Function Maximization with Dynamic Quantum Search

MOUSSA, C.; Calandra, H.; Humble, T.S.; Feld S., Linnhoff-Popien C.

Function Maximization with Dynamic Quantum Search

Charles Moussa[1,2][0000−0002−5387−564X] and Henri Calandra[3],

and Travis S. Humble*[2][0000−0002−9449−0498]

[1] TOTAL American Services Inc., Houston, Texas USA
[2] Oak Ridge National Laboratory, Oak Ridge, Tennessee USA
[3] TOTAL SA, Courbevoie, France

Abstract. Finding the maximum value of a function in a dynamic model plays an important role in many application settings, including discrete optimization in the presence of hard constraints. We present an iterative quantum algorithm for finding the maximum value of a function in which prior search results update the acceptable response. Our approach is based on quantum search and utilizes a dynamic oracle function to mark items in a specified input set. As a realization of function optimization, we verify the correctness of the algorithm using numerical simulations of quantum circuits for the Knapsack problem. Our simulations make use of an explicit oracle function based on arithmetic operations and a comparator subroutine, and we verify these implementations using numerical simulations up to 30 qubits.

Keywords: Maximization · Quantum Search · Quantum Optimization

1 Introduction

Finding the maximal value in a poorly characterized function is a challenging problem that arises in many contexts. For example, in numerical simulations of particle dynamics, it is often necessary to identify the strongest interactions across many different particle trajectories. Prior results in quantum computing for unstructured search have shown that a quadratic speed up is possible when searching a poorly characterized function, and in this work, we consider an application of quantum search to the case of finding a maximal function value.

Quantum search was proposed originally to identify a marked item by querying an unstructured database [6]. By using an oracular implementation of the database function, Grover proved that a quadratic speedup in unstructured search could be obtained relative to brute force search by using superposition states during the querying phase. These ideas have been applied to a wide variety of application contexts including function maximization and minimization. In particular, Durr and Hoyer have shown how to design an iterative version of quantum search to find the minimizing argument for an unspecified oracle [5]. Their approach uses an updated evaluation of the oracle based on prior measurement observations, namely, to identify the smallest observed value. More recently, Chakrabarty et al. have examined the problem of using a dynamic Grover search oracle for applications in recommendation systems [3]. In addition, Udrescu et al. have cast the application of quantum genetic algorithms into a variant of dynamic quantum search [9].

In this contribution, we implement the principles of dynamic quantum search for the case of function maximization. We use the Knapsack $0/1$ problem as an illustrative example, which is a constraint problem that finds the largest weighted configuration for a set of possible items. We extend recent work from Udrescu et al. [9] by using quantum circuits for arithmetic operations. Our implementation uses arithmetic operations to build the dynamic oracle for quantum search function maximization [1]. We test this implementation with a quantum program simulated using the Atos Quantum Learning Machine, which enables verification of the accuracy of the algorithm and estimation of the resources needed for its realistic implementation.

The paper is organized as follows: Sec. 2 formulates the quantum algorithm for function maximization using an iterative variant of quantum search, while Sec. 3 presents the explicit implementation of these ideas using quantum arithmetic circuits and results from numerical simulation of those circuits. Finally, we explain how quantum search is used in this context and show how we simulated the example step by step in Sec. 4.

2 Quantum Maximization of Dynamic Oracle

We show how to realize quantum search for function maximization using a dynamic oracle. As an illustrative example, we consider a Knapsack $0/1$ problem introduced in Udrescu et al. as quantum version of a binary genetic algorithm called the reduced quantum genetic algorithm (RQGA) [9]. In particular, the genetic algorithm underlying RQGA was reduced to Grover search, thus removing the need for crossover and mutation operations for creating and selecting new candidate solutions. Rather, the complete set of candidate solutions is realized as an initial superposition over all the possible binary strings, e.g., using a Hadamard transform to prepare a uniform superposition state. We extend these ideas by developing an implementation using dynamic quantum search to solve the Knapsack problem.

