Washington University in St. Louis

Washington University Open Scholarship

Report Number:

2024

Optimization of a Quantum Subset Sum Oracle

Angelo G. Benoit, Sam Schwartz, and Ron K. Cytron

We investigate the implementation of an oracle for the Subset Sum problem for quantum search using Grover's algorithm. Our work concerns reducing the number of qubits, gates, and multi-controlled gates required by the oracle.

We describe the compilation of a Subset Sum instance into a quantum oracle, using a Python library we developed for Qiskit and have published in GitHub. We then present techniques to conserve qubits and gates along with experiments showing their effectiveness on random instances of Subset Sum. These techniques include moving from fixed to varying-width arithmetic, using partial sums of a set's integers to determine... **Read complete abstract on page 2.**

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

Recommended Citation

Optimization of a Quantum Subset Sum Oracle

Angelo G. Benoit, Sam Schwartz, and Ron K. Cytron

Complete Abstract:

We investigate the implementation of an oracle for the Subset Sum problem for quantum search using Grover's algorithm. Our work concerns reducing the number of qubits, gates, and multi-controlled gates required by the oracle.

We describe the compilation of a Subset Sum instance into a quantum oracle, using a Python library we developed for Qiskit and have published in GitHub. We then present techniques to conserve qubits and gates along with experiments showing their effectiveness on random instances of Subset Sum. These techniques include moving from fixed to varying-width arithmetic, using partial sums of a set's integers to determine specific integer widths, and sorting the set to obtain provably the most efficient partial sums.

We present a new method for computing bit-string comparisons that avoids arbitrarily large multiple-control gates, and we introduce a simple modification to the oracle that allows for approximate solutions to the Subset Sum problem via Grover search.

Optimization of a Quantum Subset Sum Oracle

Angelo Benoit, Sam Schwartz, and Ron K. Cytron
Washington University
Saint Louis, Missouri 63130
cytron@wustl.edu

Abstract—We investigate the implementation of an oracle for the Subset Sum problem for quantum search using Grover's algorithm. Our work concerns reducing the number of qubits, gates, and multi-controlled gates required by the oracle.

We describe the compilation of a Subset Sum instance into a quantum oracle, using a Python library we developed for Qiskit and have published in GitHub. We then present techniques to conserve qubits and gates along with experiments showing their effectiveness on random instances of Subset Sum. These techniques include moving from fixed to varying-width arithmetic, using partial sums of a set's integers to determine specific integer widths, and sorting the set to obtain provably the most efficient partial sums.

We present a new method for computing bit-string comparisons that avoids arbitrarily large multiple-control gates, and we introduce a simple modification to the oracle that allows for approximate solutions to the Subset Sum problem via Grover search.

I. Introduction

Quantum computers are attractive for solving optimization problems due to the theoretical advantage of unstructured search using Grover's algorithm [1]. In this paper we examine qubit-conserving implementations of an *oracle* for deployment for Grover's algorithm. Here, the oracle of interest realizes an instance of the *Subset Sum* problem (SSP) [2]. That problem is known to be NP-Complete. While Grover cannot overcome the exponential cost currently believed to be necessary for such problems, it does asymptotically obtain quadratic speedup as compared with exhaustive search.

In particular, we are interested in the optimization problem related to SSP, which we describe in Section I-A. Instances of NP-Complete optimization problems, such as memory-map inference [3] and similar hard problems can be reduced from and to SSP. The techniques we describe in this paper allow such hard problems to be formulated as an instance of SSP whose solution may provide good, if not optimal, solutions to the hard problems of interest.

The contributions of our paper are as follows. We have developed a library to support arithmetic in an oracle (Section VI). That library is freely available in `GitHub` and it embodies the techniques we have describe in Section II that conserve quantum bits (qubits) in such oracles. We present results of experiments conducted in `Qiskit` [4] to quantify the savings of qubits for random SSP instances. While the results we can obtain are precise under emulation, deployment on a real quantum device (such as IBM's 127 qubit quantum computers available openly for public use [5]) encounters unacceptable noise. One likely source of that noise [6] is the use of multi-controlled gates whose controlling qubits are (for example) all the bits of integers compared for equality. In Section III we propose a novel method for determining the equality of two bit strings. If the strings under comparison are m qubits wide, current approaches use a multi-controlled gate with m control bits. We use at most two controlling inputs for any control gate and take modest additional steps $(\Theta(\log \star(m)))$ to determine equality. Finally, we describe in Section IV an adaptation of the oracle to find approximate solutions to Subset Sum using Grover search.

In our notation, we consistently use subscripts to denote an element taken from a collection, such as a set, and superscripts to denote a particular bit of a bit string or number.

A. Subset Sum

The Subset Sum problem (SSP) is a decision problem that can be described as follows. Given a (multi)set

$$S = \{a_1, a_2, \ldots, a_n\}$$

of n positive integers, is there a subset of those integers that sums to another integer, T? This problem is known to be NP-Complete, which, at the time of this writing, means there is no algorithm yet that can answer the above question any faster than trying all possible subsets of S. While SSP may not have inherent interest, it is a common problem from which interesting, hard problems can be reduced. One such problem we have previously considered is the *memory map inference* problem. Given a sequence of storage references (loads and stores), infer which memory allocation regions waste the least amount of space [3]. These would be the memory-map requests made from such a program to an operating system.

