Reasoning about Grover's Quantum Search Algorithm using Probabilistic *wp* †

MICHAEL BUTLER and PIETER HARTEL
Department of Electronics & Computer Science
University of Southampton
Southampton SO17 1BJ
United Kingdom
{mjb,phh}@ecs.soton.ac.uk

Grover's search algorithm is designed to be executed on a quantum-mechanical computer. In this paper, the probabilistic *wp*-calculus is used to model and reason about Grover's algorithm. It is demonstrated that the calculus provides a rigorous programming notation for modelling this and other quantum algorithms and that it also provides a systematic framework of analysing such algorithms.

Categories and Subject Descriptors: F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity; F.3 [**Theory of Computation**]: Logics and Meanings of Programs; G.3 [**Mathematics of Computing**]: Probability and Statistics

General Terms: Algorithms, Theory, Verification

Additional Key Words and Phrases: Quantum mechanics, Quantum computation

1. INTRODUCTION

Quantum computers are a proposed means of using quantum-mechanical effects to achieve efficient computation. Quantum-mechanical systems may be in superpositions of several different states simultaneously. The central idea of quantum computers is to perform operations on these superposed states simultaneously and thus achieve a form of parallel computation. These devices were proposed in the early 1980's [Benioff 1980; Deutsch 1985].

One essential phenomenon of quantum mechanics is that the measurement of a superposed system forces it into a single classical state. Each superposed state is present with a certain amplitude and an observation causes it to collapse to that state with a probability that depends on its amplitude. This means that, although many computations may be performed in parallel on a quantum device, the result of only one of these may be observed. This may seem like a severe limitation, but several ingenious algorithms have been devised which work by increasing the amplitude of the desired outcome before any observation is performed and thus increasing the likelihood of the observed outcome being the desired one.

One such algorithm is Grover's quantum search algorithm [Grover 1997] which performs a search on an unstructured search space of size N in $\mathcal{O}(\sqrt{N})$ steps. To find the desired search value with 100% probability in such a space, a classical computer cannot do better than a linear time search. Grover's algorithm performs operations on a superposition

† Accepted for publication in *ACM Transactions in Programming Languages and Systems*.

of all possible search values that serve to increase the amplitude of the desired search value. Grover shows that within $\mathcal{O}(\sqrt{N})$ steps there is a greater than 50% chance of finding the desired search value. [Boyer et al. 1998] proved a stronger result for the algorithm showing that the correct search value can be found in $\mathcal{O}(\sqrt{N})$ with almost 100% probability.

In this paper, we apply the probabilistic weakest-precondition (*wp*) calculus of [Morgan et al. 1996] to Grover's algorithm to redevelop the result of [Boyer et al. 1998] in a more systematic way. The probabilistic *wp*-calculus is an extension of Dijkstra's standard *wp*-calculus [Dijkstra 1976] developed for reasoning about the correctness of imperative programs. The extension supports reasoning about programs containing probabilistic choice. The measurement of a quantum superposition is an example of a probabilistic choice.

Use of the probabilistic *wp*-calculus contributes two essential ingredients to the analysis of quantum algorithms. Firstly it provides an elegant and rigorous notation for describing quantum algorithms. The existing literature uses block diagrams and structured English which can be cumbersome and potentially ambiguous. Secondly, the probabilistic *wp*-calculus provides a set of rules for the systematic analysis of the correctness of algorithms. In the case of standard algorithms, the calculus is used to determine whether a program achieves some desired outcome. In the case of probabilistic algorithms, the calculus is used to reason about the probability of a program achieving some desired outcome.

This paper is not simply about re-presenting a known result about Grover's algorithm but it also aims to demonstrate that the probabilistic *wp*-calculus is suitable for both modelling and reasoning about a quantum algorithm. Boyer et al have already derived the same result that we derive here but they do so in a less systematic way. Our hope is that the approach used here could be applied fruitfully to other quantum algorithms and will aid the development of new quantum algorithms.

The paper is organised as follows. In Section 2, we give a sufficient overview of quantum theory. In Section 3, we present our approach to modelling quantum computation using the programming language of the probabilistic *wp*-calculus. In Section 4, we present Grover's algorithm using the approach of Section 3. In Section 5, we give a sufficient introduction to the rules of the probabilistic *wp*-calculus and in Section 6, we use the *wp*-calculus to derive a formula for the probability of success of Grover's algorithm.

2. QUANTUM SYSTEMS AND QUBITS

In quantum mechanics, a superposition of two states A and B may be represented in Dirac's notation as follows:

$$S \;=\; \alpha|A\rangle \;+\; \beta|B\rangle \;.$$

System S is said to be in a superposition of A and B. A and B are the *basis states* and α and β are *amplitudes*. The amplitudes may be complex numbers.

