The running time of the BBW deterministic version is $2.46\sqrt{N}$, when there are no repeated elements while the running time of the DH algorithm is $22.5\sqrt{N}$. There is no way to improve the scaling of those algorithms using quantum computing, as been proved by Bennett *et. al* [11] and Zalka [12]. Only the prefactor may be reduced.

## 3    A New Method for Continuous Functions

The new method proposed in this paper is a hybrid algorithm that employs a classical optimization routine to find a local minimum and the GAS algorithm to escape from that minimum towards another better candidate. We consider continuous and differentiable real functions $f : D \to \mathbb{R}$ with $n$ variables, where the domain $D$ is a $n$-dimensional finite box. In order to implement a computer program to find the global minimum point of a continuous function, we discretize the function domain using intervals of same length $\epsilon$ for all variables and we convert the domain points, which form a $n$-dimensional array, into a one-dimensional array, generating a

list of $N$ points. After this discretization and conversion to one-dimensional representation, the domain of $f$ can be taken as the set $\{0, \cdots, N-1\}$. This process is mandatory because to perform Grover's rotations we need an oracle function $h$ with domain $\{0, \cdots, N-1\}$. The value of $\epsilon$ depends on the structure of the function and on the optimization problem. This parameter will be used in classical optimization routines to characterize the precision of local minimum points, and at the end to characterize the precision of the global minimum point. The general structure of the new algorithm is given in Algorithm 3.

---

**Algorithm 3:** The New Method

---

1 **begin**
2     Generate $x'$ uniformly at random in $\{0, \cdots, N-1\}$;
3     Use a classical optimization routine with input $x'$ to find a local minimum $x_0$ and set $y_0 = f(x_0)$;
4     Set $m = 1$ and $\lambda = 1.34$ (as suggested in BBW);
5     **for** $i = 1, 2, \cdots$, *until a termination condition is met* **do**
6        Define $M_i = \{x \in \{0, \cdots, N-1\}|f(x) < y_{i-1}\}$;
7        Choose $r_i$ uniformly at random in $\{0, \cdots, \lceil m-1 \rceil\}$;
8        Apply $r_i$ Grover's rotations;
9        Perform a measurement. Let $x' \in \{0, \cdots, N-1\}$ be the output;
10        **if** $x' \in M_i$ **then**
11           Use the classical optimization routine with input $x'$ to find a local minimum $x_i$;
12           Set $y_i = f(x_i)$;
13        **else**
14           Set $x_i = x_{i-1}$, $y_i = f(x_i)$, and $m = \min\{\lambda m, \sqrt{N}\}$;
15        **end**
16     **end**
17     **return** last value of $x_i$;
18 **end**

---

The *termination condition* determines the running time of the algorithm. In the new method, the termination condition takes into account the total number of Grover's rotations and the total number that the objective function is evaluated when classical optimization routines are employed. The weight of the classical objective function evaluation is higher by a factor of $\sqrt{N}/\log N$ compared to the function evaluation in each Grover's rotation. If $n_1$ is the number of Grover's rotations and $n_2$ is the number of objective function evaluations in classical optimization routines, then the termination condition is

$$n_1 + \frac{\sqrt{N}}{\log^n N}\, n_2 > 2.46\sqrt{N}, \tag{2}$$

where $n$ is the number of variables of the objective function. In the worst case, the algorithm has running time $O(\sqrt{N})$, and in cases for which classical algorithms are able to find the global minimum without using the quantum part, the running time is $O(\log^n N)$.

Algorithm 3 is a randomized algorithm. The success probability cannot be calculated until the classical optimization algorithm is specified. We are assuming that in the worst case the classical algorithm will take $O(\log^n N)$ steps to find a local minimum after a initial point is given. The computational results presented in the next Section confirm that assumption.

## 4    Computational Results

Baritompa *et. al* compare their improved version of the minimization algorithm (BBW) to the one of Dürr and Høyer's (DH) by displaying performance graphs, that depict the success probability of the algorithms in terms of the (computational) effort. The effort is the number of objective function evaluations before a new candidate for minimum point is found plus the number of measurements. The number of measurements

does not play an important role for large $N$, because it scales logarithmically in terms of $N$. In the first part of this Section, we use the same technique to show that the new method generates better results. This kind of analysis was performed in Ref. [13] for adaptive random search, the structure of which is similar to the Grover adaptive search. The details about the comparison between BBW and DH algorithms using performance graphs can be obtained in Refs. [2, 7, 8].
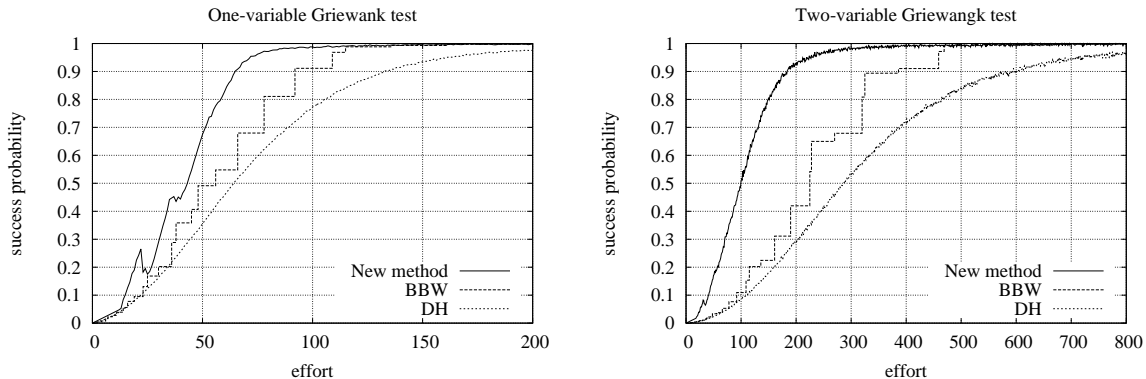