B. Corresponding optimization problem

Asking whether a memory map exists with a given efficiency can be reduced from SSP, thus showing that the memory map inference problem is NP-Complete. More interesting and relevant to this paper, the *optimization* form of SSP allows us to ask what memory map is closest to a given, specified target. This, in turn, asks whether SSP can find a subset close to a given target T.

It is important to remember that the size of an SSP instance is usually characterized by n, the number of integers in set S. While that plays a role in the quantum deployment of an SSP instance, we must concern ourselves also with the integers themselves and the partial sums computed in service of finding whether a given subset sums (close) to T. Those aspects of

SSP consume significant resources on a quantum computer. This becomes apparent when we describe how to compile an SSP instance into a quantum circuit in Section I-C.

C. Grover's algorithm and the oracle

A complete description of Grover's algorithm is beyond the scope of this paper and has been covered well elsewhere [1], [7]. The important aspects of that algorithm for the purposes of this paper are as follows:

- The algorithm incorporates instances of an oracle that models the behavior of some function $f(x)$ where x is an n-qubit value. There are thus $N = 2^n$ possible basis values for x in the computational basis.
- We assume that for exactly one such input value $x^\star$, $f(x^\star) = 1$ and $\forall\ x \neq x^\star, f(x) = 0$. Extensions where there are more input values for which our function returns 1 are straightforward [1].
- We denote the bits of x as $x_1, x_2, \ldots, x_n$.
- For the purposes of SSP, x encodes a particular subset of S whose values might sum to T. A particular bit x_i is true if and only if the associated integer from SSP a_i should be included in the sum.
- In the scenario described thus far, Grover's algorithm can find $x^\star$ in time $\Theta(\sqrt{N})$, which is quadratically faster than the exhaustive search taking $\Theta(N)$ time.
- Grover's algorithm performs a certain number of iterations, related to n, each consisting of the oracle subjected to *phase kickback* [7] followed by a *diffusion* step.
- If Grover's algorithm requires g iterations, then the quantum circuit will have g instantiations of the oracle, each followed by an instantiation of the diffusion step.

We next describe how we compile an instance of SSP into an oracle for Grover's algorithm. An outline of the steps is as follows, and a sketch of the process is depicted in Figure 1.

- The circuit accepts x (n qubits) and an ancilla y. The value of x selects the subset of integers from S, as described above. The ancilla y is initialized to the $|-\rangle$ state, as required by Grover's algorithm to induce phase kickback.
- For each integer a_i in S, a quantum register is created and initialized to the bits of a_i. We interchangeably call the integer and its quantum register a_i. For now, each integer value in the circuit occupies 32 bits, but the improvements we suggest in Section II will allocate qubits more judiciously.
- For each a_i, a *shadow* register b_i is created that will hold either a_i or 0, depending on bit x_i. We denote bit j of a register using superscripts. Thus we have

$$b_i^j = a_i^j\ \wedge\ x_i$$

Bit j of the shadow register is true if and only if the corresponding bit of a_i is true *and* x_i specifies the inclusion of integer a_i for the sum. The above logic is easily realized in the quantum circuit by making b_i^j (initialized to $|0\rangle$) the target of a **CCX** gate controlled by a_i^j and x_i.

- Next we must sum the shadow registers. A running sum of the n integers takes $n - 1$ registers. We do not as yet assume any order among the n integers, but in Section II we describe an optimal ordering for them in terms of saving qubits.
- The final sum Z of the chosen subset must then be compared with the target T. We compare each bit Z^j with T^j by performing their exclusive-or using two **CX** gates, as shown in Figure 3(a). If each resulting bit is called C^j, then we finally determine whether all of those bits are 0. In literature, this is accomplished by a multi-controlled **CX** gate, whose controls are each of the bits C^j followed by an **X** gate and whose target then indicates whether all of the control bits are $|0\rangle$. A diagram of this circuit is shown in Figure 3(b), which includes the uncomputation after the multi-controlled gate using an **X** gate. We propose an alternative computation that uses only **CCX** gates in Section III.

The final result computed by the oracle is $|1\rangle$ if and only if the input x chooses a subset of S that sums to T. Such an oracle is suitable for Grover's algorithm to find the solution x.
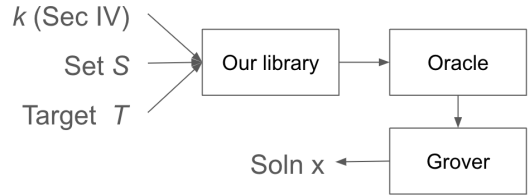

Fig. 1. Overview of the steps for generating an oracle. S is the set of integers and T is the target for the sum. The k value is described in Section IV.

II. CONSERVING QUANTUM BITS AND GATES

As described in Section I, we initially generated our oracle using 32 qubits for the integers and summation operations. In this section we describe optimizations that allowed us to save qubits. Experimental results from most of these ideas are reported in Section V, but some remain future work for us.