Let $\|z\|^2$ be the square norm[1] of complex number z. Observation of S will cause the system to collapse to state $|A\rangle$ with probability $\|\alpha\|^2$ and to $|B\rangle$ with probability $\|\beta\|^2$. The probabilities must sum to 1:

$$\|\alpha\|^2 \;+\; \|\beta\|^2 \;=\; 1 \;.$$

A *qubit* is a two-state quantum system in which the basis states are labelled 0 and 1:

$$S \;=\; \alpha|0\rangle \;+\; \beta|1\rangle \;.$$

[1] The square norm of any complex number $a + bi$ is $a^2 + b^2$.

A classical bit has $\alpha = 1$ and $\beta = 0$ or $\alpha = 0$ and $\beta = 1$.

A qubit evolves from one superposition to another using a quantum gate (or function) F:

$$F(\ \alpha|0\rangle\ +\ \beta|1\rangle\ )\ =\ \alpha'|0\rangle\ +\ \beta'|1\rangle\ .$$

F must be *unitary* which means that

—1-summing of probabilities is preserved: $\|\alpha'\|^2\ +\ \|\beta'\|^2\ =\ 1\ =\ \|\alpha\|^2\ +\ \|\beta\|^2$, and

—F has an inverse.

In quantum mechanics, a transformation F is usually modelled using matrix multiplication:

$$F(\ \alpha|0\rangle\ +\ \beta|1\rangle\ )\ =\ U_F \times \begin{pmatrix} \alpha \\ \beta \end{pmatrix},$$

where U_F is a 2×2 *unitary* matrix. Matrix U is unitary if $U.U^\dagger\ =\ U^\dagger.U\ =\ I$ where $U^\dagger$ is the conjugate transpose of U. It can be shown that such a transformation defined by unitary matrix U_F is unitary [Bernstein and Vazirani 1993].

A quantum superposition may have an arbitrary number of basis states, not just two. An N-state superposition is represented as:

$$S\ =\ \sum_{i=0}^{N-1} \alpha_i|i\rangle\ .$$

Observation of S will cause it to collapse to state $|i\rangle$ with probability $\|\alpha_i\|^2$. Again, the probabilities must sum to 1:

$$\sum_{i=0}^{N-1} \|\alpha_i\|^2\ =\ 1\ .$$

A *quantum register* is a collection of qubits and an L-qubit register gives rise to a system with 2^L basis states. Like qubits, quantum registers evolve under unitary transformations.

For further details on quantum computation, the reader is referred to papers such as [Berthiaume 1996; Deutsch and Ekert 1998].

3. MODELLING QUANTUM COMPUTERS

A quantum computer is a collection of quantum registers and quantum gates. In this section, we introduce ways of modelling various aspects of quantum computation using the programming language of the probabilistic *wp*-calculus. We use a subset of the language which includes standard assignment, probabilistic assignment, sequential composition and simple loops.

Firstly, we model an N-state quantum system as a function from state indices to complex numbers: $S : (0..N-1) \to \mathbb{C}$.

A superposition of the form

$$\sum_{i=0}^{N-1} \alpha_i|i\rangle$$

is modelled by the function S where for $0 \leq i < N$:

$$S(i) \;=\; \alpha_i \;.$$

A classical state i is modelled by the function which is zero everywhere except at i, which we write as $|i\rangle$:

$$\begin{aligned} |i\rangle(j) \;&=\; 1, \text{ if } i = j \\ &=\; 0, \text{ otherwise} \;. \end{aligned}$$

Transformation of a quantum state is modelled by a standard *assignment* statement:

$$S \;:=\; F(S) \;.$$

F must be unitary for this to be a valid quantum transformation.

We shall find it convenient to use lambda abstraction to represent transformations: $(\lambda i \mid 0 \leq i < N \cdot E)$ represents the function that takes an argument i in the range $0..N{-}1$ and returns the value E. For example, the unitary transformation that inverts the amplitude of each basis state is modelled as follows:

$$S \;:=\; (\; \lambda i \mid 0 \leq i < N \cdot -S(i) \;) \;.$$

Sequencing of transformations is modelled using *sequential composition*: if T_1 and T_2 are transformations, then their sequential composition is written $T_1 ; T_2$.

The loop which iterates C times over a transformation T is written $\mathbf{do}\; C \;\mathbf{times}\; T \;\mathbf{od}$.