Our method begins with an initial uniform superposition state of the candidate solutions stored in an n-qubit register q, where the size is determined by the number of possible items. The initial superposition is evaluated with respect to an objective function noted as f. For our example, the function f computes the fitness of the candidate solution. The computed value for the fitness is stored in a second quantum register denoted as f. The register of size p stores this value using two's complement form, where the size is determined by the largest value of all items in the database. The fitness is computed explicitly using a unitary operator U_f what applies to both quantum registers q and f, i.e.,

$$(1/\sqrt{2^n}) \sum_{i=0}^{2^n-1} |i\rangle_q |0\rangle_f \rightarrow (1/\sqrt{2^n}) \sum_{i=0}^{2^n-1} |i\rangle_q |f(i)\rangle_f \tag{1}$$

After preparing a uniform superposition of the candidate solutions and associated fitness values, the amplitudes with a fitness value greater than a current maximum value are marked by an oracle operator. The oracle operator is implemented explicitly below using a comparator circuit that takes as input the computed fitness register and a classical threshold value. The amplitude of all marked items are subsequently amplified to increase the probability to measure a better candidate. In practice, we extend the definition of the fitness operator to exclude certain candidates in the superposition as being valid or invalid candidate. For example, when valid candidates must have a positive fitness value, then a single register element may be used to indicate mark values that are negative and therefore invalid. Theoretically, Grover's search requires $\approx 13.6\sqrt{M}$ steps to achieve an error rate of less than 0.5 with M being the size of the list searched [1].

Given this general overview of the algorithm, we next describe how to apply these ideas in a specific example for the Knapsack problem. Consider a backpack of n possible items, where each item has a value and a weight. We seek the set of items that maximize the total value of the backpack while also not exceeding a defined total weight. Udrescu et al. [9] have used the example of $n = 4$ items, with the following properties: Item 1 (7 kg, \$40), Item 2 (4 kg, \$100), Item 3 (2 kg, \$50), and Item 4 (3 kg, \$30). The maximum weight allowed is 10 kg. Table 1 summarizes the set of all possible solutions for this example, from which it is apparent that the optimal solution is the set of all items excluding Item 1.

We summarize the computational steps taken to solve this example using the dynamic quantum search method within the quantum circuit model. Several quantum registers are used to store and process the problem input data, and we consider all registers to be initialized in the $|0\rangle$ state.

- q: an n-qubit register for storing candidate solutions. $n = 4$ in this example.
- f: a p-qubit register for storing the value of the fitness calculation in two's complement representation, where p is sufficient to store the sum of all possible values. $p = 6$ is this example.
- w: a register for storing the total mass of a candidate backpack. $\dim(w) = 5$ using unsigned integer representation in this example.

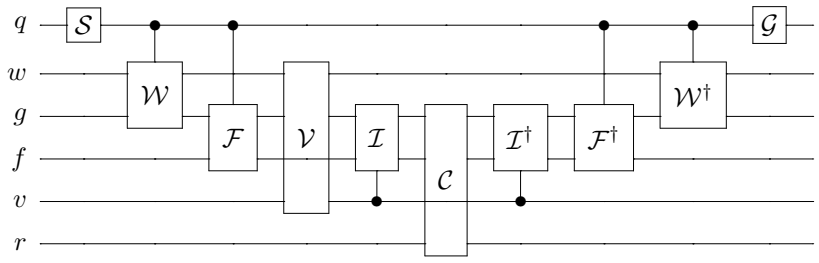

Fig. 1: This quantum circuit diagram represents the algorithm for function maximization using iterative Grover search with a dynamic oracle. We only represent with one Grover iteration for simplicity (more would require repeating the steps after creating the superposition). The quantum registers q, w, g, v, and f define respectively storage space for the candidate solutions, computed weight, garbage, validity, and fitness values. Gates in the circuit are $\mathcal{S}$ for preparing the initial superposition of candidate states, $\mathcal{W}$ for computing candidate weight, $\mathcal{F}$ for computing candidate fitness value,$\mathcal{V}$ for computing validity (by loading the maximum weight in register g and using a comparator with register g) , $\mathcal{I}$ for inversing fitnesses of invalid candidates, $\mathcal{C}$ for comparing to the current maximum (after being loaded in register g), $\mathcal{G}$ represents the inversion about average operator. Measurement of the q register should return a candidate that has a higher fitness than the current maximum loaded in g.

Table 1: Candidate solutions and values for Knapsack problem data.