A. Addition in a quantum circuit

Our library supports the addition of integers held in quantum registers. We mention here that we formulated the carry operation to use operations such as **CX** that are readily available for quantum circuits. As we generate the logic to add two registers, bit-by-bit we need the sum of three qubits. Two of the qubits, v and w, are fresh inputs; the third qubit is the carry c from the previous bits' addition. We compute the sum and carry(-out) out as follows:

$$sum = v \oplus w \oplus c$$
$$carry = vw \oplus vc \oplus wc$$

The *carry* expression differs from what is normally deployed for classical (non-quantum) computing, but is logically equivalent and avoids unnecessary boolean operations in a quantum circuit. The corresponding circuit is shown in Figure 2.
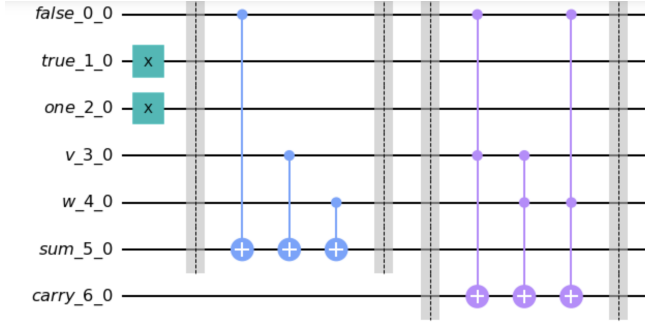

Fig. 2. Addition of qubits v and w. As this is the first addition, the carry-in is the $|0\rangle$ qubit at the top of the circuit. The circuit was generated by `Qiskit` using our library.

B. Reducing the width

Our results in Section V show that implementing all values and operations on 32 qubits integers rapidly exceeds the qubits available on real devices or even on available simulators. The set S is known *a prior*, so we can compute

$$M = \sum_{i=1}^{n} a_i$$

as the *maximum* value achievable over all possible subsets for this SSP instance. All integers and sums could then be accommodated using $\lfloor \log_2 M \rfloor + 1$ bits. Suppose WLOG that a_n is the largest integer in S and S has n elements. Then M can be bounded as follows:

$$M = \sum_{i=1}^{n} a_i \leq n * a_n$$

and the number of required qubits

$$\lfloor \log_2(n * a_n) \rfloor + 1 \leq \log_2 n + \log_2 a_n + 1$$
$$= O(\log(\max(n, a_n)))$$

The number of required qubits for an SSP instance is bounded logarithmically, by the number of elements in S and by its largest integer. Such a result would normally be encouraging, but qubits are precious on quantum devices, so we investigated other approaches to reducing the necessary qubits.

C. Varying-width set elements and sums

While M as computed above suffices to represent all values, an instance of SSP may specify integers in S that require fewer qubits. We therefore implemented a varying-width arithmetic package for SSP, so that each integer $a_i \in S$ occupies only the necessary $\lfloor \log_2 a_i \rfloor + 1$ qubits. This in turn required redefining addition to accommodate inputs of differing qubit widths.

Each shadow register b_i is either a copy of its source register a_i, or it is 0, according to whether x_i specifies that a_i should be included in the sum. Thus, each shadow register b_i is the same width as its source register a_i.

The additions performed in service of an instance of SSP on the shadow registers must be allocated sufficient qubits to hold their results. With the shadow registers summed in some order (say, b_1 through b_n for now), when b_j is to be added to the growing partial sum, we allocate a register of width

$$\left\lfloor \log_2 \sum_{i=1}^{j} b_i \right\rfloor + 1 \tag{1}$$

to hold the result. The register must be this wide whether x_i specifies b_j to be the source value a_j (so, included in the sum) or 0 (so, excluded from the sum). We therefore accommodate a sum at this point that potentially includes a_j. In other words, the sum at that point can be at most the sum of the integers in SSP that may have been incorporated thus far, and we allocate a result for the sum that accounts for the worst case.

Experiments in Section V show the effectiveness of properly sizing each integer from S and of properly (worst-case) sizing the sums.

D. Optimal ordering of integers

S is a set (or multiset, allowing duplicates) of integers, which ordinarily implies no order among its elements. However, Equation 1 shows that the number of bits at each summation step is related to the size of the integers and results that have been incorporated previously. Modeled after the *shortest job first* result [8], we introduce and prove the following:

> *Theorem* The partial sums developed in service of an SSP instance produce the smallest values when the integers of S are considered in sorted order, ascending.

Proof. Consider a set of n integers, $V = \{V_1, V_2, \ldots, V_n\}$. WLOG, assume that when sorted we obtain

$$V_1 \leq V_2 \leq \ldots \leq V_n$$

Our theorem is that every partial sum of the integers $\sum_{i=1}^{k} V_i$ is minimized if the integers are in sorted order. By contradiction, assume there exists an optimal schedule ρ that minimizes the partial sum of each step in the addition and that ρ is not sorted in ascending order. This means there must exist at least one pair of integers V_i and V_j where $i < j$ in ρ such that $V_i > V_j$.

If V_i and V_j are swapped, the partial sum for all sums between V_j and V_i decreases by $V_i - V_j$, as V_j is added first. This swap does not affect the partial sums before V_j or after V_i in the sequence.

