GROVER'S QUANTUM ALGORITHM APPLIED TO GLOBAL OPTIMISATION

W. P. Baritompa, D. W. Bulger & G. R. Wood

Department of Mathematics and Statistics
University of Canterbury
Private Bag 4800
Christchurch, New Zealand

Report Number: UCDMS2004/22 NOV 2004

GROVER'S QUANTUM ALGORITHM APPLIED TO GLOBAL OPTIMISATION*

W. P. BARITOMPA†, D. W. BULGER‡, AND G. R. WOOD §

Abstract. Grover's quantum computational search procedure can provide the basis for implementing adaptive global optimisation algorithms. A brief overview of the procedure is given and a framework called Grover Adaptive Search is set up. A method of Dürr and Høyer and one introduced by the authors fit into this framework and are compared.

Key words. discrete optimisation, global optimisation, Grover iterations, Markov chains, quantum computers, random search

AMS subject classifications. 90C30, 68Q99, 68Q25

1. Introduction.

This paper aims to provide the global optimisation community with some background knowledge of quantum computation, and to explore the importance of this topic for the future of global optimisation.

Quantum computing [7] holds great potential to increase the efficiency of stochastic global optimisation methods. Current estimates are that quantum computers are likely to be in commercial production within two or three decades. These devices will be in many respects similar to the computers of today, but will utilise circuitry capable of quantum coherence [7], enabling data to be manipulated in entirely new ways.

Grover introduced in [9] a quantum algorithm (that is, an algorithm to be executed on a quantum computer) for locating a "marked" item in a database. This was extended in [2] to a quantum algorithm for locating one of an unknown number of marked items. The latter method was incorporated into a minimisation algorithm by Dürr and Høyer in [8] (unpublished, but available electronically—see the reference list).

Dürr and Høyer's algorithm can be viewed as an example of Grover adaptive search (GAS), an algorithmic framework we introduced in [4]. GAS in turn is a quantum-computational implementation of hesitant adaptive search [6], a parameterised pseudoalgorithm whose performance is well understood. Here we analyse Dürr and Høyer's method, present another version of GAS, and explore the relative merits of the two methods via numerical simulation.

Outline. Section 2 presents the general optimisation problem and introduces some notation and terminology. Section 3 gives a brief overview of quantum computation and Grover's algorithm. Section 4 describes the GAS framework. Section 5 discusses the considerations involved in choosing the "rotation count sequence", the parameter distinguishing one GAS algorithm from another. Section 6 presents Dürr and Høyer's algorithm, extending and correcting the theoretical analysis in [8]. In Section 7, we present a refined version of GAS, and in Section 8 this version is compared to that of Dürr and Høyer by numerical simulation. Section 9 concludes the

*The authors would like to thank the Marsden Fund of the Royal Society of New Zealand for support of this research.

†Department of Mathematics and Statistics, University of Canterbury, Christchurch, New Zealand (b.baritompa@math.canterbury.ac.nz).

‡Department of Statistics, Macquarie University, NSW 2109, Australia (dbulger@efs.mq.edu.au).

§Department of Statistics, Macquarie University, NSW 2109, Australia (gwood@efs.mq.edu.au).

paper.

2. Optimisation problem. We consider the following finite global optimisation problem:

$$\begin{aligned} &\text{minimise } f(x) \\ &\text{subject to } x \in S \end{aligned}$$

where f is a real-valued function on a finite set S.

Throughout this paper we associate with the objective function f the following definitions. Let $N = |S|$, the cardinality of the finite set S. We will usually assume N to be a power of two. Let $\ell_1 < \cdots < \ell_K$ be the distinct objective function values in the range of f. Notice that there may be more than K points in S. Given the uniform probability measure μ on S, we let π be the range measure given by the stochastic vector $(\pi_1, \ldots, \pi_K)$ induced by f. That is, $\pi_j = |f^{-1}(\ell_j)|/N$ for $j = 1, 2, \ldots, K$. Let p_j denote $\sum_{i=1}^{j} \pi_i$, the probability that a random point has value of ℓ_j or less. In particular, $p_K = 1$. Corresponding to each function value is an *improving region*, that part of the domain having a strictly better value, and we call its measure under μ the *improving fraction* p. (Usually the specified function value will be the best yet seen, and thus the improving region will be the set of points with objective function values better than any yet seen.)

3. Quantum computing. This paper concerns optimisation algorithms that require the use of a quantum computer. The characteristic feature of a quantum computer is that, in place of conventional computer *bits*, *quantum bits* or *qubits* are used. A qubit can be in a simultaneous superposition of "off" and "on" and thus allows *quantum parallelism*, where a single quantum circuit can simultaneously perform a calculation on a superposed input, corresponding to very many conventional inputs.

The Grover mechanism. The quantum procedure germane to our purposes is Grover Search [9]. This is one of the major advances to date in the fledgling field of quantum computation. More details are given in [4], but we reiterate the salient features here and give an intuitive discussion.

Consider the following general search problem. Let n be a positive integer, and let $S = \{0,1\}^n$, so that the domain size $N = 2^n$. Let $h : S \to \{0,1\}$. We wish to find a point $u \in S$ such that $h(u) = 1$. We further assume that h is a black-box, that is, that knowledge of h can only be gained by sampling (evaluation), but no structural information is available.

With conventional computing, the Boolean function h could be implemented as a subroutine, i.e., a conventional logic circuit constructed to take an input string of bits, representing a point of S, and output the associated bit value of h. The subroutine could then be applied to all points of S, in succession, to find a required point. Such a conventional program would require on average $N/2$ evaluations to find a marked point.