We model the observation of a quantum system using a *probabilistic assignment* statement. In the simple case, this is a statement of the form:

$$\begin{aligned} x \;:=\; & E \;\;@\;\; p, \\ & F \;\;@\;\; (1-p) \;. \end{aligned}$$

This says that x takes the value E with probability p and the value F with probability $1-p$. For example, a coin flip is modelled by

$$\begin{aligned} coin \;:=\; & head \;\;@\;\; 0.5 \;, \\ & tail \;\;@\;\; 0.5 \;. \end{aligned}$$

Observation of a two-state superposition forces the system into a classical state. This is modelled with the following probabilistic assignment:

$$\begin{aligned} S \;:=\; & |0\rangle \;\;@\;\; \|S(0)\|^2, \\ & |1\rangle \;\;@\;\; \|S(1)\|^2 \;. \end{aligned}$$

A generalised probabilistic statement has the form

$$x \;:=\; E_i \;\;@\;\; p_i\,, \quad 0 \leq i < N,$$

where $(\sum_{i=0}^{N-1} p_i) = 1$.

Now observation of an N-state quantum system S may be modelled by

$$S \;:=\; |i\rangle \;\;@\;\; \|S(i)\|^2, \quad 0 \leq i < N \;.$$

That is, S collapses to the classical state i with probability $\|S(i)\|^2$.

$$S \;:=\; (\lambda i \mid 0 \leq i < N \cdot \tfrac{1}{\sqrt{N}})\;; \qquad \textit{Init}$$

$$\begin{array}{ll}
\textbf{do}\ C\ \textbf{times} & \\
\quad S \;:=\; (\lambda i \mid 0 \leq i < N \cdot S(i) - 2.f(i).S(i)\ )\;; & \\
\quad S \;:=\; (\lambda i \mid 0 \leq i < N \cdot 2.\overline{S} - S(i)\ ) & \textit{Body} \\
\textbf{od}\;; &
\end{array}$$

$$S \;:=\; |i\rangle \;@\; \|S(i)\|^2, \quad 0 \leq i < N \qquad \textit{Measure}$$

Fig. 1. Grover's search algorithm.

4. GROVER'S SEARCH ALGORITHM

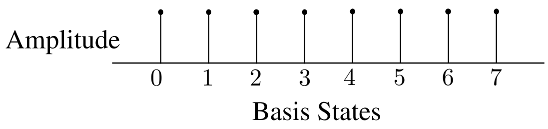
The Grover search problem may be stated as follows:

> Given a function $f : (0..N-1) \rightarrow \{0,1\}$ that is zero everywhere except for one argument x_0, where $f(x_0) = 1$, find that argument x_0 .

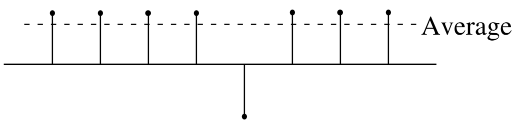
The algorithm makes use of the mean of a superposition S, written $\overline{S}$, where

$$\overline{S} \;=\; \frac{\sum_{i=0}^{N-1} S(i)}{N} \;.$$

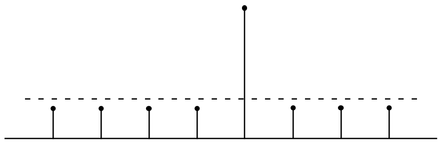
The algorithm is represented in the programming language of the probabilistic *wp*-calculus in Fig. 1. The initialisation of this algorithm sets the system S up in an equal superposition of all possible basis states. Details of how this distribution is achieved may be found in [Grover 1997]. Successive iterations of the loop then serve to increase the amplitude of the search argument x_0 while decreasing the amplitude of the other arguments. To see why this is so, consider the case of $N = 8$. The initialisation sets S up in an equal superposition of the eight possible states, represented diagrammatically as follows:



The first step of the loop body replaces each $S(i)$ with $S(i) - 2.f(i).S(i)$. This inverts $S(i)$ about the origin in the case that $f(i) = 1$ and leaves $S(i)$ unchanged in the case that $f(i) = 0$. Assuming that $f(4) = 1$, this replaces our example superposition with



The second step of the loop body inverts each amplitude about the average of all the amplitudes resulting in:



The amplitude of state $|4\rangle$ has increased as a result of the two steps of the loop body, while the amplitude of the others has decreased.

After an optimum number of iterations, C, the amplitude of $|x_0\rangle$ approaches 1 while the amplitude of the other states approaches 0. An observation is then performed. Since the amplitude of $|x_0\rangle$ approaches 1, the probability of the observation yielding $|x_0\rangle$ is close to 1. C depends on the number of states N and, as discussed in Section 6, it is $\mathcal{O}(\sqrt{N})$.

5. PROBABILISTIC *WP*

In two-valued logic, a *predicate* may be modelled as a function from some state space to the set $\{0,1\}$. For example, the predicate $x > y$ evaluates to 1 in a state in which x is greater then y and evaluates to 0 in any other state. A *probabilistic predicate* generalises the range to the continuous space between 0 and 1 [Kozen 1985; Morgan et al. 1996]. For example, the probabilistic predicate $0.5 \times (x > y)$ evaluates to 0.5 in a state in which x is greater then y and evaluates to 0 in any other state.