Figure 1: Performance graphs comparing the new method with Baritompa *et. al.* (BBW) and Dürr-Hoyer (DH) using one and two-variable Griewank test functions.

Figs. 1 and 2 show the performance graphs that compare the new method with the BBW and DH algorithms. We use one, two, and three-variable Griewank test functions with domain $-40 \leq x_0, x_1, x_2 \leq 40$, which are described in Appendix A. The classical routine used in Algorithm 3 (lines 3 and 11) to find a local minimum is the BOBYQA routine [25]. To generate those graphs, we create a sample with $N$ function values taking $\{0, \ldots, N-1\}$ as the domain set (as described in Sec. 3). The value of $N$ for one-variable Griewank test is $N = 2048$, for two-variable is $N = 256^2$, and for three-variable is $N = 64^3$. We average out this process 100,000 times for each graph. We take larger parameter $\epsilon$ for the three-variable case, because to calculate the average is time-consuming. All algorithms are implemented in the C language, and it takes about half an hour in a 2.2 GHz Intel Core i7 processor to generate the graphs of Fig. 2.
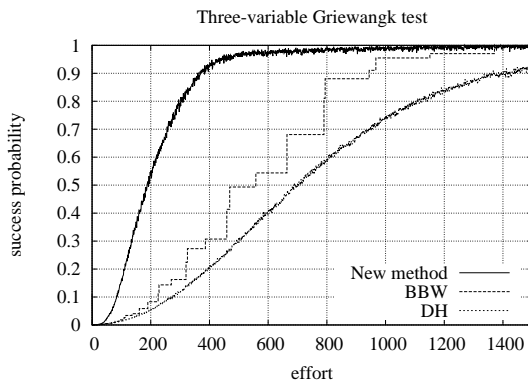


Figure 2: Performance graphs comparing the new method with Baritompa *et. al.* (BBW) and Dürr-Hoyer (DH) using three-variable Griewank test function.

The new method is better than the previous methods in all those tests. Note that it becomes even better when the number of variables increases, as can be seen in Fig. 2. To have a success probability of 50% for

the three-variable Griewank function, the effort of the new method is less than 200 and the effort of the BBW and DH algorithms must be greater than 500. Similar curves are obtained when we use other classical optimization routines, mainly for one and two-variable objective functions. For three-variable functions, some optimization routines did not produce good results. We discuss this problem later on, when we compare the efficiency of different optimization routines.

To compare the new method with BBW and DH in more details, we employ the most used performance test problems in global optimization, selecting test functions with one to three variables (the list of test functions is in Appendix A). For our experiments we use the *NLopt C library* [14]. NLopt is an open-source library under GNU license for nonlinear optimization with a collection of classical optimization routines. It can be used both for global and local optimization problems. In this work, we use this library for the latter case. It is written in C and has interface with many languages. The specific routines that we use in this work are:

- LBFGS - This routine is based on variable-metric updates via Strang recurrence of the low-storage BFGS routine [15, 16].

- TNEWT - This routine is based on truncated Newton algorithms for large-scale optimization [17].

- MMA - This routine is based on a globally-convergent method-of-moving-asymptotes algorithm for gradient-based local optimization, including nonlinear inequality constraints [18].

- COBYLA - This routine is a constrained optimization by linear Approximations algorithm for derivative-free optimization with nonlinear inequality and equality constraints [19].

- NMEAD - This routine is a simplex method for function minimization (Nelder-Mead simplex algorithm) [20, 21].

- AGL - This routine is a globally convergent augmented Lagrangian algorithm with general constraints and simple bounds [23, 24].

- BOBYQA - This routine performs derivative-free bound-constrained optimization using an iteratively constructed quadratic approximation for the objective function [25].

- CCSA - This routine is a conservative convex separable approximation method, which is a variant of MMA algorithm [18].

In the experiments that compare the optimization routines, we create a sample with $N$ function values taking $\{0, \dots, N-1\}$ as the domain set (as described in Sec. 3). The value of $N$ depends on the number of variables: for functions with one variable, we take $N = 2048$, for 2 variables, $N = 2048^2$, and for 3 variables, $N = 256^3$. With this sample, we compute the running time (number of objective function evaluations) of each algorithm until they find the correct global minimum. We are overriding the termination conditions in all algorithms in order to check what is the total number of function evaluations until the correct minimum is found. We average out this process 100 times for each routine.