Candidate	Fitness	Weight	Validity
0 0 0 0	0	0	valid
0 0 0 1	30	3	valid
0 0 1 0	50	2	valid
0 0 1 1	80	5	valid
0 1 0 0	100	4	valid
0 1 0 1	130	7	valid
0 1 1 0	150	6	valid
0 1 1 1	**180**	**9**	**valid**
1 0 0 0	40	7	valid
1 0 0 1	70	10	valid
1 0 1 0	90	9	valid
1 0 1 1	120	12	invalid
1 1 0 0	140	11	invalid
1 1 0 1	170	14	invalid
1 1 1 0	190	13	invalid
1 1 1 1	220	16	invalid

- g: a register for storing intermediate computational states and numerical constants. $\dim(g) = 6$ for this example.
- v: a 1-qubit register to mark a candidate fitness as valid or invalid.
- r: a 1-qubit register used by the oracle.

Following the declaration and initialization of all registers, the candidate register q is transformed under the n-fold Hadamard operator to prepare an uniform superposition of possible solution states. An initial fitness value threshold is selected by random number generation. We then apply a composite operator denoted as $o(x)$ that consists of the following stages:

1. Calculate the total mass, fitness, and validity of the candidates in registers w, f, and v.
2. Compare the fitness register f against the current threshold value stored in g. If the fitness value is greater than the current threshold value, then set r with the effect of marking the states, that is having the effect $|-\rangle \rightarrow (-1)^{o(x)}|-\rangle$ on the qubit oracle.
3. Uncompute the register f, w, and v.
4. We finally measure (after application of Grover iterations) the candidate solution register q (and fitness register f if we apply again the fitness computation circuit).

When computing the total mass and fitness, we store the weight and value of each item in the register g. Using a controlled adder, we perform addition with this register to store the sum of all weights and values for each candidate solution. As an illustration, consider the case that the fitness and mass are initially zero.

We begin by adding the mass and value of the first item to the register. In our example problem, Item 1 has a mass 7 kg and value 4$. We load the unsigned integer mass 7 (000111) into the workspace register g. The first qubit of the register q acts as a control register for adding the mass of Item 1 in the register m using controlled addition circuit. Similarly, we add the value 4 to the fitness register f for those candidate solutions containing Item 1. This sequence of steps prepare the following quantum state (omitting workspace registers for clarity),

$$\sum_{i=0}^{7} |i\rangle_q |00000\rangle_w |000000\rangle_f + \sum_{i=8}^{15} |i\rangle_q |00111\rangle_w |000100\rangle_f \tag{2}$$

The remaining items in the database are treated similarly.

As shown in Table 1, some candidate solutions have a total weight exceeding the constraint condition. These candidates are flagged by setting the register v to 1, a step implemented by comparing the maximum weight constraint with the value of the register w. If the candidate weight exceeds the constraint, the register v is flipped and we inverse their values to put them into their negative form so that when comparing to the current maximum, they will not be marked using the comparator quantum circuit operator. The negation operation is controlled by the validity qubit and implemented by complementing the fitness register f and adding 1 to it as it is done in two's complement arithmetic.

$$\left(\sum_{i \in valid} |i\rangle_q |m(i)\rangle_w |f(i)\rangle_f |0\rangle + \sum_{i \in invalid} |i\rangle_q |m(i)\rangle_w |-f(i)\rangle_f |1\rangle_v \right) |001010\rangle_g \tag{3}$$

At this stage, we wish to mark those candidate states that are valid and uncompute all values other than the candidate register. After preparing a superposition state representing valid possibilities, we use amplitude amplification to other registers as $|0\rangle$ and the qubit oracle as $|-\rangle$:

$$\sum_x |i\rangle_q (-1)^{o(i)} |-\rangle_r \tag{4}$$

We provide a pseudo-code representation for the steps before amplitude amplification in the Appendix. Grover iterations are finally used to change amplitudes. The number of required Grover iterations depends on the number size of the database as well as the number of actual solutions M as

$$\left\lceil \frac{\pi}{4} \sqrt{\frac{N}{M}} \right\rceil \tag{5}$$

In our search for the maximizing solution, we do not know the number of solutions at each iteration in a maximization process with this quantum algorithm for finding the maximum. We therefore must use a version of quantum search that circumvents the need to estimate M. Boyer et al. [2] have provided methods for ensuring probabilistic success of the algorithm in the absence of knowledge about the number of items.

Quantum search for the case M is unknown

1. Initialize $m = 1$ and set a constant $\lambda = 6/5$.
2. Choose a random integer j that is less than m.
3. Apply j Grover routines over the superposition of candidates
4. In our case, apply again the part 1 of the oracle to have a fitness and the validity. We can also do this part with a classical function.
5. Measure to get an outcome candidate/fitness/validity.
6. If we get a valid solution and the fitness is greater than the current max, we stop and return them as new solution and new current max.
7. Else set m to $\min(\lambda m, \sqrt{N})$ and repeat from step 2.