In the standard *wp*-calculus, the semantics of imperative programs is given using weakest-precondition formulae: for program $prog$ and postcondition $post$, $wp(prog, post)$ represents the weakest precondition (or maximal set of initial states) from which $prog$ is guaranteed to terminate and result in a state satisfying $post$. The *wp* rule for assignment is given by:

$$wp(\ x := E,\ post\ )\ =\ post[x := E]\ . \tag{1}$$

Here, $post[x := E]$ represents predicate $post$ with all free occurrences of x replaced by E. For example,

$$\begin{aligned} wp(x := 7,\ x > y) &= x > y\,[x := 7] \\ &= 7 > y\ . \end{aligned}$$

That is, the assignment $x := 7$ is guaranteed to establish $x > y$ provided $7 > y$ initially. The *wp* rule for sequential composition is given by:

$$wp(\ prog1; prog2,\ post\ )\ =\ wp(\ prog1,\ wp(prog2,\ post\ )\ )\ . \tag{2}$$

Both (1) and (2) also apply in the probabilistic *wp*-calculus. The *wp* rule for simple probabilistic assignment is given by:

$$\begin{aligned} & wp(\ x\ :=\ E\ @\ p,\ F\ @\ 1-p\ ,\ \ post\ )\ = \\ & \quad p \times post[x := E]\ +\ (1-p) \times post[x := F]\ . \end{aligned} \tag{3}$$

In the case of non-probabilistic $post$, $wp(prog, post)$ represents the probability that program $prog$ establishes $post$. For example

$$\begin{aligned} & wp(\ coin\ :=\ head\ @\ 0.5,\ tail\ @\ 0.5,\ \ coin = head\ ) \\ = \quad & \text{by (3)} \\ & 0.5 \times (coin = head\,[coin := head])\ +\ 0.5 \times (coin = head\,[coin := tail]) \end{aligned}$$

$$
\begin{aligned}
&= \quad \text{substitution} \\
&\quad 0.5 \times (head = head) \ + \ 0.5 \times (tail = head) \\
&= 0.5 \times 1 \ + \ 0.5 \times 0 \\
&= 0.5 \ .
\end{aligned}
$$

That is, a coin flip establishes $coin = head$ with probability 0.5 .

The *wp* rule for the generalised probabilistic assignment is given by:

$$wp(\ x \ := \ E_i \ @ \ p_i, \ 0 \le i < N, \ \ post \ ) \ = \ \sum_{i=0}^{N-1} p_i \times post[x := E_i] \ . \tag{4}$$

Our *wp*-rules for probabilistic assignment are based on [Morgan et al. 1996].

The only other programming construct we need in order to model Grover's algorithm is the DO-loop. Since the algorithm only loops a constant and finite number of times, we can model $\mathbf{do}\ C\ \mathbf{times}\ prog\ \mathbf{od}$ as a finite sequential composition of C copies of $prog$ which we write as $prog^C$. We have that

$$prog^0 \ = \ skip \tag{5}$$

$$prog^{i+1} \ = \ prog \ ; \ prog^i \ . \tag{6}$$

Here, $skip$ is the statement that does nothing, with $wp(skip, post) = post$. The semantics of more general looping constructs is given by least fixed points in the usual way [Kozen 1985; Morgan et al. 1996], but we do not need that here.

6. REASONING ABOUT GROVER

The postcondition we are interested in for the Grover algorithm is that the correct solution is found, i.e., $S = |x_0\rangle$. The probability that Grover establishes $S = |x_0\rangle$ is given by $wp(\ Grover, \ S = |x_0\rangle\ )$, so we shall calculate this.

The Grover algorithm has the following structure:

$$
\begin{array}{l}
Init \ ; \\
\mathbf{do}\ C\ \mathbf{times} \\
\quad Body \\
\mathbf{od} \ ; \\
Measure \ .
\end{array}
$$

which we shorten to

$$Init \ ; \ Body^C \ ; \ Measure \ .$$

When calculating a formula of the form $wp(\ prog1; prog2, \ post \ )$, we first calculate $wp(\ prog2, \ post \ )$ and then apply $wp(\ prog1, \ \_\ )$ to the result of this. Thus, to calculate $wp(\ Grover, \ S = |x_0\rangle\ )$, we first calculate $wp(\ Measure, \ S = |x_0\rangle\ )$:

$$
\begin{aligned}
&\quad wp(\ Measure, \ S = |x_0\rangle\ ) \\
&= wp(\ S := |i\rangle \ @ \ \|S(i)\|^2, \ 0 \le i < N, \quad S = |x_0\rangle\ ) \\
&= \quad \text{by (4)} \\
&\quad \sum_{i=0}^{N-1} \|S(i)\|^2 \times (|i\rangle = |x_0\rangle)
\end{aligned}
$$

$$
\begin{aligned}
= \quad & \text{since } (|i\rangle = |x_0\rangle) \text{ is 0 for } i \neq x_0 \\
& \|S(x_0)\|^2 \; . \qquad\qquad (7)
\end{aligned}
$$