Table 1 shows the total number of function evaluations for the one-variable case performed by the new method. The first line lists the optimization routine used in Algorithm 3 (routines with asterisk use function derivative) and the first column lists the name of the test function. The expression and domain of the test functions are described in Appendix A. To highlight the best and worst routines described in Table 1, we highlight the smallest number of evaluations in blue and largest in red for each test function. The results show that the routine efficiency depends on the test function in general. For one-variable functions, the LBFGS and BOBYQA routines have the best performance while NMEAD has the worst, except for the Shekel and Schwefel functions. The timings increase when we increase the number of sample values (smaller $\epsilon$), but the table structure remains the same. On the other hand, if we decrease too much the number of sample values, the structure of the table changes significantly, and the information about the best and worst routines is almost meaningless. From the timings in Table 1, we cannot conclude that routines using function derivative

| rout.<br>fcn. | LBFGS* | TNEWT* | MMA* | COBYLA | NMEAD | SBPLX | AGL* | BOBYQA | CCSAQ* |
|---|---|---|---|---|---|---|---|---|---|
| Neumaier | **9.00** | 17.52 | 33.00 | 11.00 | 193.5 | **236.2** | 34.00 | 11.00 | 32.67 |
| Griewank | 58.63 | 92.77 | 104.0 | 99.21 | **252.5** | 214.5 | 108.8 | **52.61** | 108.6 |
| Shekel | 12.00 | 15.00 | 32.67 | 11.81 | **5.05** | 11.37 | **33.66** | 11.00 | 33.00 |
| Rosenbrock | **84.86** | 110.5 | 153.8 | 129.7 | **220.4** | 207.9 | 159.1 | 101.7 | 165.5 |
| Michalewicz | 55.93 | 92.71 | 85.75 | 131.2 | **185.7** | 153.6 | 89.46 | **39.06** | 91.82 |
| Dejong | **9.00** | 15.00 | 32.67 | 84.74 | 133.7 | **179.7** | 33.66 | 11.00 | 33.00 |
| Ackley | 90.39 | 106.3 | 129.6 | 136.6 | **212.8** | 210.3 | 141.6 | **44.49** | 145.4 |
| Schwefel | 9.00 | 12.00 | **66.26** | 21.95 | **4.00** | 25.15 | 34.00 | 19.57 | 33.00 |
| Rastrigin | 75.75 | 91.58 | **33.00** | 70.12 | **244.9** | 236.6 | 76.09 | 36.51 | 89.57 |
| Raydan | **25.87** | 43.97 | 33.00 | 55.94 | **191.8** | 178.6 | 34.00 | 28.89 | 36.73 |

Table 1: Average number of evaluations of one-variable testbed functions using optimization routines specified in the first line. Smallest number of evaluations in blue and largest in red.

| rout.<br>fcn. | LBFGS* | TNEWT* | MMA* | COBYLA | NMEAD | SBPLX | AGL* | BOBYQA | CCSAQ* |
|---|---|---|---|---|---|---|---|---|---|
| Neumaier | 1126 | 1671 | 1434 | **22.00** | 8943 | **9771** | 1204 | 122.8 | 1213 |
| Griewank | 1583 | 1441 | 1290 | 612.0 | **9285** | 9077 | 1012 | **412.2** | 1036 |
| Shekel | 2643 | **2785** | 1575 | 22.00 | **8.00** | 22.01 | 1824 | 21.78 | 1933 |
| Rosenbrock | **217.3** | 770.2 | 1333 | 5762 | 7614 | **9707** | 1300 | 1086 | 1155 |
| Michalewicz | 1707 | 1881 | 1409 | 3887 | **10323** | 9841 | 1283 | **390.4** | 1258 |
| Dejong | 1534 | 1316 | 1379 | 22.00 | 6174 | **7356** | 1477 | **21.78** | 1433 |
| Ackley | 2385 | 2083 | 1487 | 1958 | **7611** | 7232 | 2001 | **638.0** | 2822 |
| Schwefel | **2101** | 1795 | 1422 | 356.7 | **53.71** | 1502 | 1306 | 70.32 | 1475 |
| Rastrigin | 1390 | 1219 | 1413 | 1360 | **7695** | 6180 | 1265 | **191.4** | 1153 |
| Raydan | 1718 | 1726 | 1445 | 467.0 | 9430 | **10211** | 1540 | **324.7** | 1651 |

Table 2: Average number of evaluations of two-variable testbed functions using optimization routines specified in the first line. Smallest number of evaluations in blue and largest in red.

are better than routines that do not use derivative. The total simulation time to produce the data in Table 1 is at order of some minutes.