We may not have a solution in the case we already have the maximum of the problem but an appropriate time-out or early stopping can be set and the complexity would remain $\sqrt{N}$.

As an illustration, consider the case that the current threshold is 13. Candidate states in the superposition for our example problem will be marked if their associated weight is strictly greater than 13, as is the case for candidates 0110 and 0111. The resulting superposition becomes

$$\frac{1}{4}\left(\sum_{i\neq 0110, i\neq 0111} |i\rangle - |0110\rangle - |0111\rangle\right) \tag{6}$$

The result of the Grover search prepares the state

$$\frac{1}{8}\left(\sum_{i\neq 0110, i\neq 0111} |i\rangle\right) - \frac{5}{8}\left(|0110\rangle + |0111\rangle\right) \tag{7}$$

which yields acceptable candidate solutions with a probability $|5/8|^2 \approx 0.39$.

3 Quantum Circuit Simulation

We describe our implementation of these algorithm using integer arithmetic with unsigned integers presented in a binary representation and signed integers presented as two's complement. Let $a = (a_0, a_1, ..., a_{n-1})$ be the n-bit representation of the integer A, such that $A = \sum_i a_i 2^i \in [0, 2^n - 1]$. For signed integers, one bit is used for the sign and we can represent numbers $[-2^{n-1}, 2^{n-1} - 1]$. The most significant bit represents the sign, which is '+' for 0 and '−' for 1. Given two integers a and b, bitwise addition is used to calculate the resultant $a + b$. For subtraction, recall that $a - b = (a' + b)'$ where x' is the bit-wise complement of x obtained by flipping all bits. Usually, an extra bit is required for addition but when not used, we actually do modular addition. To compare two numbers a and b, one can do the subtraction and look at the last bit of the subtraction result. When it is 1, $a < b$.

The quantum ripple adder acts on inputs a and b by encoding those binary representations as the $|a, b\rangle$. The resulting output from in-place addition is the

state $|a, a+b\rangle$. Note that we use another bit called high bit as the addition result may exceed 2^n. In Ref. [8] a quantum circuit without ancilla qubit and with less quantum operations than previously proposed circuits was designed. This approach uses the Peres gates described in Fig. 2.

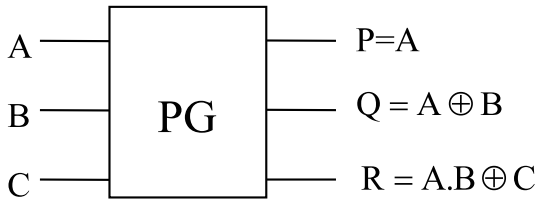

Fig. 2: Peres gate representation.

Cuccaro et al. previously described how the quantum ripple-adder may be adapted for other arithmetic tasks [4], and we make use of these implementations for our oracle definitions. For example, we define a subtraction method using the complement technique mentioned above. For modular addition, we consider the last significant bit of the number b as the high bit, and apply the adder on the $n-1$ first bits of a and b, and apply a CNOT onto qubits representing the last significant bits. For a comparator, we use only the part of the adder where we compute the high bit and uncompute. If the high bit is 1, $a < b$. For a controlled adder, where given a control qubit we apply the adder if the control is $|1\rangle$, a quantum circuit was proposed in [7].

For our simulations, we use the Atos Quantum Learning Machine, a numerical simulation environment designed for verifying quantum software programs. It is dedicated to the development of quantum software, training and experimentation. We use the QLM as a programming platform and a high-performance quantum simulator, which features its own universal quantum assembly programming language (AQASM, Atos Quantum Assembly Language) and a high-level quantum hybrid language (PyQASM) embedded in the popular Python language. The QLM is capable of performing exact simulations of up to about 30 qubits using a vector-based representation of register states. The simulator manages the execution of the programmed sequence of quantum gates.

For our small example instance, we can find the optimal backpack in two Grover iterations (or two circuits of one Grover iteration). Some instances require more due to the probabilistic measurement outcomes, but a typical execution converges to valid solutions. It is possible to use the suboptimal solution as the threshold for the next instance of the algorithm in order to verify if better solutions are available. We just need a set of n qubits to represent all possibilities while classically, this would be represented by many bit strings to form a population of candidates that will change over iterations like it is done in genetic algorithms.