Next we calculate $wp(\ Body^C\ ,\ \|S(x_0)\|^2\ )$. $Body^C$ is defined recursively by (5) and (6) so we shall develop recursive equations for $wp(\ Body^C\ ,\ \|S(x_0)\|^2\ )$. First we look at the weakest precondition of a single iteration. Let $P[S]$ stand for a predicate P containing one or more free occurrences of variable S and $P[T]$ stand for P with all free occurrences of S replaced by T. We calculate $wp(\ Body\ ,\ P[S]\ )$ as follows (omitting the constraint $0 \leq i < N$):

$$
\begin{aligned}
& wp(\ Body\ ,\ P[S]\ ) \\
= \;& wp(\ S := (\lambda i \cdot S(i) - 2.f(i).S(i))\ ;\ S := (\lambda i \cdot 2.\overline{S} - S(i))\ ,\ P[S]\ ) \\
= \;& \quad \text{by (2)} \\
& wp(\ S := (\lambda i \cdot S(i) - 2.f(i).S(i))\ ,\ wp(\ S := (\lambda i \cdot 2.\overline{S} - S(i))\ ,\ P[S]\ )\ ) \\
= \;& \quad \text{by (1)} \\
& wp(\ S := (\lambda i \cdot S(i) - 2.f(i).S(i))\ ,\ P[(\lambda i \cdot 2.\overline{S} - S(i))]\ ) \\
= \;& \quad \text{by (1)} \\
& P[(\lambda i \cdot 2.\overline{S'} - S'(i))] \quad \text{where } S'(i) = S(i) - 2.f(i).S(i) \; .
\end{aligned}
$$

Now, straightforward calculation shows that:

$$
\begin{aligned}
\overline{S'} \;&=\; \overline{S} - \frac{2.S(x_0)}{N} \\
2.\overline{S'} - S'(i) \;&=\; 2.\overline{S} - \frac{4}{N}.S(x_0) + (2.f(i) - 1).S(i) \; .
\end{aligned}
$$

Hence, we get:

$$
\begin{aligned}
wp(\ Body\ ,\ P[S]\ ) \;&=\; P[S''] \qquad\qquad (8) \\
\text{where } S''(i) \;&=\; 2.\overline{S} - \frac{4}{N}.S(x_0) + (2.f(i) - 1).S(i).
\end{aligned}
$$

From (8), we have that

$$
\begin{aligned}
wp(\ Body,\ \|S(x_0)\|^2\ ) \;&=\; \|S''(x_0)\|^2 \\
&=\; \|2.\overline{S} - \frac{4}{N}.S(x_0) + (2.f(x_0) - 1).S(x_0)\|^2 \\
&=\; \|2.\overline{S}\ +\ (1 - \frac{4}{N}).S(x_0)\|^2 \; .
\end{aligned}
$$

Now this has the form $\|\ A.\overline{S}\ +\ B.S(x_0)\ \|^2$ and using (8) we can again show that for any values A, B:

$$
\begin{aligned}
wp(\ Body,\ \|\ A.\overline{S}\ +\ B.S(x_0)\ \|^2\ ) \;&=\; \|\ A'.\overline{S}\ +\ B'.S(x_0)\ \|^2 \qquad\qquad (9) \\
\text{where } A' \;&=\; A + 2.B \\
B' \;&=\; \frac{N.B - 2.A - 4.B}{N} \; .
\end{aligned}
$$

This recurring structure suggests that we define A_i and B_i as follows:

$$
A_{i+1} \;=\; A_i + 2.B_i \qquad\qquad (10)
$$

$$B_{i+1} \;=\; \frac{N.B_i - 2.A_i - 4.B_i}{N}, \tag{11}$$

to give

$$wp(\; Body,\; \| \; A_i.\overline{S} \;+\; B_i.S(x_0) \; \|^2 \;) \;=\; \| \; A_{i+1}.\overline{S} \;+\; B_{i+1}.S(x_0) \; \|^2 \;. \tag{12}$$

By induction over j, we get

$$wp(\; Body^j,\; \| \; A_i.\overline{S} \;+\; B_i.S(x_0) \; \|^2 \;) \;=\; \| \; A_{i+j}.\overline{S} \;+\; B_{i+j}.S(x_0) \; \|^2 \;. \tag{13}$$