In quantum computing, the circuit implementing h (using gates that work with qubits) inputs and outputs *superpositions*. Thus it "sees" many possible answers at once. On a quantum computer, observing the output will collapse it into a conventional bit string, according to a probability distribution determined by the superposition; thus quantum computing has a stochastic side. Rather than loop through the N points in S, a quantum computer can operate on superposed states in such a way that the probablity distribution governing the collapse can be changed. Grover in [9]

showed if exactly one point is marked, then only $\frac{\pi}{4}\sqrt{N}$ such operations are required to find the marked point.

Denote the set of *marked* points by $M = \{u \in S | h(u) = 1\}$ and denote the number of these marked target points by t. We may or may not be aware of the value of t. Let p be the proportion of marked points, t/N.

Grover introduced the *Grover Rotation Operator*, which incorporates the oracle for h and provides a means of implementing a certain phase-space rotation of the states of a quantum system encoding points in the domain S. Repeated applications of this rotation can be used to move from the equal amplitude state, which is simple to prepare within a quantum computer, toward the states encoding the unknown marked points. For details see [4, 2, 9].

A *Grover search of r rotations* applies the above rotation operator r times, starting from the equal amplitude superposition of states, and then observes (and hence collapses to a point) the output state. The mathematical details in [2] show that executing such a search of r rotations generates domain points according to the following probability distribution γ on S:

$$\gamma(\{x\}) = \begin{cases} \frac{g_r(p)}{t}, & x \in M, \\ \frac{1-g_r(p)}{N-t}, & x \in S \backslash M, \end{cases} \tag{3.1}$$

where

$$g_r(p) = \sin^2\left[(2r+1)\arcsin\sqrt{p}\right]. \tag{3.2}$$

Note that, in the special case of $r = 0$, Grover search only observes the prepared equal amplitude superposition of states and so reduces to choosing a point uniformly from the domain.

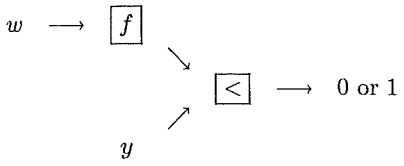
Most of the work in implementing the Grover Rotation Operator is in the oracle query, so the cost of a Grover search of r rotations is taken as the cost of r oracle queries. The output is a point in S, and as one would usually want to know if it is in M or not, a further oracle query (acting on the point) would give the function value under h.

Grover Search is sometimes portrayed as a method for the database table lookup problem. This is only one elementary application, however. Other interesting applications concern "marking functions" h which are more than simple tests of indexed data. Examples relating to data encryption and the satisfiability problem are given in [2, 9].

From searching to optimising. Grover Search solves a special global optimisation problem: it finds a global maximum of h. For the more general problem introduced in Section 2, our intention is to use Grover search repeatedly within a global optimisation method of the *adaptive search* variety. Adaptive search methods produce, or attempt to produce, an improving sequence of samples, each uniformly distributed in the improving region of the previous sample (see [16, 15, 5]).

Given an objective function $f : S \to \mathbb{R}$ and a point $X \in S$ with $f(X) = Y$, we use Grover's algorithm to seek a point in the improving region $\{w \in S : f(w) < Y\}$. As described above, Grover's algorithm requires an oracle, a quantum circuit able to classify a point $w \in S$ as inside or outside the target set (see [10]). This will be the oracle for the Boolean function $h(w) = (f(w) < Y)$.

We denote by a "boxed" name the oracle for a given function. Symbolically $\boxed{h}$ is found as shown:



The additional comparison logic circuitry $\boxed{<}$ to construct $\boxed{\text{h}}$ is minimal, and we will take the cost of $\boxed{h}$ and $\boxed{f}$ to be the same.

As far as Grover's algorithm is concerned, $\boxed{h}$ is simply a black box quantum circuit, inputting a point w in S (or a superposition of such points), and outputting

$$\begin{cases} 1, & f(w) < y, \\ 0, & f(w) \geq y \end{cases}$$

(or the appropriate superposition of such bits).

Grover search of r rotations, using the compound oracle depicted above, will require r uses of the objective function sub-oracle $\boxed{f}$, and will output a random domain point. An additional oracle query is required to determine whether the output is an improvement or not. Therefore, for practical purposes, we can consider the cost of running Grover's algorithm to be $r+1$ objective function evaluations (plus additional costs, such as the cost of the comparisons, which we will ignore).

As a point of departure for the mathematics to follow, we can condense this subsection into the following axiom, and henceforth dispense with any direct consideration of quantum engineering. (Note that the content of this axiom is taken for granted in [2] and many other recent publications on quantum searching.)

AXIOM 1. Given $f : S \to \mathbb{R}$ and $Y \in \mathbb{R}$, there is a search procedure on a quantum computer, which we shall call a "Grover search of r rotations on f with threshold Y", outputting a random point $x \in S$ distributed uniformly in

$$\begin{array}{ll} \{w \in S : f(w) < Y\} & \text{with probability } g_r(p), \text{ or uniformly in} \\ \{w \in S : f(w) \geq Y\} & \text{otherwise,} \end{array}$$

where $p = |\{w \in S : f(w) < Y\}|/|S|$. The procedure also outputs $y = f(x)$. The cost of the procedure is $r+1$ objective function evaluations.

4. Grover Adaptive Search. This section presents the Grover adaptive search (GAS) algorithm introduced in [4]. The algorithm requires as a parameter a sequence $(r_n : n = 1, 2, \ldots)$ of *rotation counts*. Initially, the algorithm chooses a sample uniformly from the domain and evaluates the objective function at that point. At each subsequent iteration, the algorithm samples the objective function at a point determined by a Grover search. The Grover search uses the best function value yet seen as a threshold. Here is the algorithm in pseudocode form:

Grover Adaptive Search (GAS)

1. Generate X_1 uniformly in S, and set $Y_1 = f(X_1)$.
2. For $n = 1, 2, \ldots$ until a termination condition is met, do:
   (a) Perform a Grover search of r_n rotations on f with threshold Y_n, and denote the outputs by x and y.
   (b) If $y < Y_n$, set $X_{n+1} = x$ and $Y_{n+1} = y$, otherwise, set $X_{n+1} = X_n$ and $Y_{n+1} = Y_n$.