Thus, swapping V_i and V_j to follow ascending order strictly decreases the partial sums between the V_i and V_j additions. By applying this swap iteratively for all such inversions in ρ, ρ becomes sorted in ascending order without increasing the partial sum at any step.

Since the assumption was ρ is optimal but we can decrease the partial sums (or keep them the same) by enforcing ascending order, the initial assumption that ρ is optimal must be incorrect. Therefore, sorting by ascending order minimizes the partial sum for all steps in the addition. □

Section V presents results on the savings realized by sorting set S.

E. Uncomputing the carry bits

Our library introduces new carry bits for each summation of qubits. An addition of two qubit registers, each m qubits wide, causes the allocation of m carry bits to hold the values produced by each pair of bit additions. The initial carry-in is $|0\rangle$ as shown in Figure 2, but we generate a carry-out at the leftmost addition that is never needed.

It is therefore plausible that $\frac{1}{3}$ of our qubits are allocated for carry operations. It has been shown that a single carry bit could service all additions, threading its way through the circuit [9]. This would work as follows:

- The carry-out computation is prepared as shown in Figure 2 using three **CCX** gates that target the carry-out bit.
- The value at that point is then used for the next two qubit additions as the carry-in. In Figure 2, the role of that qubit is the *false* qubit shown at the top.
- Once the control point is specified, the carry qubit can be *uncomputed* by applying the inverse of the gates that gave it its current value. The **CCX** gate is its own inverse, so we simply replay the three operations shown in purple in Figure 2 in reverse of the order in which they were applied.
- The carry qubit can now be used to receive carry-out of the next qubits' summation.

While this increases the gate count for the resulting circuit, we expect the number of qubits could be cut by $\frac{1}{3}$. Another issue is the noise on the singleton carry qubit, relied upon throughout the entire circuit. We consider this future work, to investigate the savings on qubits as compared with possible increased noise when so many operations are applied to a single qubit.

F. Double-buffering of sums

In the same way that the carry bit can be uncomputed and used for each carry computation [9], the partial sums that result from incorporating designated elements of S could be accomplished using *double buffering*. The summation register that received its result at time t could be used at time $t+2$, with the result from $t+1$ and the next integer from S summed into the recycled register. Instead of n summation registers, we would need only 2. The register from time t must be *uncomputed* so that it returns to its initial state and is ready to receive the summation at step $t+2$.

Using analysis similar to Section II-B, we could save

$$O(n \times \log(\max(n, a_n)))$$

qubits, where n is the size of S and a_n is the largest integer in S. This must be investigated because uncomputing has implications for noise and delay in the quantum circuit.

III. COMPUTING STRING EQUALITY

At the end of an SSP instance, the final sum developed by the various approaches described in Section II must be compared with the target T for the SSP decision problem. In this section, we present a new approach for computing equality between two strings (nominally considered to be the same width m).

Extensions to strings of different width are possible, but the implementation would depend on whether the strings are viewed as

- numbers, in which case the shorter string can be prepended with sufficient 0s to match the width of the longer string; or
- text, in which case the strings would be viewed as unequal owing to their difference in width.
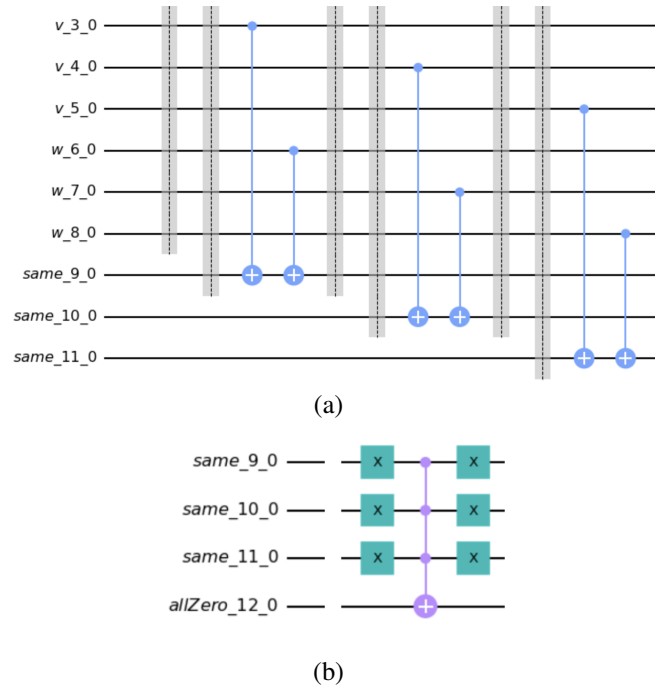

(a)

(b)

Fig. 3. Current method to determine $v = w$. In (a), qubits v_i and w_i are compared via exclusive-or, with the resulting `same` bit $|0\rangle$ if and only if they agree. The circuit continues in (b), where the multi-controlled **X** gate checks that all of the *same* control bits are 0.

The standard approach for a quantum circuit to detect whether strings v and w are identical is shown in Figure 3.