4 Conclusion

In this contribution, we have shown how to implement function maximization using dynamic quantum search. Our approach has applied an explicit oracle for the quantum search algorithm and implemented a quantum algorithm for binary optimization of the Knapsack problem. Our implementation illustrates the idea of evaluating the fitness of all possible candidate solutions. However, our approach does require uncomputation of any ancillary register to ensure the correctness of the Grover iteration. While the added complexity is manageable, it would be advantageous to develop alternative search methods that act on subspaces of entangled registers. We anticipate that such methods would avoid the need for uncomputation of the ancillary registers and, in our example, permit more efficient steps to iteratively isolate the maximal fitness value.

Acknowledgments

CM and HC acknowledge support from Total and TSH acknowledges support from the U.S. Department of Energy, Office of Science, Early Career Research Program. Access to the Atos Quantum Learning Machine was provided by the Quantum Computing Institute at Oak Ridge National Laboratory.

A Pseudo-code for the Knapsack example

References

1. Ahuja, A., Kapoor, S.: A quantum algorithm for finding the maximum (11 1999), arXiv:quant-ph/9911082
2. Boyer, M., Brassard, G., Hoeyer, P., Tapp, A.: Tight bounds on quantum searching (1996). https://doi.org/10.1002/(SICI)1521-3978(199806)46:4/5<493::AID-PROP493>3.0.CO;2-P
3. Chakrabarty, I., Khan, S., Singh, V.: Dynamic grover search: applications in recommendation systems and optimization problems. Quantum Information Processing **16**(6), 153 (Apr 2017). https://doi.org/10.1007/s11128-017-1600-4, https://doi.org/10.1007/s11128-017-1600-4
4. Cuccaro, S., Draper, T., Kutin, S.: A new quantum ripple-carry addition circuit (10 2004), arXiv:quant-ph/0410184
5. Durr, C., Hoyer, P.: A quantum algorithm for finding the minimum. arXiv preprint quant-ph/9607014 (1996)
6. Grover, L.K.: A fast quantum mechanical algorithm for database search (1996)
7. Muñoz-Coreas, E., Thapliyal, H.: T-count optimized design of quantum integer multiplication. CoRR **abs/1706.05113** (2017)
8. Thapliyal, H., Ranganathan, N.: Design of efficient reversible logic-based binary and bcd adder circuits. J. Emerg. Technol. Comput. Syst. **9**(3), 17:1–17:31 (Oct 2013). https://doi.org/10.1145/2491682, http://doi.acm.org/10.1145/2491682
9. Udrescu, M., Prodan, L., Vladutiu, M.: Implementing quantum genetic algorithms: a solution based on grover's algorithm. In: Conf. Computing Frontiers (2006)

Algorithm 1 Pseudo-code of the quantum circuit for the steps before amplitude amplification.

Require: Current maximum value, quantum circuit with all qubits prepared in $|0\rangle$.

1: Apply Hadamard Transform on register q to get all possible candidates in superposition.
2: Set register r as $|-\rangle$.

▷ Compute the weight w_i of the candidates.
3: Load w_1 in register g.
4: Control-add w_1 into register w if the first qubit of register q is 1.
5: Load w_2 in register g.
6: Control-add w_2 into register w if the second qubit of register q is 1.
7: Load w_3 in register g.
8: Control-add w_3 into register w if the third qubit of register q is 1.
9: Load w_4 in register g.
10: Control-add w_4 into register w if the fourth qubit of register q is 1.

▷ Compute the fitness f_i of the candidates.
11: Load f_1 in register g.
12: Control-add f_1 into register f if the first qubit of register q is 1.
13: Load f_2 in register g.
14: Control-add f_2 into register f if the second qubit of register q is 1.
15: Load f_3 in register g.
16: Control-add f_3 into register f if the third qubit of register q is 1.
17: Load f_4 in register g.
18: Control-add f_4 into register f if the fourth qubit of register q is 1.

▷ Define validity of the candidates.
19: Load the maximum weight (10 kg in the example) in register g.
20: Use a comparator with register w to define the validity of a candidate in v.
21: Inverse the fitness f of the invalid candidates (for which v is 1).

▷ Mark better candidates.
22: Load current maximum in register g.
23: Use a comparator with register f to mark better candidates than the current one in register r.
24: Uncompute.

This figure "Totallogo.png" is available in "png" format from:

http://arxiv.org/ps/1902.00445v1

This figure "logoornl.png" is available in "png" format from:

http://arxiv.org/ps/1902.00445v1