GAS fits into the adaptive search framework developed in [5, 6, 15, 16, 17] which has proved useful for theoretical studies of convergence of stochastic global optimisation methods. All adaptive algorithms assume "improving" points can be found (at some cost). If Grover's algorithm were only applicable to database lookup, one might get the impression that GAS would require all function values to be first computed and tabled, before they could then be marked. However, Grover's algorithm can find points in an unknown target set, specified by an oracle. GAS exploits this ability by constructing, at each iteration, an oracle targeting the current improving region. In this way, it builds a sequence of domain points, each uniformly distributed in the improving region of the previous point. Such a sequence is known to converge to the global optimum very quickly; for instance, a unique optimum in a domain of size N will be found after $1 + \ln N$ such improvements, in expectation (see [17]).

Unfortunately this does not mean that GAS can find the global optimum for a cost in proportion to $\ln N$. The reason is that as the improving fraction p decreases, larger rotation counts become necessary to make improvement probable; thus the cost of GAS varies super-linearly in the number of improvements required. Note also that not every iteration finds a point in the improving region. The probability of finding an improvement is given by Equation (3.2), and for a known $p \ll 1$, a rotation count r can be found making this probability very nearly 1. But since in general we can only guess at p, lower probabilities result.

Readers may wonder why we use the best value yet seen as the threshold in the Grover search. In a sense, all of the work of the algorithm is done in the last step, when a Grover search is performed using a threshold only a little larger than the global minimum. This final Grover search is not made any easier by the information gained in earlier steps. In the general case, however, where we have no prior knowledge of the objective function's range, these earlier steps are an efficient way of finding a good value to use as a threshold in the final step. The earlier steps are not great in number. Moreover, the cost of each step is roughly inversely proportional to the square root of the improving fraction; thus, if the sequence of rotation counts is chosen suitably, most of the earlier steps will be much quicker than the final one.

5. Choosing the rotation count sequence. This section provides a general discussion of the selection of the rotation count sequence used in the GAS algorithm, as a precursor to Sections 6 and 7, each of which presents a specific selection method.

Why the rotation count should vary. In [4] we considered the possibility of using the same rotation count at each iteration. Although it is easy to construct objective functions for which this method works well, they are exceptional, and in general it is preferable to vary the rotation count as the algorithm progresses.

To see why, suppose that at a certain point in the execution of the GAS algorithm, the best value seen so far is Y, and the improving fraction is $p = |\{w : f(w) < Y\}|/N$. For any given rotation count r, the probability of success of each single iteration of the

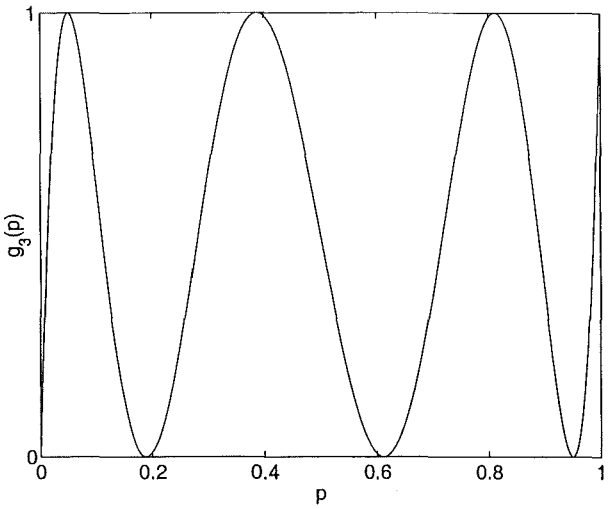


FIGURE 5.1. *The probability of a step of three Grover rotations finding an improvement, as a function of the improving fraction p.*

algorithm is given by $g_r(p)$. Although the rationale for using Grover's algorithm is to increase the probability of finding improving points, there are combinations of values of r and p where the opposite effect occurs. For instance, Figure 5.1 plots $g_3(p)$ versus p. If $p = 0.2$, then the step is almost guaranteed *not* to find an improvement. If the rotation count varies from each iteration to the next, then this is only an occasional nuisance. But if it is fixed at r, and if the algorithm should happen to sample a point x such that the improving fraction p for $Y = f(x)$ has $g_r(p)$ zero or very small, then the algorithm will become trapped.

How the rotation count should vary. In fact, at each iteration during the execution of the algorithm, some optimal rotation count r is associated with the improving fraction p of the domain (assuming $p > 0$). If it is used for the next Grover search, then an improving point will almost certainly be found. This r is the first positive solution to the equation $g_r(p) = 1$. (Actually of course we must round this to the nearest integer, and therefore success is not absolutely guaranteed, but this would contribute little to the expected cost of the algorithm.)

Unfortunately, in the general case the improving fraction p is unknown, so we are somewhat in the dark as to the choice of rotation counts. In order to make the most use of all the information available to us at each iteration, we could take a Bayesian approach, and keep track of a sequence of posterior distributions of the improving fraction at each iteration, and choose each rotation count to optimise the change in some statistic of this posterior distribution. As might be expected, this kind of approach appears to be very complex and unwieldy. The methods outlined in the following two sections, however, strike a happy balance between implementability and optimality of rotation count selection.

6. Dürr and Høyer's random method. In this section we outline a method due to Dürr and Høyer for randomly choosing rotation counts and correct two key

arguments in its originators' analysis.