We take $A_0 = 0$ and $B_0 = 1$ and apply $Body^C$ to (7) as follows:

$$\begin{array}{ll} & wp(\; Body^C,\; \|S(x_0)\|^2 \;) \\ = & \quad \text{since } A_0 = 0,\, B_0 = 1 \\ & wp(\; Body^C,\; \| \; A_0.\overline{S} \;+\; B_0.S(x_0) \; \|^2 \;) \\ = & \quad \text{by (13)} \\ & \| \; A_C.\overline{S} \;+\; B_C.S(x_0) \; \|^2 \;. \end{array}$$

Finally, we apply the initialisation to this:

$$\begin{array}{ll} & wp(\; Init,\; \| \; A_C.\overline{S} \;+\; B_C.S(x_0) \; \|^2 \;) \\ = & \| \; A_C.\dfrac{1}{\sqrt{N}} \;+\; B_C.\dfrac{1}{\sqrt{N}} \; \|^2 \\ = & \dfrac{\| \; A_C + B_C \; \|^2}{N} \;. \end{array}$$

Thus we have shown that:

$$wp(\; Grover,\; S = |x_0\rangle \;) \;=\; \frac{\| \; A_C + B_C \; \|^2}{N} \;.$$

That is, the probability, $P(C, N)$, of observing the correct value after C iterations is:

$$P(C, N) \;=\; \frac{\| \; A_C + B_C \; \|^2}{N} \;.$$

By applying standard mathematical analysis techniques to the formula $A_C + B_C$, we can derive the following closed form for $P(C, N)$:

$$\begin{array}{l} P(C, N) \;=\; \sin^2((2.C + 1).\theta_N) \\ \text{where } \theta_N \;=\; \arcsin \dfrac{1}{\sqrt{N}} \;. \end{array}$$

This is the same as the formula presented in [Boyer et al. 1998]. The derivation of this closed form is outlined in the appendix.

It is interesting to note that $P(C, N)$ is periodic in C. This can be seen clearly in Fig. 2 which graphs $P(C, N)$ against C for $N = 128$. Here, an optimum probability of success is reached after eight iterations, where $P(8, 128) = 0.996$. After eight iterations, the probability starts to decrease again. The reason for the decrease is that, after eight iterations, the average immediately after the inversion about the origin operation goes below zero.

We wish to determine the optimum number of iterations for a given N. P reaches a maximum (and a minimum) for a given N when:

$$\frac{d}{dx} P(x, N) \;=\; 0 \;.$$

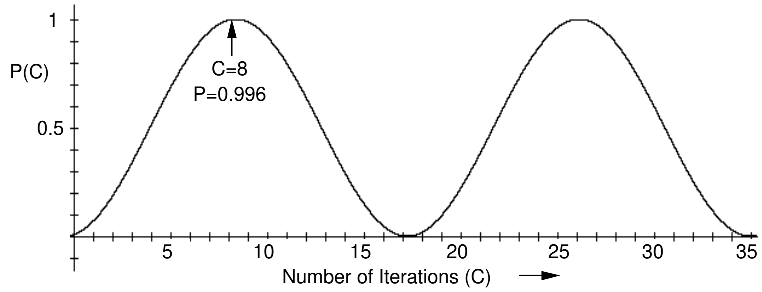


Fig. 2. Probability for Grover search with $N = 128$.

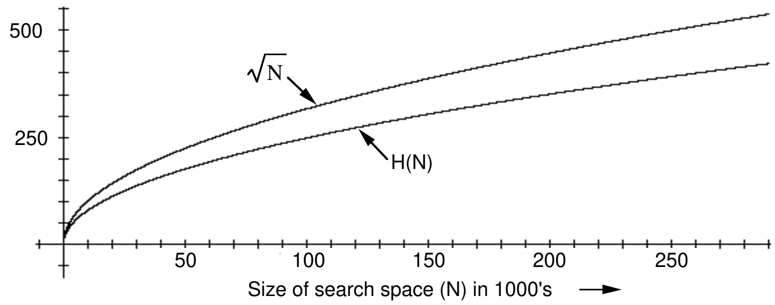


Fig. 3. Optimum number of iterations versus search space size.

It is easy to show that the first maximum for a given N is reached at $H(N)$ where

$$H(N) \quad = \quad \frac{\pi}{4.\theta_N} - \frac{1}{2} \quad \text{and} \quad \theta_N \ = \ \arcsin\frac{1}{\sqrt{N}} \ .$$

Thus the number of iterations in the Grover algorithm for a search space of size N should be the closest whole number to $H(N)$. $H(N)$ is graphed in Fig. 3. Since $\arcsin x$ is $\mathcal{O}(x)$ [Abramowitz and Stegun 1964, 4.4.40], we have that θ_N is $\mathcal{O}(1/\sqrt{N})$ and hence that $H(N)$ is $\mathcal{O}(\sqrt{N})$.