Table 2 shows the total number of function evaluations for the two-variable case performed by the new method. The results show that the best-performance routines depend heavily on the test function. The BOBYQA routine has the best performance, while SBPLX has the worst performance. It is remarkable that the NMEAD routine has the best performance for the Shekel and Schwefel test functions and the worst performance for the Griewank, Michalewicz, Ackley, and Rastrigin functions. This behavior also occurs with one-variable functions, as we remarked earlier. All those six functions are multimodal. The total simulation time to produce the data in Table 2 is about one hour.

Table 3 shows the total number of function evaluations for the three-variable case performed by the new method. The results show again that the best-performance routines depend heavily on the test function. The BOBYQA routine has a small advantage while NMEAD and SBPLX have the worst performance. Notice that we are using $N = 256^3$ in the discretization procedure, which means that we use 256 points in each axis. This number is rather small and represents the continuous functions in a gross manner. We do not use larger number of function values, because the total simulation time, which includes the averages, is long. Using such small number of function values, the best method seems to be BOBYQA and the worst NMEAD, similar to what happens with two-variable functions. It is also similar to what happens with one-variable functions, except that LBFGS is not as efficient as in one-variable case. The total simulation time to produce the data in Table 3 is about five hours.

Table 4 shows the total number of objective function evaluations until the global minimum is found for the three algorithms used in this work. In this experiment, we use the same discretization parameters used in the previous tables and we override again the termination condition of the algorithms, that is, the algorithms run until the correct global minimum is found. When we use the correct termination condition for each

| rout.<br>fcn. | LBFGS* | TNEWT* | MMA* | COBYLA | NMEAD | SBPLX | AGL* | BOBYQA | CCSAQ* |
|---|---|---|---|---|---|---|---|---|---|
| Neumaier | 1789 | 4467 | 3622 | 4147 | 6099 | **6805** | 4080 | **1390** | 3215 |
| Griewank | 2932 | 2172 | 1500 | 1128 | 11392 | **12001** | 3179 | **711.4** | 938.7 |
| Shekel | 4775 | **5137** | 1900 | 24.00 | **12.00** | 24.00 | 1300 | 24.00 | 1742 |
| Rosenbrock | **1776** | 1847 | 1898 | 3795 | 18074 | **18199** | 2592 | 4018 | 2531 |
| Michalewicz | 8051 | 4843 | – | 4037 | **14779** | 9944 | 2209 | 3873 | **1468** |
| Dejong | 2604 | 2685 | 1582 | 1005 | **9735** | 6608 | 1953 | **23.76** | 1338 |
| Ackley | – | **918.3** | 1674 | 2944 | **16455** | 15680 | 2390 | 1098 | 1249 |
| Schwefel | 6988 | **7549** | 557.0 | **141.7** | 1100 | 209.3 | 883.2 | 172.4 | 1619 |
| Rastrigin | 2578 | **555.7** | 1779 | 1638 | **11386** | 9408 | 1365 | 925.1 | 1180 |
| Raydan | 2826 | 2908 | 1816 | **726.9** | **17283** | 16469 | 2081 | 780.1 | 1628 |

Table 3: Average number of evaluations of three-variable testbed functions using optimization routines specified in the first line. Smallest number of evaluations in blue and largest in red.

| # variables | one variable | | | two variables | | | three variables | | |
|---|---|---|---|---|---|---|---|---|---|
| meth.<br>fcn. | NEW | BBW | DH | NEW | BBW | DH | NEW | BBW | DH |
| Neumaier | **9.00** | 97.50 | **112.4** | **22.00** | **960.8** | 694.0 | 1390 | **371.3** | **13660** |
| Griewank | **52.61** | 70.21 | **88.44** | **412.2** | 865.1 | **1119** | 711.4 | 884.4 | **2643** |
| Shekel | **5.05** | 99.04 | **113.6** | **8.00** | 944.0 | **4588** | 12.00 | 528.0 | **18143** |
| Rosenbrock | **84.86** | 87.68 | **124.3** | **217.3** | 852.8 | **11017** | 1776 | – | **4580** |
| Michalewicz | **39.06** | 90.13 | **107.7** | **390.4** | 739.3 | **2794** | 1468 | **685.5** | **17334** |
| Dejong | **9.00** | 75.00 | **97.50** | **21.78** | 826.6 | **3056** | **23.76** | 826.9 | **5886** |
| Ackley | **44.49** | 102.0 | **114.4** | **638.0** | 805.7 | **875.0** | 918.3 | **617.0** | **4260** |
| Schwefel | **4.00** | 88.38 | **124.2** | **53.71** | 871.1 | **3675** | **141.7** | 685.5 | **13475** |
| Rastrigin | **33.00** | 74.48 | **99.45** | **191.4** | 792.2 | **3510** | **555.7** | 967.0 | **1010** |
| Raydan | **25.87** | 93.24 | **115.2** | **324.7** | 609.6 | **10063** | **726.9** | 774.0 | **4948** |

Table 4: Total number of objective function evaluations for the new method, the BBW and DH algorithms using one, two, and three-variable test functions.

algorithm, the number of objective function evaluations is close to the ones showed in the Table 4, but in a fraction of cases (smaller than 50%) we do not find the correct global minimum point. In this experiment, the new method uses the most efficient routine for each test function, information that is obtained from Tables 1, 2, and 3. For one and two-variable test functions, the advantage of the new method is remarkable. The advantage in the three-variable case is not so impressive, in contradiction to what we have concluded earlier, when we analyzed the performance graphs in Figs. 1 and 2. For the Neumaier, Michalewicz, and Ackley functions, the BBW algorithm is better than the new method. This seems to be a consequence of the small number of functions values in the discretization procedure ($N = 256^3$).