1) The equality of each pair of bits, v^i and w^i is computed by their exclusive-or. Thus, we compute

$$same^i = v^i \oplus w^i$$

using two **CX** gates as shown in Figure 3(a). The result $same^i$ is 0 if and only if the ith bits of v and w are the same. The resulting string $same$ shows which qubit

positions in v and w agree, and which do not. Where agreement exists, the corresponding bit of $same$ is $|0\rangle$

2) We next determine if *all* the qubits in register $same$ are $|0\rangle$. This is typically achieved using each as a control bit of a multi-controlled **X** gate, whose target is some qubit $allZero$, as shown in Figure 3(b). Thus, $allZero = |1\rangle$ if and only if all the qubits of $same$ are $|0\rangle$. Because we are checking that all control bits are 0, each requires an **X** gate prior to the multi-controlled **X** gate (and after, possibly, to uncompute).

We are concerned with the noise experienced for Grover on the IBM Q devices [5], and the multi-control gates have been identified as a source of such noise [6]. We therefore developed an alternate formulation for string equality. We base our new approach on the following:

$$\sum_{i=1}^{m} same^i = 0 \longleftrightarrow same = 0$$

In other words, $same$ is the integer 0 if and only if its digits sum to 0. The sum of such digits is at most m, when each is a 1. With reference to Equation 1, we can represent the sum of m bits using $\lfloor \log_2 m \rfloor + 1$ bits. For example, if $m = 1000$ bits then its sum can be represented in 10 bits. That sum can in turn be represented in 4 bits, and so on. Each step reduces the number of bits needed logarithmically. The number of such steps is thus $\Theta(\log \star(m))$. In the limit, once we have a sum held in two bits, we can reduce the number of bits no further. We finally test that the remaining two qubits are $|0\rangle$ which takes a single **CCX** gate.

We have thus reduced the problem of string equality to the problem of a series of additions, culminating in a **CCX** gate. We perform $\Theta(\log \star(m))$ additions, each using only **CX** and **CCX** gates. We have thus eliminated the need for a multi-controlled **CX** gate with more than 2 control bits.

Investigation of this replacement for string equality is future work for us. We intend to use IBM's emulators [5] that can model the noise of their quantum computers.

IV. ORACLE ADAPTATION TO ALLOW SSP APPROXIMATION

While the SSP decision problem is itself interesting, in terms of the hard problems we seek to solve, the *optimization* form of the problem is more attractive:

Given a (multi)set

$$S = \{a_1, a_2, \ldots, a_n\}$$

of n positive integers, which subset of S sums to a value as close to T as possible.

We say "which subset" because any subset is a possible solution, including the empty set. The optimization problem seeks a subset whose sum is as close to T as possible given the integers in S.

With an SSP instance formulated to find a subset whose sum is exactly T, a solution near T could be found by repeated runs of a Grover search using targets acceptably close to T.

However, the complexity of this approach introduces a linear factor related to the acceptable range around T. Binary search cannot help if the oracle performs exact matches for its target. This is because the integer range around T does not help to guide the search toward a solution.

In this section, we describe a simple adaptation of the oracle that allows for an approximate search around T in a single run of Grover's algorithm. That run can also provide information about whether the approximate match is greater than or less than T.

This problem is certainly NP-Hard because the acceptably close range of T could be just T itself.

A. Approximate solution with a run

The approach we have described in Section III culminates with an exact comparison with T. We describe here how that comparison can be relaxed to:

- search for a sum whose value approximates T and
- provide information for the next formulation of a Grover instance that will make the approximation more precise.

The approach works as follows, with an example shown in Figure 4.

- Within a given run of an SSP instance using Grover's algorithm, instead of matching against T we match against all *but* the least significant k bits of T.
- For $k = 0$ this tests for an exact match against T. When viewed from right to left, as k increases, the range for a match is correspondingly relaxed.
- In the quantum circuit, this is accomplished by matching only the most significant $n - k$ qubits and ignoring the others. We can use the string-equality circuit described in Section III.

			Value of k					
8	7	6	5	4	3	2	1	0
1	0	0	1	1	1	0	1	

Fig. 4. Example for approximate matching of T, with the bits of $T = 157$ shown on the bottom.

The effect of this approach, ignoring the least significant k bits, is to search for match in the interval

$$[T - (T \mod 2^k), \ T - (T \mod 2^k) + 2^k - 1]$$

The inclusive lower bound is obtained when the least significant k bits are all 0, and the upper bound is obtained when they are all 1. Using the example in Figure 4, we obtain the following inclusive intervals for the approximate search, based on a particular provided value for k:

k	$T \bmod 2^k$	Low	High	Diff
0	0	157	157	0
1	1	156	157	1
2	1	156	159	3
3	5	152	159	7
4	13	144	159	15
5	29	128	159	31
6	27	128	191	63
7	27	128	255	127

The range expands at the low side or high side, depending on the bit pattern of T. A zero at the current value of k maintains the low value, and a one at that position maintains the high value. As shown in the rightmost column, the approximation becomes exponentially worse with each increase in k.

We use this idea to modify our oracle generator from Figure 1 as follows:

- The generator accepts a specific value for k.
- The generated oracle determines success if T is matched on all but the least significant k bits.