Grover's search algorithm provides a method of finding a point within a subset of a domain. If the size of the target subset is known, the algorithm's rotation count parameter can easily be tuned to give a negligible failure probability. The case of a target subset of *unknown* size is considered in [2], where the following algorithm is presented:

Boyer et al. search algorithm

1. Initialise $m = 1$.
2. Choose a value for the parameter λ (8/7 is suggested in [2]).
3. Repeat:
   (a) Choose an integer j uniformly at random such that $0 \leq j < m$.
   (b) Apply Grover's algorithm with j rotations, giving outcome i.
   (c) If i is a target point, terminate.
   (d) Set $m = \lambda m$.

Actually, in [2], the final step updates m to $\min\{\lambda m, \sqrt{N}\}$. It *is* pointless to allow m to exceed $\sqrt{N}$, because for a target set of *any* size, it is known [2] that the optimal rotation count will be no more than $\lceil \pi\sqrt{N}/4 \rceil$. In the global optimisation context, however, this point will usually be immaterial, since the target region, though comprising a small proportion of the domain, will normally be large in absolute terms.

For instance, suppose the domain contains 10^{20} elements and suppose finding one of the smallest 10000 points is required. The optimal rotation count to find a target set of this size is $10^8\pi/4$, substantially less than $\lceil \pi\sqrt{N}/4 \rceil$. The actual target size will be unknown, and therefore the actual optimal rotation count will be unknown. But when m reaches this magnitude, if not before, each step will have a substantial probability (on the order of 1/2) of finding a target point. Therefore, unless λ is very large, there will be negligible probability of m reaching $\sqrt{N} = 10^{10}$ before a target point is produced. For simplicity, therefore, in this article we ignore the $\sqrt{N}$ ceiling on the growth of m.

In the `quant-ph` internet archive, Dürr and Høyer [8] propose using the Boyer et al. algorithm as the nucleus of a minimisation algorithm. Their paper gives the impression that the algorithm is just for the database problem. They begin with "an unsorted table of N items each holding a value from an ordered set. The minimum searching problem is to find the index y such that $T[y]$ is minimum." Again we stress their algorithm fits in the GAS framework and is thus applicable to the general optimisation problem.

In their paper, they indicate that every item that is improving is explicity marked. However, this is a mistake as it is incompatible with their complexity analysis later in the paper. We describe a corrected version of their method using the terminology of this paper.

Dürr and Høyer's algorithm

1. Generate X_1 uniformly in S, and set $Y_1 = f(X_1)$.
2. Set $m = 1$.
3. Choose a value for the parameter λ (as in the previous algorithm).
4. For $n = 1, 2, \ldots$ until a termination condition is met, do:
   (a) Choose a random rotation count r_n uniformly distributed on $\{0, \ldots, \lceil m - 1 \rceil\}$.
   (b) Perform a Grover search of r_n rotations on f with threshold Y_n, and denote the outputs by x and y.
   (c) If $y < Y_n$, set $X_{n+1} = x$, $Y_{n+1} = y$, and $m = 1$; otherwise, set $X_{n+1} = X_n$, $Y_{n+1} = Y_n$, and $m = \lambda m$.

This is the special case of GAS arising when the rotation count r_n is chosen randomly from an integer interval which is initialised to $\{0\}$ at each improvement, but which grows exponentially to a maximum of $\{0, \ldots, \lceil \sqrt{N} - 1 \rceil\}$ between improvements.

The analysis of the algorithm reported in the archive [8] uses incorrect constants from a preprint of [2]. In our analysis that follows, we correct this by using the published version of [2]. Because the Boyer et al. algorithm underpins that of Dürr and Høyer, we begin with an analysis of the former algorithm. Theorem 3 in [2] is an order of magnitude result, but inspection of the proof implies that the expected time required by the Boyer et al. algorithm to find one of t marked items among a total of N items is bounded by $8\sqrt{N/t}$. This constant can be improved upon, though, as we shall see after the following theorem.

THEOREM 6.1. *The expected number of oracle queries required by the Boyer et al. algorithm with parameter λ to find and verify a point from a target subset of size t from a domain of size N is*

$$\sum_{j=0}^{\infty} \frac{\lceil \lambda^j \rceil}{2} \prod_{i=0}^{j-1} \left(\frac{1}{2} + \frac{\sin(4\theta \lceil \lambda^i \rceil)}{4 \lceil \lambda^i \rceil \sin(2\theta)} \right) \tag{6.1}$$

where $\theta = \arcsin(\sqrt{t/N})$

Proof. Conditioned on reaching iteration j, the expected number of oracle queries required at that iteration is $\lceil \lambda^j \rceil / 2$ (including the test of the output of Grover's algorithm for target subset membership.) The probability of reaching iteration j is a product of failure rates; the probability of the algorithm failing to terminate at iteration j, having reached this iteration, is

$$\frac{1}{2} + \frac{\sin(4\theta \lceil \lambda^i \rceil)}{4 \lceil \lambda^i \rceil \sin(2\theta)}$$

(this is Lemma 2 in [2]). Thus the expected number of oracle queries required at iteration j, *not* conditioned on whether the iteration is reached, is

$$\frac{\lceil \lambda^j \rceil}{2} \prod_{i=0}^{j-1} \left(\frac{1}{2} + \frac{\sin(4\theta \lceil \lambda^i \rceil)}{4 \lceil \lambda^i \rceil \sin(2\theta)} \right),$$

and summing over all possible iterations $j = 0 \ldots \infty$ gives the result. □

It is straightforward to evaluate the geometrically convergent series (6.1) numerically. By graphing the ratio of (6.1) to $\sqrt{N/t}$ versus t for a range of λ, empirically