Table 5 shows the success probability of Algorithm 3 when we select the best optimization routine. The success probability is calculated in the following way: We run Algorithm 3 many times with the termination condition given by Eq. (2) and we count the number of times that the algorithm finds the correct global minimum. The success probability is the success rate. Table 5 is built using the data described in Appendix C, which shows the success probability of Algorithm 3 for all optimization routines. Notice that the success probabilities improve when we increase the number of variables. This shows that the termination condition given by Eq. (2) is sound. If the success probability for one variable (0.94) is not high enough for practical purposes, one can rerun the algorithm many times in order to improve this value. From Appendix C, we conclude that the BOBYQA routine is the best one on average for the testbed functions with one to three variables, while the SBPLX routine is the worst on average for one and two-variable functions and the NMEAD routine is the worst on average for three-variable functions. As an exception, the BOBYQA routine has a bad performance for the one-variable Rosenbrock function. Similar conclusions were drawn

| # vars.<br>fcn. | one-variable | two-variable | three-variable |
|---|---|---|---|
| Neumaier | 1.00 | 1.00 | 1.00 |
| Griewank | 0.87 | 1.00 | 1.00 |
| Shekel | 0.87 | 0.96 | 1.00 |
| Rosenbrock | 0.99 | 1.00 | 0.99 |
| Michalewicz | 1.00 | 1.00 | 0.99 |
| Dejong | 0.97 | 0.99 | 1.00 |
| Ackley | 1.00 | 1.00 | 1.00 |
| Schwefel | 0.98 | 1.00 | 1.00 |
| Rastrigin | 1.00 | 1.00 | 1.00 |
| Raydan | 1.00 | 1.00 | 1.00 |
| AVERAGE | 0.96 | 0.99 | 0.99 |

Table 5: Success probability of Algorithm 3 using the best classical optimization routine.

from Tables 1, 2, and 3. Those coincidences were expected since the experiments are correlated.

## 5   Conclusions

This paper proposed a new method for continuous global optimization problems, using GAS and classical routines to find efficiently a local minimum. Our numerical simulations show that DH, BBW, and the new method have very different asymptotic behavior, where the new method presented a better performance.

## Acknowledgments

## References

[1] M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. Fortschritte der Physik, 46:493–506, 1998.

[2] W.P. Baritompa, D.W. Bulger, and G.R. Wood. Grover's quantum algorithm applied to global optimization. SIAM Journal of Optimizaton, 15:1170–1184, 2005.

[3] C. Dürr and P. Høyer. A quantum algorithm for finding the minimum, http://lanl.arxiv.org/abs/quant-ph/9607014, 1999.

[4] C. Floudas and C. Gounaris. A review of recent advances in global optimization. Journal of Global Optimization, 45:3–38, 2009.

[5] L.K. Grover. Quantum Mechanics Helps in Searching for a Needle in a Haystack. Physical Review Letters, 79:325–328, 1997.

[6] L.A.B. Kowada, C. Lavor, R. Portugal, and C.H. Figueiredo. A new quantum algorithm to solve the minimum searching problem. International Journal of Quantum Information, 6:427–436, 2008.

[7] Y. Liu and G.J. Koehler. Using Modifications to Grover's Search Algorithm for Quantum Global Optimization. European Journal of Operational Research, 207;620-632, 2010.

[8] Y. Liu and G.J. Koehler. A Hybrid Method for Quantum Global Optimization. Journal of Global Optimization, 52:607–626, 2011.

[9] V. Protopopescu and J. Barhen. Solving a Class of Continuous Global Optimization Problems using Quantum Algorithms. Physics Letters A, 296:9–14, 2002.

[10] V. Protopopescu and J. Barhen. Quantum Algorithm for Continuous Global Optimization. Qi Liqun (ed.) et al., Optimization and control with applications. New York, Springer, 2005.

[11] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh V. Vazirani. Strengths and weaknesses of quantum computing. *SIAM J. Comput.*, 26(5):1510–1523, 1997.

[12] C. Zalka, Grover's quantum searching algorithm is optimal, *Phys. Rev. A*, 60(4):2746–2751, 1999.

[13] Hendrix E.M.T. and Klepper O. On uniform covering, adaptive random search and raspberries. *Journal of Global Optimization*, 18(2):143–163, 2000.

[14] Steven G. Johnson, The NLopt nonlinear-optimization package, http://ab-initio.mit.edu/nlopt.

[15] Jorge Nocedal. Updating Quasi-Newton Matrices with Limited Storage. *Mathematics of Computation*, 35(151):773–782, 1980.