7. CONCLUSIONS

We have shown how Grover's search algorithm may be represented in the programming notation of the probabilistic *wp*-calculus. Any quantum computation consists of unitary transformations and probabilistic measurement and these can be modelled in this notation. Thus any quantum algorithm may be modelled in the notation. We believe that this language provides a more rigorous and elegant means of describing quantum algorithms than is normally used in the literature.

We have also shown how the rules of the probabilistic *wp*-calculus may be used to derive a recursive formula for the probability that Grover's algorithm finds the required solution. Using standard mathematical techniques, we were then able to find a closed form for this probability which corresponds to the formula presented in [Boyer et al. 1998]. The *wp*-calculus provides a clear and systematic means of stating the required outcome and of deriving the probability of achieving it.

It is instructive to compare the reasoning approach used here with the approach used in [Grover 1997] and [Boyer et al. 1998]. When reasoning using the probabilistic *wp*-calculus, we reasoned backwards from the expected outcome with a single arithmetic formula $\| A_i.\overline{S} + B_i.S(x_0) \|^2$. In contrast, [Grover 1997] and [Boyer et al. 1998] worked in a forwards manner reasoning about how a superposition, i.e., a vector of amplitudes, evolves from an initial distribution to a final distribution. Thus we only need to work with a single formula whereas they need to work with a vector of formulae[2] which can be more cumbersome. A similar observation is made in [Kozen 1985] and [Morgan et al. 1996] where it is shown that, in general, *wp*-style probabilistic reasoning involves working backwards with an expectation, whereas more conventional reasoning involves working forwards with a probability distribution.

In the case of Grover, we were able to derive an exact probability for success because the algorithm iterates a fixed number of times. Some algorithms iterate until some condition is met rather a fixed number of times. One such example is a generalisation of Grover's presented in [Boyer et al. 1998] which deals with the situation where there are an unknown number of values x satisfying $f(x) = 1$. In a case like this, we need to find the expected number of iterations rather than the probability of success. Such cases would require the specialised loop rules of [Morgan 1996].

An important feature of the calculus of [Morgan et al. 1996] not found in [Kozen 1985] and not used in this paper is the modelling of demonic nondeterminism. This is crucial in standard programming calculi to the notion of program specification and refinement as exemplified by [Morgan 1994] and it may be possible to apply such techniques to the derivation of quantum algorithms.

A. DERIVING A CLOSED FORM

We outline the derivation of the closed form expression for the probability of success of Grover's algorithm. The probability $P(C, N)$ is expressed in terms of the series A_i and B_i, which in turn are defined by the recurrence equations (10) and (11). To find a closed form for these recurrences, we first compute the generating functions $G_A(z) = (\Sigma_{i=0}^{\infty} A_i.z^i)$ and $G_B(z) = (\Sigma_{i=0}^{\infty} B_i.z^i)$ for A_i and B_i respectively using basic techniques [Knuth 1973, Sect. 1.2.9]:

$$G_A(z) = \frac{2Nz}{N + 2(2-N)z + Nz^2} \qquad G_B(z) = \frac{N - Nz}{N + 2(2-N)z + Nz^2} .$$

Computing the probability involves the sum $A_i + B_i$ and it seems reasonable to examine the Taylor series expansion of the sum of the two generating functions. Assume that z is such that the Taylor series expansion of $G_A(z) + G_B(z)$ converges:

$$\begin{array}{ll} & G_A(z) + G_B(z) \\ = & \dfrac{N + Nz}{N + 2(2-N)z + Nz^2} \\ = & \quad \text{Taylor expansion} \end{array}$$

[2] In fact, [Grover 1997] and [Boyer et al. 1998] simplify the reasoning by working with a pair of formulae, one representing the amplitude of $|x_0\rangle$ and one representing the amplitude of the other $N - 1$ states, rather than N formulae. However, this simplification may not be possible for all quantum algorithms.

$$1 + \frac{3N-4}{N}z + \frac{5N^2-20N+16}{N^2}z^2 + \frac{7N^3-56N^2+112N-64}{N^3}z^3 + \cdots .$$

We now observe that there is a strong similarity between the coefficients and powers of N in the enumerators above and the coefficients and powers of $\sin(\theta)$ in the multiple angle formula for $\sin(n\theta)$:

$$\begin{array}{l}\sin(1\theta) = 1\sin(\theta)\\ \sin(3\theta) = 3\sin(\theta) - 4\sin^3(\theta)\\ \sin(5\theta) = 5\sin(\theta) - 20\sin^3(\theta) + 16\sin^5(\theta)\\ \sin(7\theta) = 7\sin(\theta) - 56\sin^3(\theta) + 112\sin^5(\theta) - 64\sin^7(\theta)\ .\end{array}$$

(Recall that $\sin^i \theta$ stands for $(\sin\theta)^i$ rather than repeated application of $\sin$.)