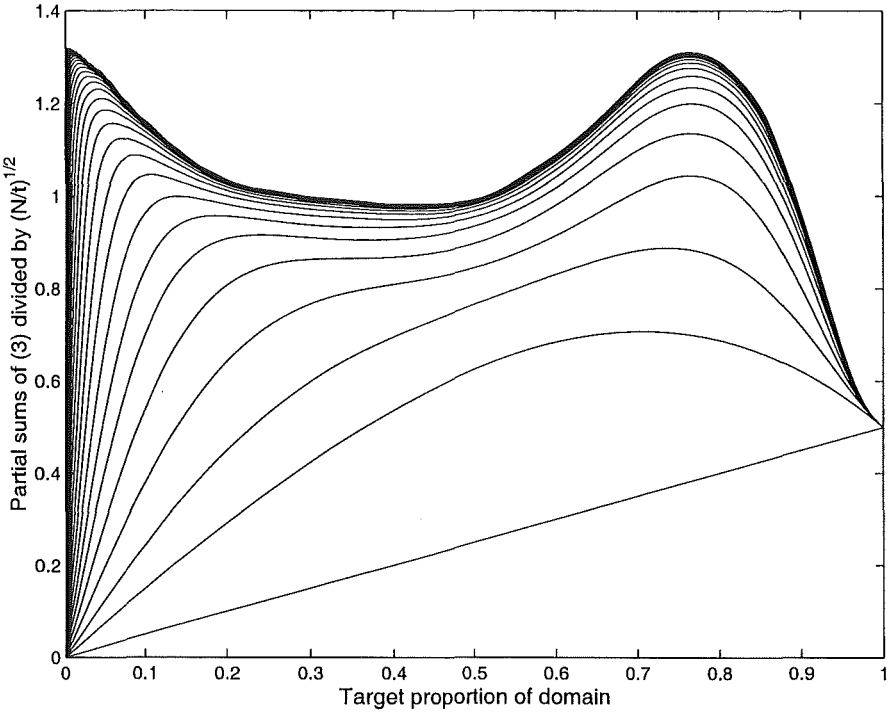


FIGURE 6.1. *The ratio between the partial sums of the geometrically convergent series (6.1) and* $\sqrt{N/t}$ *when* $\lambda = 1.34$*, plotted against* t/N*. Note that* 1.32 *appears to be an upper bound.*

λ that gave the lowest maximum is 1.34. The plot of Figure 6.1 uses this value of λ, and it justifies the following observation.

OBSERVATION 1. *The expected number of oracle queries required by the Boyer et al. algorithm with parameter* $\lambda = 1.34$ *to find and verify a point from a target subset of size* t *from a domain of size* N *is at most* $1.32\sqrt{N/t}$.

Now we can derive a performance bound for Dürr and Høyer's algorithm. This is similar to and extends the result in [8]; the main difference is in our treatment of the coefficient of the order bound. Also we correct another technical error in their argument, which is pointed out in our proof below.

THEOREM 6.2. *Assume the validity of the above observation. Let* $1 \leq s \leq N$ *and assume that there are* s *points in the domain with strictly better objective function values than the remaining* $N-s$ *points. The expected number of oracle queries required by Dürr and Høyer's algorithm with* $\lambda = 1.34$ *to find one of these* s *points is bounded above by*

$$1.32\sqrt{N} \sum_{r=s+1}^{N} \frac{1}{r\sqrt{r-1}}.$$

Note that, if s is small compared to N, then the above bound approximately equals $2.46\sqrt{N/s}$.

Proof. Assign the domain points ranks from 1 to N, giving the best point rank 1 and so forth. Where several points have equal objective function value, break ties arbitrarily, but let $l(r)$ be the least rank and $h(r)$ the greatest rank among the points with the same value as the rank r point. (In the distinct values case we will have $l(r) = h(r) = r$ for each $r \in \{1, \ldots, N\}$.)

Since Dürr and Høyer's algorithm will move through a succession of threshold values with rank above s before finding the desired target point, the bound on the expectation in question is given by

$$\sum_{r=s+1}^{N} p(N,r)B(N, l(r)-1), \tag{6.2}$$

where $p(N,r)$ is the probability of the rank r point ever being chosen and $B(N, l(r)-1)$ is the expected number of iterations required by the Boyer et al. algorithm to find and verify a point from a target subset of size $l(r)-1$.

The probability $p(N,r) = 1/h(r)$. This is demonstrated in the proof of Theorem 1 in [17], and in Lemma 1 of [8]. Also, by the observation on page 9, $B(N, l(r)-1) \leq 1.32\sqrt{N/(l(r)-1)}$.

In the distinct values case, substitution of the above value for $p(N,r)$ and bound for $B(N, l(r)-1) = B(N, r-1)$ into (6.2) gives the theorem immediately. In [8] it is claimed for the case of repeated objective function values that since the equation $p(N,r) = 1/r$ becomes the inequality $p(N,r) \leq 1/r$, the bound still holds. This argument ignores that the value of $B(N, l(r)-1)$ *increases* (for a given r) when repeated values are allowed. Nevertheless, the theorem holds as follows. Consider $\hat{r} \in \{1, \ldots, N\}$ with $l(\hat{r}) < h(\hat{r})$. We examine just that part of the summation in (6.2) with index going from $l(\hat{r})$ to $h(\hat{r})$.

$$\begin{aligned}
\sum_{r=l(\hat{r})}^{h(\hat{r})} p(N,r)B(N, l(r)-1) &\leq 1.32\sqrt{N} \sum_{r=l(\hat{r})}^{h(\hat{r})} \frac{1}{h(r)\sqrt{l(r)-1}} \\
&= 1.32\sqrt{N(l(\hat{r})-1)} \sum_{r=l(\hat{r})}^{h(\hat{r})} \frac{1}{h(\hat{r})(l(\hat{r})-1)} \\
&= 1.32\sqrt{N(l(\hat{r})-1)} \sum_{r=l(\hat{r})}^{h(\hat{r})} \frac{1}{r(r-1)} \\
&\leq 1.32\sqrt{N} \sum_{r=l(\hat{r})}^{h(\hat{r})} \frac{1}{r\sqrt{r-1}}. \qquad \square
\end{aligned}$$

REMARK 1. Dürr and Høyer's method can be viewed as an implementation of Pure Adaptive Search [17], requiring no more than $1.32(N/t)^{1/2}$ iterations in expectation to find an improvement, when t is the cardinality of the improving region.