[16] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Math. Program.*, 45(3):503–528, December 1989.

[17] Ron S. Dembo and Trond Steihaug. Truncated-newtono algorithms for large-scale unconstrained optimization. *Mathematical Programming*, 26:190–212, 1983.

[18] Krister Svanberg. A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM J. on Optimization*, 12(2):555–573, February 2002.

[19] M. J. D. Powell. Direct Search Algorithms for Optimization Calculations. *Acta Numerica*, 7:287–336, 1998.

[20] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7:308–313, 1965.

[21] Joel A. Richardson and J. L. Kuester. Algorithm 454: the complex method for constrained optimization [e4]. *Commun. ACM*, 16(8):487–489, August 1973.

[22] Thomas Harvey Rowan. *Functional stability analysis of numerical algorithms*. PhD thesis, Austin, TX, USA, 1990. UMI Order No. GAX90-31702.

[23] Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM J. Numer. Anal.*, 28(2):545–572, February 1991.

[24] E. G. Birgin and J. M. Martínez. Improving ultimate convergence of an augmented lagrangian method. *Optimization Methods Software*, 23(2):177–195, April 2008.

[25] M. J. D. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. August 2009.

# A    Test functions

Neumaier

$$f(x_0, \ldots, x_{n-1}) = \sum_{i=0}^{n-1} (x_i - 1)^2 - \sum_{i=1}^{n-1} x_i x_{i-1}, \ 0 \le x_i \le 4$$

Griewank

$$f(x_0, \ldots, x_{n-1}) = \frac{1}{4000} \sum_{i=0}^{n-1} x_i^2 - \prod_{i=0}^{n-1} \cos\left(\frac{x_i}{\sqrt{i+1}}\right) + 1, \ -40 \le x_i \le 40$$

Shekel

$$f(x_0, \ldots, x_{n-1}) = \sum_{i=0}^{m-1} \frac{1}{c_i + \sum_{j=0}^{n-1}(x_j - a_{ji})^2}, \ -1 \le x_i \le 1$$

Rosenbrock

$$f(x_0, \ldots, x_{n-1}) = \sum_{i=0}^{n-2} (1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2, \ -30 \le x_i \le 30$$

Michalewicz

$$f(x_0, \ldots, x_{n-1}) = -\sum_{i=0}^{n-1} \sin(x_i) \sin^{2m}\left(\frac{ix_i^2}{\pi}\right), \ 0 \le x_i \le 10$$

Dejong

$$f(x_0, \ldots, x_{n-1}) = \sum_{i=0}^{n-1} x_i^2, \ -5.12 \le x_i \le 5.12$$

Ackley

$$f(x_0, \ldots, x_{n-1}) = -20 \exp\left(-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=0}^{n-1} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=0}^{n-1} \cos(2\pi x_i)\right) + 20 + \exp(1), \ -15 \le x_i \le 20$$

Schwefel

$$f(x_0, \ldots, x_{n-1}) = -\sum_{i=0}^{n-1} x_i \sin\left(\sqrt{|x_i|}\right), \ -20 \le x_i \le 20$$

Rastrigin

$$f(x_0, \ldots, x_{n-1}) = \sum_{i=0}^{n-1} \left(x_i^2 - 10\cos(2\pi x_i) + 10\right), \ -5.12 \le x_i \le 5.12$$

Raydan

$$f(x_0, \ldots, x_{n-1}) = -\sum_{i=0}^{n-1} \frac{(i+1)}{10}\left(\exp(x_i) - x_i\right), \ -5.12 \le x_i \le 5.12$$

# B   Standard Deviation

This Appendix shows the tables of the standard deviation of the number of evaluations for one, two, and three-variable test functions associated with Tables 1, 2, and 3, respectively. In all tables, the smallest standard deviations are depicted in blue and largest in red.

| rout. \ fcn. | LBFGS* | TNEWT* | MMA* | COBYLA | NMEAD | SBPLX | AGL* | BOBYQA | CCSAQ* |
|---|---|---|---|---|---|---|---|---|---|
| Neumaier | **0.00** | 3.55 | 0.00 | 0.00 | 150.4 | **210.3** | 0.00 | 0.00 | 3.27 |
| Griewank | 22.05 | 38.36 | 40.63 | 90.67 | **212.7** | 181.5 | 37.47 | **21.55** | 41.86 |
| Shekel | **0.00** | 0.00 | 3.27 | **9.28** | 2.82 | 2.60 | 3.37 | 0.00 | 0.00 |
| Rosenbrock | **35.08** | 45.33 | 52.07 | 96.28 | **190.4** | 146.1 | 55.62 | 52.44 | 50.81 |
| Michalewicz | 29.42 | 39.70 | 35.52 | 107.4 | **136.3** | 103.4 | 36.53 | **23.93** | 38.09 |
| Dejong | **0.00** | 0.00 | 3.27 | 62.03 | 128.4 | **139.2** | 3.37 | 0.00 | 0.00 |
| Ackley | **48.83** | 54.81 | 54.75 | 91.30 | **182.4** | 171.6 | 54.63 | 50.23 | 52.63 |
| Schwefel | **0.00** | 0.00 | **26.05** | 13.99 | 0.00 | 24.04 | 0.00 | 8.44 | 0.00 |
| Rastrigin | 28.23 | 32.11 | **0.00** | 40.89 | **216.2** | 207.0 | 34.35 | 17.76 | 34.00 |
| Raydan | 5.88 | 21.42 | **0.00** | 27.22 | **177.9** | 143.3 | 0.00 | 34.48 | 15.96 |