Grover search then yields a superposition of subsets whose sums fall within the approximate range as described above for a given k. As described in Section I-C, a subset is represented as a particular value of x whose bits encode which elements of S are included in the subset whose sum falls within the approximate range. The superposition is then measured to yield a particular value of x, which corresponds to a particular subset of S.

Once a subset is observed, subsequent Grover searches can be formulated to expand or contract the search space for a solution near T.

V. EXPERIMENTS

To explore how effective our optimizations are on the number of qubits and gate operations of the circuit, we conducted empirical experiments on most of the ideas presented in Section II.

These experiments were run on `Python` 3.11.4 and `Qiskit` 0.45. A record of the raw data and mean calculations is archived in our `GitHub` repository.[1]

A. Setup

For testing, we generated instances of the Subset Sum problem with a randomized (multi)set of positive integers and a target between 1 and the sum of the entire set: an instance with a target beyond the entire set's sum clearly has no solution. We vary the maximum value allowed in the set between 64, 128, and 256. Our primary goal was to reduce the qubit requirements of the oracle, but we measured both qubit and gate operations as a function of the number of values in the set.

The number of qubits used in each circuit was calculated by counting each register created using our Quantum Circuit (QC) class. The gates of interest were counted by subtracting the uses of cosmetic quantum circuit barriers and measurement

[1] https://github.com/ABenoit0226/quantum-place-route

operations from a built-in `Qiskit` method for counting the number of circuit operations.

To obtain meaningful measurements, we compute the mean number of qubits and gate operations used over 100 runs (each with a randomized SSP instance) of each problem. Experiments were run between set sizes of 5 and 100 in increments of 5.

We evaluate 4 configurations of the subset sum implementation to demonstrate our circuit optimizations:

1) The baseline configuration uses fixed-width, 32-bit registers to hold integers and perform addition operations.
2) Here, we apply the optimization described in Section II-C and use varying-width registers, but only for the set's elements. We continue to use fixed-width registers for the partial sums based on the size of the total sum of all elements in the set, as described in Section II-B.
3) Continuing with the optimization described in Section II-C, we used varying-width registers for set elements *and* the partial sums.
4) Finally, we use the above optimizations but also perform sums using the sorted order of the set S as described in Section II-D.

B. Qubit Results

As shown in Figure 7, our transition to using varying width integers rather than a standard 32-bit integer (as a classical computer does) reduces the number of qubits used to create the SSP solver circuit substantially. This is particularly noticeable as the size of the problem grows and the qubit count between the baseline and the optimization diverges. This value manifests as a reduction of $\sim 58 - 73\%$ (see Figure 5).

We can further reduce the number of qubits used through two other optimizations in our calculation of the partial sums: using varying width partial sums and ordering the set before summation (see Section II-D). These optimizations further reduce the number of qubits used by $\sim 1.5 - 2.5\%$ each (see Figure 5).

As the maximum value allowed in the set increases, the percentage reduction in qubits decreases. This is expected as the baseline uses 32-bit registers by default and larger integers use a register size closer to 32 so the varying width methods save fewer qubits.

The decrease in qubit reduction as the set size increases is comes from the fact that the optimized methods' functions are respectively less linear. Despite appearing linear in terms of qubits/set size (see Figure 7), the fully optimized method's function actually increases in slope by roughly 15% between 10 and 100 values, whereas the fixed method's function experiences a negligible change ($<.01\%$) when the maximum allowed value is 256. This is due to partial sums approaching 32 bit width as the set size grows.

C. Gate Results

Reducing the number of gate operations is also important, as each gate introduces noise and delay in the circuit. As shown in Figure 8, using a width based on the total sum of all set

Max Value = 64

Set size:	5 values	50 values	100 values
Sums fixed width	72.80%	67.27%	65.40%
Sums varying width	74.57%	69.59%	67.79%
Optimal ordering	75.96%	72.02%	70.33%

Max Value = 128

Set size:	5 values	50 values	100 values
Sums fixed width	69.13%	63.83%	61.95%
Sums varying width	71.02%	66.12%	64.33%
Optimal ordering	72.39%	68.66%	66.90%

Max Value = 256

Set size:	5 values	50 values	100 values
Sums fixed width	65.67%	60.37%	58.53%
Sums varying width	67.43%	62.68%	60.91%
Optimal ordering	68.92%	65.19%	63.53%

Fig. 5. Percentage of qubits saved over Fixed 32 bit by optimization (max value in set = 64, 128, 256 respectively)

Max Value = 64

Set size:	5 values	50 values	100 values
Sums fixed width	61.73%	52.42%	49.82%
Sums varying width	64.19%	55.38%	52.87%
Optimal ordering	65.81%	58.38%	56.05%

Max Value = 128

Set size:	5 values	50 values	100 values
Sums fixed width	57.35%	48.27%	45.82%
Sums varying width	59.91%	51.13%	48.78%
Optimal ordering	61.47%	54.19%	51.91%

Max Value = 256

Set size:	5 values	50 values	100 values
Sums fixed width	53.53%	44.38%	41.92%
Sums varying width	55.85%	47.19%	44.81%
Optimal ordering	57.52%	50.14%	47.92%

Fig. 6. Percentage of gates saved over Fixed 32 bit by optimization (max value in set = 64, 128, 256 respectively)

elements instead of 32 bits significantly reduces the number of gates used in the SSP circuit as well. This value is further optimized by using varying widths for the partial sums and by ordering the set. As the problem grows, the gate count between the baseline and optimization diverges, although to a lesser extent than with the qubit counts. Overall, we see a reduction of $\sim 42 - 62\%$ from fixed-width to the size of the total sum of all elements, an additional $\sim 2.5 - 3\%$ by using varying width in the partial sums, and another $\sim 1.5 - 3.25\%$ from ordering the elements before summing.