7. A new method. In this section we propose an explicit sequence of integers to be used as the GAS rotation count sequence. This gives a special case of GAS that can be identified with an inhomogeneous Markov chain having states $\ell_1, \ldots, \ell_K$.

For this paper we have sought an efficient choice for the rotation count sequence used in GAS. This has led us to the special case of GAS arising when the sequence (r_n) is fixed in advanced, and determined by the following pseudocode. Note that the

sequence of rotation counts it produces is independent of the particular optimisation task; its first 33 entries are

$$0, 0, 0, 1, 1, 0, 1, 1, 2, 1, 2, 3, 1, 4, 5, 1, 6, 2, 7, 9, \\ 11, 13, 16, 5, 20, 24, 28, 34, 2, 41, 49, 4, 60, \ldots. \tag{7.1}$$

Here is the pseudocode:

Rotation Schedule Construction Algorithm

1. Initialise u to be the polynomial $u(y) = y$.
2. For $i = 1, 2, \ldots$, do:
   (a) Set $E_u = 1 - \int_0^1 u\, dy$.
   (b) Set $b' = 0$.
   (c) For $r = 0, 1, \ldots$ until $E_u/(r+1) \leq 2b'$, do:
      i. Set $v = u + y \int_y^1 (g_r(t)/t)\, du(t)$.
      ii. Set $E_v = 1 - \int_0^1 v\, dy$.
      iii. Set $b = (E_u - E_v)/(r+1)$.
      iv. If $b > b'$ then:
         A. Set $r' = r$.
         B. Set $b' = b$.
         C. Set $v' = v$.
   (d) Set $u = v'$.
   (e) Output ith rotation count r'.

The resulting sequence (7.1) is heuristically chosen to maximise a benefit-to-cost ratio, denoted b in the pseudocode, at each GAS iteration. The reader can verify that u and E_u are the cumulative distribution function and expectation, respectively, of the improving fraction of the domain, after the first $i-1$ iterations of the GAS algorithm. The symbols v and E_v denote the corresponding cumulative distribution function and expectation after a further GAS step of r rotations. The benefit is (somewhat arbitrarily) taken to be the expected decrease in the improving fraction of the domain, $E_u - E_v$. The cost is $r+1$, where r is the number of rotations chosen, as per the axiom on page 4. The inner loop at (2c) terminates since even if g_r were identically one, the expected improving region measure would halve. Thus, higher rotation counts need not be considered once we pass the point where half the expected improving region measure, divided by the cost, exceeds the current best found benefit-to-cost ratio.

8. Computational results. In Section 6 we presented a corrected version of Dürr and Høyer's demonstration of a performance bound for their algorithm. This readily establishes the $O(\sqrt{N/s})$ complexity, inherited from Grover's algorithm. However, even the improved coefficient of 2.46 suggested by Theorem 6.2 is based on an upper bound, and may be a poor indicator of the algorithm's actual performance. In this section we study the methods described in Sections 6 and 7 using numerical simulation. Our aim is twofold: to tune the parameter λ appearing in Dürr and Høyer's algorithm, and then to compare their tuned method against the method of Section 7.

Our simulations will determine the length of time each algorithm requires to sample a point in a target region, constituting a certain proportion of the domain. Intuitively, the algorithm terminates upon finding a value equal to or lower than the quantile determined by a proportion α.

Recall the proportion of the domain with value lower than or equal to ℓ_j is p_j. More precisely, we specify an *intended* quantile proportion α_{nominal} and set $k = \min\{j : p_j \geq \alpha_{\text{nominal}}\}$. We require the algorithm to find a point with value less than or equal to ℓ_k. The target set is $f^{-1}(\{\ell_1, \ell_2, \ldots, \ell_k\})$. Let s be its cardinality. So $\alpha = p_k = s/N$ and gives the quantile the algorithm will find. Note that it is the measure under π of $\{\ell_1, \ell_2, \ldots, \ell_k\}$. It may be inevitable that α and α_{nominal} differ since it can happen that $p_{k-1} < \alpha_{\text{nominal}} < \alpha = p_k$.

Thus the quantity α is often unknown in practice, and is a "global" piece of information. The dependence of performance on global information is unavoidable [14], but we will see that for certain methods, the dependence is primarily on α. For the rest of this paper we assume α is close to α_{nominal}.

Methodology. For the performance of either algorithm under consideration, the distribution of objective function values influences performance only via the range measure π. Our primary focus here will be the case where π is uniformly distributed over a finite set of distinct function values. Without loss of generality we can take this finite set to be $\{1, \ldots, K\}$. For example to explore seeking the best 1% of the domain under a uniform range distribution (i.e. $\alpha_{\text{nominal}} = 0.01$), using $K = 100$ will be fairly representative. At the end of this section we look briefly at other distributions.

To compare the algorithms, we plot their *performance graphs* [11] which relate practical computational effort to the probability of finding a point in the target set. The performance graph is simply the cumulative distribution function of the effort to success, defined as the number of objective function evaluations before a point in the target set is sampled. We compute these with Matlab, using standard techniques for Markov chains and stochastic processes.

Tuning λ. The observation on page 9 suggests the parameter choice $\lambda = 1.34$ for Dürr and Høyer's algorithm. Numerical experimentation agrees with this choice. Figure 8.1 shows the performance graphs, seeking 1% ($K = 100$) or 0.2% ($K = 500$) of the domain, of Dürr and Høyer's algorithm using a selection of values of λ ranging from 1.05 to 30, and including the values 8/7 and 1.34 suggested by [2] and Figure 6.1. Performance deteriorates slowly outside of the range from 1.34 to 1.44, but within that range there is no visible performance gradient. The value of λ may become more important for smaller values of α, but for the remainder of this section we shall use the value $\lambda = 1.34$.