Table 6: Standard deviation for one-variable test functions.

| rout. \ fcn. | LBFGS* | TNEWT* | MMA* | COBYLA | NMEAD | SBPLX | AGL* | BOBYQA | CCSAQ* |
|---|---|---|---|---|---|---|---|---|---|
| Neumaier | 1240 | 3163 | 849.5 | **0.00** | **9011** | 8935 | 712.7 | 452.5 | 602.9 |
| Griewank | 2757 | 1394 | 633.2 | 475.0 | 9369 | **10116** | 531.5 | **271.4** | 412.8 |
| Shekel | **2736** | 2692 | 1076 | **0.00** | 0.00 | 3.17 | 1305 | 2.18 | 1531 |
| Rosenbrock | **307.4** | 751.4 | 1258 | 6865 | **8471** | 7946 | 883.1 | 1713 | 714.2 |
| Michalewicz | 1817 | 2190 | 900.0 | 3552 | **9537** | 9376 | 670.2 | **615.7** | 658.8 |
| Dejong | 2625 | 917.5 | 909.1 | **0.00** | 7359 | **7635** | 1042 | 2.18 | 817.4 |
| Ackley | 1693 | 1778 | 1017 | 4867 | **7990** | 6736 | 1177 | **918.7** | 2156 |
| Schwefel | 4390 | 2498 | 1041 | 2038 | **23.94** | **4840** | 1030 | 182.6 | 851.1 |
| Rastrigin | 1006 | 825.5 | 856.9 | 1567 | **8918** | 6952 | 922.0 | **110.3** | 679.0 |
| Raydan | 2823 | 2181 | 791.9 | **493.0** | **9429** | 8425 | 1068 | 535.3 | 1336 |

Table 7: Standard deviation for two-variable test functions.

| rout. \ fcn. | LBFGS* | TNEWT* | MMA* | COBYLA | NMEAD | SBPLX | AGL* | BOBYQA | CCSAQ* |
|---|---|---|---|---|---|---|---|---|---|
| Neumaier | **1421** | 5056 | 3018 | 2847 | **8923** | 5652 | 3628 | 2173 | 2425 |
| Griewank | 3866 | 2720 | 801.1 | 1460 | **15369** | 15037 | 4247 | 699.2 | **695.4** |
| Shekel | 6247 | **7481** | 1599 | **0.00** | 0.00 | 0.00 | 843.6 | 0.00 | 1154 |
| Rosenbrock | 2196 | **1306** | 1390 | 3521 | **17103** | 15874 | 1806 | 3026 | 1931 |
| Michalewicz | 5928 | 2770 | – | 4859 | **13681** | 8277 | 2209 | 4610 | **1468** |
| Dejong | 5084 | 3581 | 1082 | 843.7 | **14373** | 9545 | 1241 | **2.38** | 857.6 |
| Ackley | – | 1299 | 1674 | 3644 | **15637** | 14966 | 1797 | 1654 | **1026** |
| Schwefel | 8141 | **9863** | 557.0 | **343.5** | 2164 | 645.1 | 550.4 | 870.5 | 1155 |
| Rastrigin | 2024 | 785.8 | 1266 | 1727 | 12457 | **12655** | 868.4 | **636.2** | 798.6 |
| Raydan | 4042 | 3778 | 1325 | **778.6** | 15591 | **16436** | 1393 | 877.8 | 1058 |

Table 8: Standard deviation for three-variable test functions.

# C  Success Probability

This Appendix shows the tables of success probability of Algorithm 3 with the termination condition given by Eq. (2) for one, two, and three-variable test functions using the classical optimization routines. In all tables, the largest probability are depicted in blue and lowest in red. We have performed an average over 100 rounds for each table.