Similar to the qubit results, there are slightly diminished reductions from the optimizations compared to the baseline as the maximum value allowed in the set increases and approaches 32 bits in size. There are also diminished reductions for more values in the set as the resulting sums are closer to the size of the 32-bit sums.

VI. Oracle library

Our library for initializing and executing subset sum oracles consists of 3 classes that support quantum circuits, arithmetic, and the Subset Sum problem. We also developed methods for testing and data generation that produced the data reported in Section V.

A. Quantum Circuit Class

We define a base quantum circuit class, `QC`, a wrapper class for the `Qiskit` Quantum Circuit class. It creates and manages the quantum circuit, with methods for adding registers that allow us to account for qubit usage and name the registers according to their use as an aid to understanding the generated circuit. The class provides for all basic Boolean operations and offers methods that optimize the generation of carry bits and chains of exclusive-ors, with a resulting circuit shown in Figure 2. While those operations can in theory be generated from the basic Boolean operators, qubits are spared by realizing these operations in a more compact form. For example,

a direct realization of exclusive-or in terms of the more basic Boolean operations burns 6 qubits for temporary values and uses 10 quantum gates. We do not report experimental results separately for these improvements but use them throughout our experiments.

The methods in this and the following classes accept information to perform a given operation and can either

- generate a new register to contain the result, or
- accept a register as a parameter that should receive the result.

The first use case is *functional* in nature and allowed us to write complicated operations very easily in terms of their constituent operations.

B. Quantum Arithmetic Class

Following the *decorator* pattern [10], we developed a class `QVarArith` that decorates the `QC` class. This supports the instantiation of integers, arithmetic on integer values, and comparison of integer values.

The class supports varying-width arithmetic by default, but it can be invoked to create or add integers of a specified width. This allows us to experiment with the savings of varying-width arithmetic as described in Section II-C.

C. Subset Sum Class

We define a class to manage instances of the Subset Sum problem, `QSubsetSum`. The class takes a list of integer values (our set S), and an integer target (T) generates a `QC` object containing the oracle.

VII. Conclusion and future work

We have examined several techniques for reducing the number of qubits needed to generate an oracle for the Subset Sum problem. We presented results that show the greatest savings in qubits were seen by moving from fixed-width arithmetic
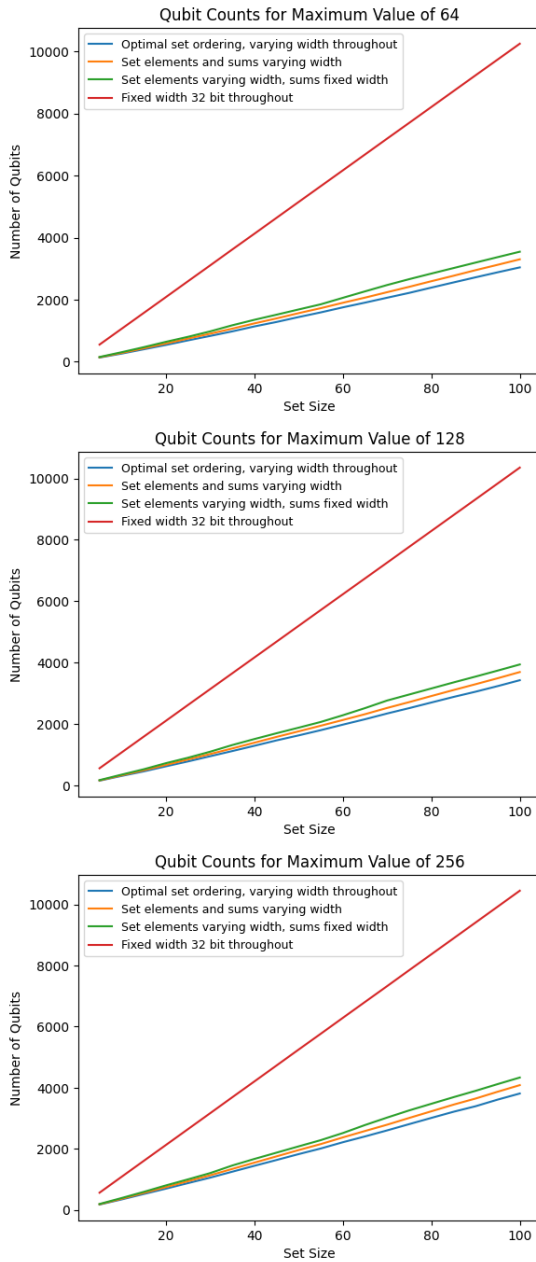
Fig. 7. Mean qubit usage of optimization vs. set size

Fig. 8. Mean gate usage of optimization vs. set size

to varying-width for the integers in S. Extending that idea to the sums brought some improvement, and sorting the set to minimize the partial sums yielded additional, modest gains. Our circuits also required fewer gates with each optimization, but qubits had been the limiting factor for running the subset sum problem in emulation or on actual hardware.