This similarity suggests that we express N in the form $\sin^{-2}(\theta)$ since, for example,

$$\frac{5N^2-20N+16}{N^2}[N := \sin^{-2}(\theta)] \quad = \quad 5 - 20\sin^2(\theta) + 16\sin^4(\theta)\ .$$

We write θ as θ_N and choose it so that $\sin^{-2}(\theta_N) = N$, i.e., $\theta_N = \arcsin\frac{1}{\sqrt{N}}$.

Rewriting N as $\sin^{-2}(\theta_N)$ in $G_A(z) + G_B(z)$ and re-calculating the Taylor series expansion gives:

$$G_A(z) + G_B(z)\ [N := \sin^{-2}(\theta_N)] \ = \ \sum_{i=0}^{\infty} \frac{\sin((2i+1)\theta_N)}{\sin\theta_N} z^i\ .$$

Note that $\sin\theta_N \neq 0$ for $N \neq 0$. Since the Taylor series expansion has the form

$$G_A(z) + G_B(z) \ = \ \sum_{i=0}^{\infty} (A_i + B_i)\, z^i,$$

we conclude that

$$A_i + B_i \ = \ \frac{\sin((2i+1)\theta_N)}{\sin\theta_N}\ .$$

The probability of success $P(C, N)$ is $\parallel A_C + B_C \parallel^2/N$. However, it is easy to see from equations (10) and (11) that for any positive naturals C, N, $A_C + B_C$ is real and not complex so that $\parallel A_C + B_C \parallel^2 = (A_C + B_C)^2$. We then obtain a closed form for the probability:

$$\begin{aligned} P(N, C) &= \parallel A_C + B_C \parallel^2/N \\ &= (A_C + B_C)^2/N \\ &= (\sin((2C+1)\theta_N).\sin^{-1}\theta_N)^2/N \\ &= (\sin((2C+1)\theta_N).\sqrt{N})^2/N \\ &= \sin^2((2C+1)\theta_N)\ . \end{aligned}$$

ACKNOWLEDGEMENTS

Thanks to Tony Hey and other members of the Quantum Sticky Bun Club for inspiration and to Peter Høyer and the anonymous referees for valuable comments on the paper.

REFERENCES

ABRAMOWITZ, M. AND STEGUN, I. A. 1964. *Handbook of mathematical functions*. Number 55 in Applied Mathematics Series. U. S. Dept. of Commerce, National Bureau of Standards.

BENIOFF, P. 1980. The computer as a physical system: a microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics 22*, 563–591.

BERNSTEIN, E. AND VAZIRANI, U. 1993. Quantum complexity theory. In *25th ACM Annual Symposium on Theory of Computing*. 11–20.

BERTHIAUME, A. 1996. Quantum computation. In *Complexity Theory Retrospective II*. Springer-Verlag. Available from `http://andre.cs.depaul.edu/Andre/publicat.htm`.

BOYER, M., BRASSARD, G., HØYER, P., AND TAPP, A. 1998. Tight bounds on quantum searching. *Fortschritte Der Physik 46,* 4-5, 493–505. Available from `http://xxx.lanl.gov/abs/quant-ph/9605034`.

DEUTSCH, D. 1985. Quantum theory, the Church-Turing principle and the universal quantum computer. In *Royal Society London A 400*. 96–117.

DEUTSCH, D. AND EKERT, A. 1998. Quantum computation. *Physics World 11,* 3, 47–52. Available from `http://www.qubit.org/`.

DIJKSTRA, E. 1976. *A Discipline of Programming*. Prentice-Hall.

GROVER, L. 1997. Quantum mechanics helps in searching for a needle in a haystack. *Physical Review Letters 79,* 2, 325–328.

KNUTH, D. 1973. *The Art of Computer Programming - Volume 1: Fundamental Algorithms*, second ed. Addison-Wesley, Reading, Mass.

KOZEN, D. 1985. A probabilistic PDL. *J. Comput. Syst. Sci. 30*, 162–178.

MORGAN, C. 1994. *Programming from Specifications*, Second ed. Prentice–Hall.

MORGAN, C. 1996. Proof rules for probabilistic loops. In *BCS-FACS Refinement Workshop*, H. Jifeng, J. Cooke, and P. Wallis, Eds. Workshops in Computing. Springer–Verlag.

MORGAN, C., MCIVER, A., AND SEIDEL, K. 1996. Probabilistic predicate transformers. *ACM Trans. Program. Lang. Syst. 18,* 3, 325–353.

Received September 8 1998; accepted December 17 1998