| rout. / fcn. | LBFGS* | TNEWT* | MMA* | COBYLA | NMEAD | SBPLX | AGL* | BOBYQA | CCSAQ* |
|---|---|---|---|---|---|---|---|---|---|
| Neumaier | **1.00** | **1.00** | **1.00** | 0.37 | **0.10** | 0.12 | **1.00** | **1.00** | **1.00** |
| Griewank | 0.85 | 0.59 | 0.42 | 0.83 | 0.06 | **0.04** | 0.42 | **0.87** | 0.40 |
| Shekel | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** |
| Rosenbrock | **0.87** | 0.50 | 0.25 | 0.44 | 0.13 | **0.10** | 0.25 | 0.27 | 0.15 |
| Michalewicz | 0.82 | 0.46 | 0.68 | 0.21 | **0.08** | 0.08 | 0.68 | **0.99** | 0.58 |
| Dejong | **1.00** | **1.00** | **1.00** | 0.63 | 0.22 | **0.16** | **1.00** | **1.00** | **1.00** |
| Ackley | 0.57 | 0.44 | 0.32 | 0.23 | 0.13 | **0.10** | 0.32 | **0.97** | 0.34 |
| Schwefel | 0.91 | 0.84 | **0.79** | **1.00** | **1.00** | **1.00** | 0.79 | **1.00** | 0.88 |
| Rastrigin | 0.52 | 0.53 | 0.76 | 0.51 | 0.03 | **0.02** | 0.76 | **0.98** | 0.59 |
| Raydan | **1.00** | **1.00** | **1.00** | 0.87 | 0.11 | **0.08** | **1.00** | **1.00** | **1.00** |
| AVERAGE | 0.854 | 0.736 | 0.722 | 0.609 | 0.286 | **0.27** | 0.722 | **0.908** | 0.694 |

Table 9: Success probability for one-variable test functions.

| rout. / fcn. | LBFGS* | TNEWT* | MMA* | COBYLA | NMEAD | SBPLX | AGL* | BOBYQA | CCSAQ* |
|---|---|---|---|---|---|---|---|---|---|
| Neumaier | 0.96 | 0.98 | 0.92 | 0.86 | 0.37 | **0.28** | 0.93 | **1.00** | 0.98 |
| Griewank | **1.00** | 0.94 | 0.81 | **1.00** | **0.39** | 0.39 | 0.84 | **1.00** | 0.86 |
| Shekel | 0.95 | 0.94 | 0.74 | **1.00** | **1.00** | **1.00** | **0.70** | **1.00** | 0.77 |
| Rosenbrock | 0.90 | 0.93 | 0.95 | 0.65 | 0.35 | **0.30** | 0.96 | 0.95 | 0.95 |
| Michalewicz | 0.96 | 0.97 | 0.87 | 0.98 | 0.42 | **0.36** | 0.86 | **1.00** | 0.90 |
| Dejong | 0.99 | 0.97 | 0.93 | **1.00** | 0.72 | **0.52** | 0.93 | **1.00** | 0.93 |
| Ackley | 0.71 | 0.46 | 0.68 | **0.99** | **0.33** | 0.37 | 0.79 | **0.99** | 0.81 |
| Schwefel | 0.97 | 0.95 | 0.85 | **1.00** | **1.00** | 0.94 | 0.87 | **1.00** | **0.84** |
| Rastrigin | 0.94 | 0.78 | 0.88 | **1.00** | 0.51 | **0.41** | 0.86 | **1.00** | 0.91 |
| Raydan | 0.99 | 0.99 | 0.94 | **1.00** | 0.34 | **0.29** | 0.90 | **1.00** | 0.94 |
| AVERAGE | 0.937 | 0.891 | 0.857 | 0.948 | 0.543 | **0.486** | 0.864 | **0.994** | 0.889 |

Table 10: Success probability for two-variable test functions.

| rout. / fcn. | LBFGS* | TNEWT* | MMA* | COBYLA | NMEAD | SBPLX | AGL* | BOBYQA | CCSAQ* |
|---|---|---|---|---|---|---|---|---|---|
| Neumaier | 0.94 | **0.80** | 0.88 | **1.00** | 0.96 | 0.91 | 0.89 | 0.98 | 0.90 |
| Griewank | 0.95 | 0.96 | 0.94 | **1.00** | **0.57** | 0.60 | 0.92 | **1.00** | 0.95 |
| Shekel | 0.95 | 0.92 | 0.93 | **1.00** | **1.00** | **1.00** | 0.92 | **1.00** | **0.91** |
| Rosenbrock | 0.69 | 0.75 | 0.85 | 0.72 | 0.43 | **0.40** | 0.89 | **0.99** | 0.88 |
| Michalewicz | 0.79 | 0.86 | 0.78 | 0.77 | **0.59** | 0.68 | 0.77 | **0.99** | 0.80 |
| Dejong | 0.99 | 0.99 | 0.97 | **1.00** | **0.84** | 0.86 | 0.99 | **1.00** | 0.92 |
| Ackley | 0.77 | 0.68 | 0.89 | 0.96 | **0.46** | 0.70 | 0.89 | **1.00** | 0.83 |
| Schwefel | 0.92 | 0.88 | 0.81 | 0.99 | **1.00** | 0.99 | **0.73** | **1.00** | 0.84 |
| Rastrigin | 0.78 | 0.87 | 0.95 | **1.00** | **0.59** | 0.64 | 0.94 | **1.00** | 0.96 |
| Raydan | 0.94 | 0.94 | 0.96 | 0.99 | **0.39** | 0.47 | 0.92 | **1.00** | 0.93 |
| AVERAGE | 0.872 | 0.865 | 0.896 | 0.943 | **0.683** | 0.725 | 0.886 | **0.996** | 0.892 |

Table 11: Success probability for three-variable test functions.