

Using Grover's Algorithm for Genetic Search

Matthew Whitehead
mwhiteh@cs.indiana.edu

Dec. 14, 2005

Abstract

Grover's Algorithm provides an efficient way to perform database searches. We show how this form of quantum search may be used to improve the effectiveness of traditional genetic search on a classical computer.

1 Introduction

Genetic Algorithms [3] provide an effective way of searching complex fitness landscapes. They do this using an artificial survival of the fittest technique to weed out less desirable solutions and only keep the more desirable solutions.

While they are good at searching complex search areas, they are still considered computationally intensive. Complex fitness landscapes require complex solutions which take a long time to produce even with a genetic algorithm.

Grover's Algorithm [1] is a quantum algorithm used to search through a database of values and determine where a particular target value is stored. It does so by using quantum parallelism so that it is more efficient than its classical counterparts. The way the algorithm was originally designed allows for finding only a single value. We use repeated applications of Grover's Algorithm to get a variety of decent chromosomes that will then be used to form a starting population for classical genetic search.

The technique described in this paper is to combine the use of Grover's Algorithm with classical genetic search in order to lessen the computational requirements. This would allow a genetic search algorithm to quickly converge on reasonable solutions.

The basic idea is as follows:

1. Initialize a quantum computer using l qubits, where l is the necessary length of the chromosomes.
2. Set the quantum computer to a superposition of all possible chromosomes. This represents a population of all possible solutions.
3. Decide on the size, n , of the seed population needed. This is based upon the complexity of the problem and the available computing resources.
4. Perform Grover's Algorithm n times using the fitness function f to guide the search. Each solution found will be a single chromosome of the final seed population. The quantum registers are re-initialized in between each run.
5. Use the n solutions from above as an initial seed population for classical genetic search. This seed population will be fitter than a randomly generated population, as is commonly used as a starting population.

2 Classical Genetic Search

Genetic search algorithms [3] attempt to optimize arbitrarily complex functions using techniques inspired by natural evolution. They do this by creating a *population of chromosomes* that are encoded solutions to the given problem. These solutions are then recombined with one another in different ways in order to mix desirable schemata from both parent solutions. The resulting children solutions will then, hopefully, be more effective at solving the given problem.

Genetic search works well for problems that other search algorithms have difficulty with. These problems are typically ones with complex fitness landscapes.

2.1 Pseudocode for a standard genetic algorithm

- Let f be a fitness function that describes how well a particular chromosome solves the problem at hand.
- Let n be the size of starting population of chromosomes.
- Let c be the crossover rate (the probability that a particular selected chromosome "mates" with another selected chromosome and the offspring shares attributes of both parents).
- Let m be the mutation rate (the probability that a particular value in a chromosome changes when crossover occurs).
- Let g be the number of generations.

- Let t be the desired fitness threshold.
- Let $\max(pop)$ be the maximum fitness of all the individual chromosomes in pop .
- 1. Generate the initial population of random chromosomes, pop_1
 2. Repeat the following steps g times or until $\max(pop) > t$:
 - (a) Compute $f(x)$ for each x in pop_i
 - (b) Select $n \times c$ chromosomes from pop_i . Add the non-selected chromosomes to pop_{i+1} .
 - (c) Take the set of selected chromosomes and perform the crossover operation between pairs. Chromosomes with higher fitness functions are more likely to be chosen as part of a pair.
 - (d) As offspring chromosomes are created, flip a bit with probability m as bits are copied to offspring.
 - (e) Add the resulting offspring to pop_{i+1}

3 Grover's Algorithm

Grover's Algorithm is a quantum algorithm used to search for a particular value in a database. It provides speedup through the use of quantum parallelism. Multiple values can be searched at the same time.

3.1 Pseudocode for Grover's Algorithm

1. Initialize the quantum computer to be an equal superposition of all possible N states.
2. Repeat the following steps $O(\sqrt{N})$ times:
 - (a) Let the system be in any state. Evaluate the search function at that state. If it evaluates to 1, then phase rotate by π radians. Otherwise, do not change the system.
 - (b) Apply the diffusion transform. Create a diffusion transform matrix with $D_{ij} = \frac{2}{N}$ if $i \neq j$ and $D_{ii} = -1 + \frac{2}{N}$ otherwise.
3. Finally, measure the system to get the desired state with probability greater than $\frac{1}{2}$.

4 Using Grover's Algorithm for Function Optimization

The problem we want to solve is that of function optimization. That is, optimize a given fitness function. This is really the same as searching for the most fit individual chromosome in a population of all possible chromosomes. We want to do this with the speedup of quantum computation using Grover's Algorithm, but there is a problem. Grover's Algorithm is only defined for search where there is a single desirable solution. The problems that we want to be able to handle may have multiple ideal solutions. Also, perhaps a "good" solution would still be acceptable to us if finding the best solution is much more difficult.

In order to still use the speedup of quantum parallelism provided by Grover's Algorithm, we modify it slightly. The basic idea is to use an adaptive search function [2] instead of a static search function and run the algorithm multiple times until an appropriate solution is found. With an adaptive search function, the search function being used by Grover's Algorithm changes with each consecutive run, each time improving upon the last run's results.

4.1 Pseudocode for adaptive Grover search

- Let f be the given fitness function.
- Let S be the set of all possible values.
- Let sf_i be the i^{th} search function.
 1. Pick a random starting value, x_0 , from S
 2. Evaluate $y_0 = f(x_0)$
 3. Repeat the following steps until y_i is as large as desired
 4. (a) Set sf_i to $f(x_i) > y_{i-1}$
 - (b) Perform a Grover search using sf_i for $O(\sqrt{N})$ steps
 - (c) Evaluate $y_i = f(x_i)$
 5. x_i is the solution

Each iteration through the above algorithm has sf changing to search for new values that yield better results than the previous iteration.

5 Seeding a Genetic Search

Now we have all the pieces necessary to improve classical genetic search.

1. Determine an appropriate fitness function, f , for the given problem.
2. Determine an appropriate size of population of chromosomes, n .
3. Perform the adaptive Grover search algorithm given above n times. Each solution is used as a single member of the starting population.
4. Use the seed population to start a classical genetic search. The highly fit seed population will help the classical genetic search algorithm converge on a good solution more quickly than with a randomized starting population.

References

- [1] Grover, Lov K., “A fast quantum mechanical algorithm for database search” *Proceedings, STOC 1996*, Philadelphia PA, USA.
- [2] Bulger, D., Baritomba, W. P., Wood, G. R., “Implementing Pure Adaptive Search with Grover’s Quantum Algorithm” *Report UCDMS2000: College of Sciences, Massey University*, June 10, 2000. 1094–1105, Aug 1996.
- [3] Whitley, D. “A Genetic Algorithm Tutorial”, *Computer Science Department, Colorado State University*, Fort Collins, CO.