Comparing the new method to Dürr and Høyer . Having settled on the parameter value $\lambda = 1.34$ for Dürr and Høyer's method, we can compare it to the method of Section 7. Figure 8.2 shows that, in the two cases studied, the new method dominates that of Dürr and Høyer. For instance, to sample a target comprising 0.2% of the domain with probability 90% or more, Dürr and Høyer's method requires more than 100 units of effort, whereas the new method requires only 79 (and in fact it then samples the target with probability 96%).

Note also, in the two situations depicted in Figure 8.2, the estimated bound of $2.46\sqrt{N/s}$ on the expected time required by Dürr and Høyer's algorithm, mentioned following Theorem 6.2, amounts to 24.6 and 55.0. While the true expectations cannot be computed from any finite portion of the performance graphs, these figures do appear visually to be in approximate agreement with the numerical results.

Nonuniform range distributions. Until now in this section we have assumed a uniform range distribution. This corresponds to the assumption of injectivity of the objective function, that is, that different points in the domain map to different values

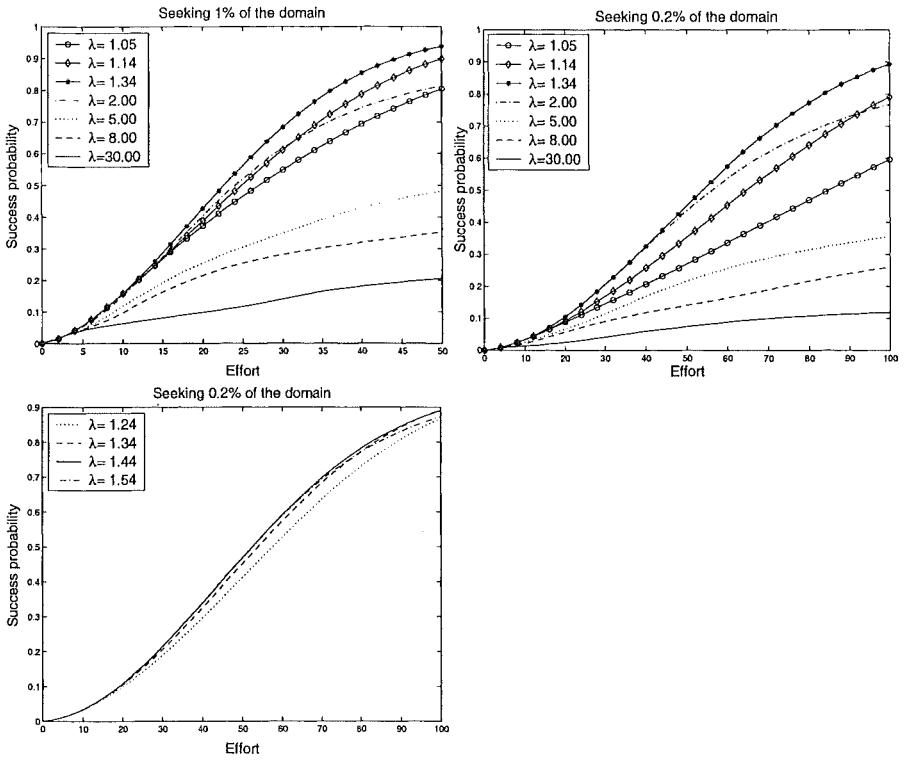


FIGURE 8.1. *Performance graphs for Dürr and Høyer's algorithm for various values of the parameter λ and two domain sizes. The third graph repeats the second with a finer mesh of λ values.*

in the range. In many cases, however, for instance in combinatorial optimisation,

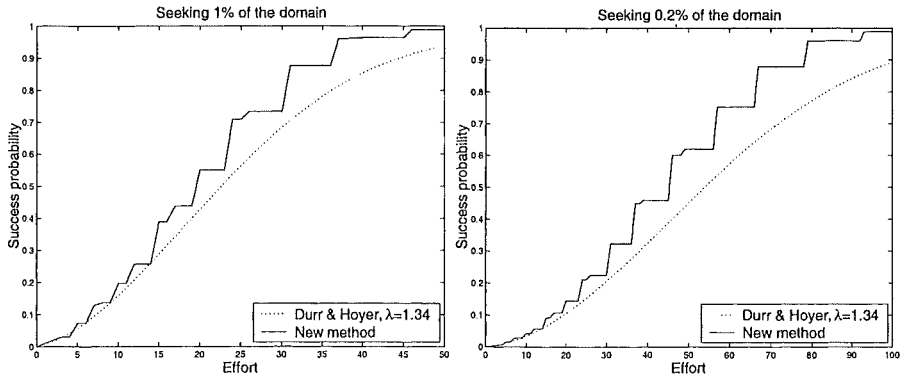


FIGURE 8.2. *Performance graphs comparing Dürr and Høyer's method to the method of Section 7, for a uniform range distribution.*