We described a new technique for bit-string comparisons in a quantum circuit that avoids a multi-controlled gate. We described a method for adapting the oracle to allow an approximate search.

The code producing our results in Section V is available in the `GitHub` repository referenced there. For future work we
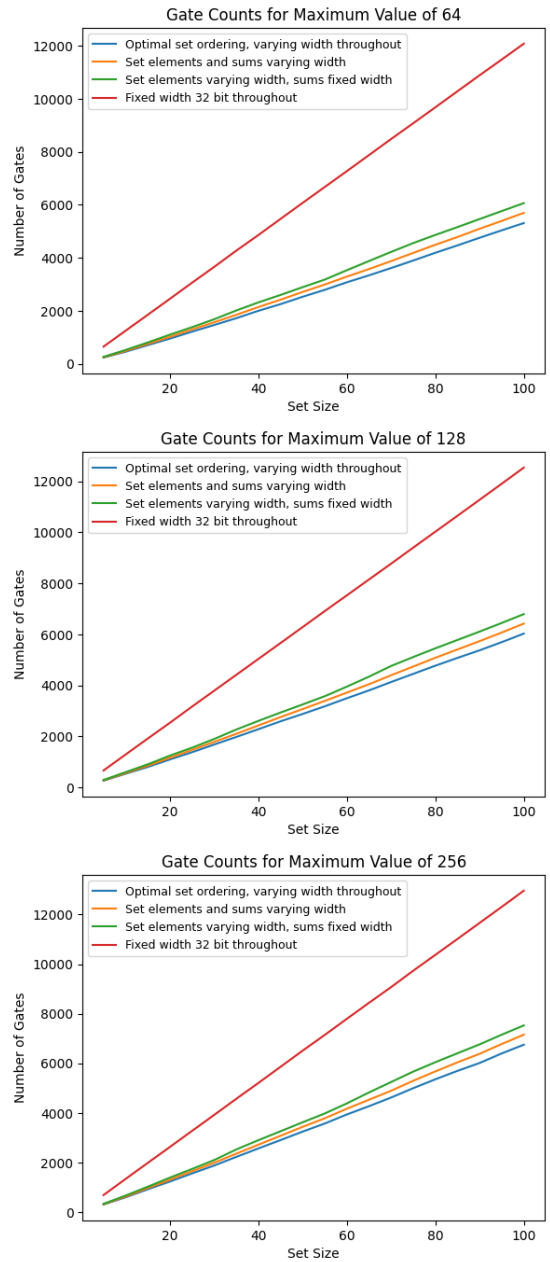
are interested in the following directions:

- We hope to apply the idea from [9] to save carry bits by uncomputing and reusing them. An issue could be the accumulation of noise using just a single qubit for such a purpose. We hope to consider a qubit uncompute and noise management scheme, in which we could use a given qubit for carries until the noise might become intolerable, prior to which point we would allocate a new qubit for carries.
- The double-buffering idea described in Section II-F could significantly reduce the qubits needed for the oracle, at the expense of a deeper circuit. The depth matters in

terms of delay and noise. We could manage those buffers in the same manner as the carry qubit, described above. We would reuse a given pair of sum registers until noise or depth becomes egregious.

- Our emulations using Grover's search provide great results, but the same circuits on current IBM hardware [5] are disappointing. We hope to localize the source of noise, which we believe to be the multi-controlled **X** gates. The technique in Section III avoids one such gate, but the diffuser step of Grover's algorithm still uses such a gate.

REFERENCES

[1] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 212–219. [Online]. Available: https://doi.org/10.1145/237814.237866

[2] M. R. Garey and D. S. Johnson, *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[3] P. T. Scheid, A. J. Spilo, and R. K. Cytron, "Inferring memory map instructions," *SIGACT News*, vol. 45, no. 3, p. 47–52, sep 2014. [Online]. Available: https://doi.org/10.1145/2670418.2670433

[4] Qiskit contributors, "Qiskit: An open-source framework for quantum computing," 2023.

[5] A. C. Santos, "O computador quântico da ibm e o ibm quantum experience," *Revista Brasileira de Ensino de Física*, vol. 39, no. 1, Sep. 2016. [Online]. Available: http://dx.doi.org/10.1590/1806-9126-RBEF-2016-0155

[6] E. Wilson, F. Mueller, L. Bassman Oftelie, and C. Iancu, "Empirical evaluation of circuit approximations on noisy quantum devices," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476189

[7] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[8] A. Cobham, "Priority Assignment in Waiting Line Problems," *Operations Research*, vol. 2, no. 1, pp. 70–76, February 1954. [Online]. Available: https://ideas.repec.org/a/inm/oropre/v2y1954i1p70-76.html

[9] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton, "A new quantum ripple-carry addition circuit," 2004.

[10] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.