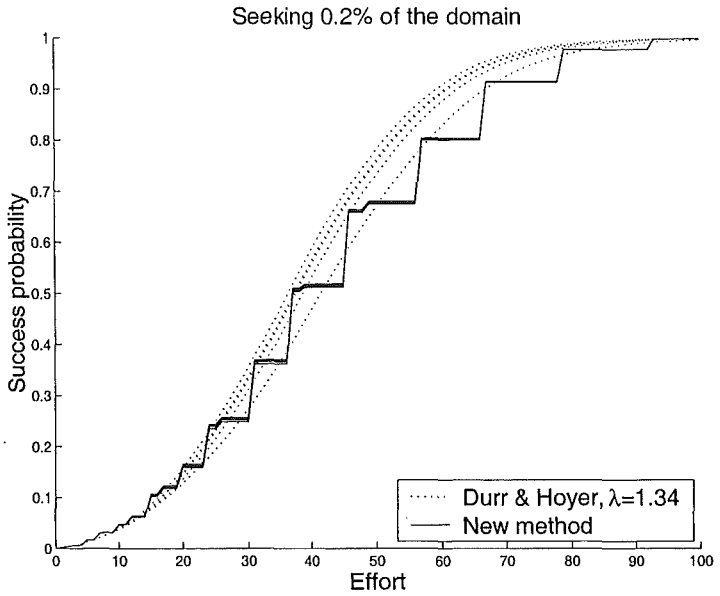


FIGURE 8.3. *Performance graphs comparing Dürr and Høyer's method to the method of Section 7, for a nonuniform range distribution.*

there may be a unique optimum, or a small number of optimal domain points, but large sets of the domain sharing values in the middle of the range; this results in a nonuniform range distribution.

Experimentation indicates that nonuniformity of the range distribution improves the performance of both methods under study. To produce Figure 8.3, we randomly created five stochastic vectors of length 20 with first element 0.002 (the remainder of each vector was a point uniformly distributed in $[0,1]^{19}$ and then scaled to sum to 0.998), and simulated the performance of both methods. Compare this with the last plot of Figure 8.2. Nonuniformity has improved the performance of the method of Section 7 somewhat. However, a greater improvement in Dürr and Høyer's method has allowed it to overtake the method of Section 7. Here, for most of the five sample range distributions, Dürr and Høyer's method reaches the target with probability 90% or more after 61 or fewer units of effort, whereas the new method now requires 67.

9. Conclusion. This paper outlines the significance of Grover's quantum search algorithm (with its performance characteristics implying $O(\sqrt{N/t})$ performance taken as an axiom) for global optimisation. Grover search can provide the basis of implementing adaptive global optimisation algorithms. One example is an algorithm of Dürr and Høyer's introduced as a method for finding minimum values in a database. An improved analysis of Dürr and Høyer's algorithm suggests increasing its parameter λ from 8/7 to 1.34. Also, that algorithm fits the Grover Adaptive Search framework, and thus is applicable to the more general global optimisation problem. A new algorithm within the same framework is proposed in Section 7. Our numerical experiments in Section 8 show that the algorithms have similar performance. The method proposed in Section 7 had its parameters tuned for the distinct objective function value

case, and shows superior performance to that of Dürr and Høyer's in that case. On the other hand, Dürr and Høyer's method (with $\lambda = 1.34$) overtakes the new method if there is a great deal of repetition in objective function values.

A final comment concerning implementation on a quantum computer. This is work mainly for computer engineers of the future, but some indications are known at the present time. A fully functional quantum computer would be able to evaluate an objective function in just the same way as a conventional computer, by executing compiled code. A technical requirement to control quantum coherence, which we have not mentioned previously, is that the gates must implement reversible operations. The code implementing the objective function must be run in the forward direction and then in the reverse direction. This obviously at most doubles the computational effort for a function evaluation compared to a conventional computer.

REFERENCES

[1] W. P. Baritompa, Handling different cost algorithms, Department of Mathematics and Statistics, University of Canterbury Report, preprint, (2001).

[2] M. Boyer, G. Brassard, P. Høyer and A. Tapp, Tight bounds on quantum searching, Fortschr. Phys., 46 (1998), pp. 493–506.

[3] S. H. Brooks, A discussion of random methods for seeking maxima, Oper. Res., 6 (1958), pp. 244–251.

[4] D. W. Bulger, W. P. Baritompa and G. R. Wood, Implementing pure adaptive search with Grover's quantum algorithm, J. Optim. Theory Appl., 116 (2003), pp. 517-529.

[5] D. W. Bulger, D. L. J. Alexander, W. P. Baritompa, G. R. Wood and Z. B. Zabinsky, Expected hitting time for backtracking adaptive search, Optimization , 53 (2004), pp. 189-202.

[6] D. W. Bulger and G. R. Wood, Hesitant adaptive search for global optimisation, Math. Program., 81 (1998), pp. 89–102.

[7] Centre for Quantum Computation, Oxford, `http://www.qubit.org`.

[8] C. Dürr and P. Høyer, A quantum algorithm for finding the minimum, `http://lanl.arxiv.org/abs/quant-ph/9607014`, version 2 (7 Jan 1999).

[9] L. K. Grover, A fast quantum mechanical algorithm for database search, Proceedings of the 28th Annual ACM Symposium on Theory of Computing, (1996).

[10] L. K. Grover, A framework for fast quantum mechanical algorithms, Proceedings of the 30th Annual ACM Symposium on Theory of Computing, (1998).

[11] E. M. T. Hendrix and O. Klepper, On uniform covering, adaptive random search and raspberries, J. Global Optim., 18 (2000), pp. 143–163.

[12] R. Laflamme, Los Alamos scientists make seven bit quantum leap, `http://www.lanl.gov/worldview/news/releases/archive/00-041.html` (2000).

[13] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J. Comput., 26 (1997), pp. 1484–1509.

[14] C. P. Stephens and W. P. Baritompa, Global optimization requires global information, J. Optim. Theory Appl., 96 (1998), pp. 575–588.

[15] G. R. Wood, Z. B. Zabinsky and B. P. Kristinsdottir, Hesitant adaptive search: the distribution of the number of iterations to convergence, Math. Program., 89 (2001), pp. 479–486.

[16] Z.B. Zabinsky and R.L. Smith Pure adaptive search in global optimization, Math. Program., 53 (1992), pp. 323–338.

[17] Z. B. Zabinsky, G. R. Wood, M. A. Steel and W. P. Baritompa, Pure adaptive search for finite global optimization, Math. Program., 69 (1995), pp. 443–448.

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