Quantum A-Star Search

Presenting a Grover's Based Approach to Optimal Path Finding

Sam Bieler, Michael Tylko, Eli Shieber

ES 170 | Professor Prineha Narang

May 8, 2019

Problem Statement

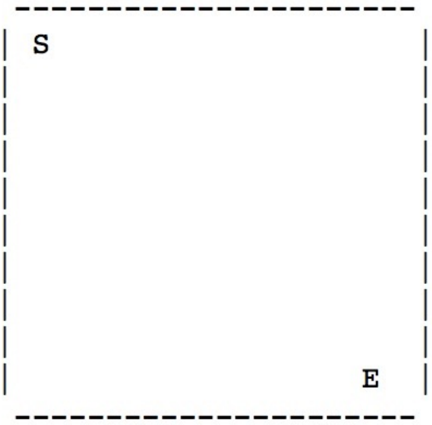
Find The Optimal Path

- Given a set of nodes, N, a set of edges, E, a start node, S, and a goal node, F, what is the optimal route to traverse from start to goal?
- Let $f : E \to \mathbb{R}$ represent a function from edge to edge cost.
- Let e_{ij} be the edge between node i and node j.
- Our goal is then to find a path, $P = (e_{Si}, \ldots, e_{jF})$, for which we minimize the objective:

$$\sum_{e_{ij} \in P} f(e_{ij})$$

Grid World Problem Set Up

- We frame this graph-path finding problem in a grid world with our start node in the top left corner and our end node in the bottom right corner:



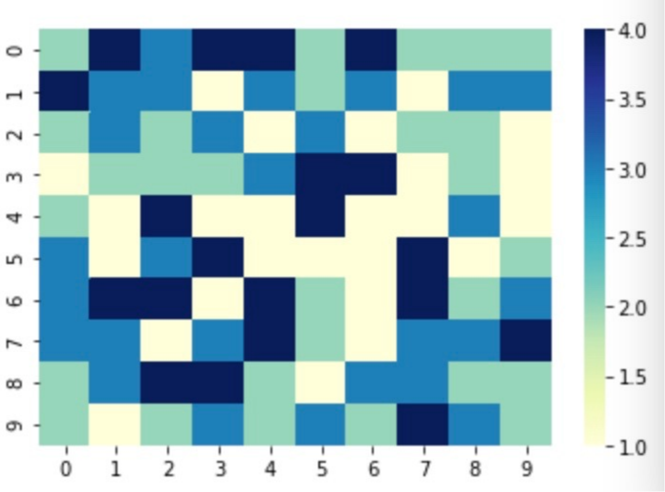
Grid World Problem Set Up

- We also constructed problems which involve a series of obstacles which our search agent will have to navigate around:

```
 -----------
|S    X     |
|   X       |
|   X   X   |
|   X X X   |
|   X X     |
| X   X X   |
|   X X     |
| X X X     |
|     X X   |
|   X     E |
 -----------
```

Grid World Problem Set Up

- We encode the cost for traversing to as node as either uniform 1 or as a random integer between 1 and 4:



Classical A^* Search

- One of the most widely used classical approaches to this problem is A^* search.
- A^* search constructs a tree of paths which originate from the start node. It then extends those paths one node at a time.
- A^* decides to extend nodes by selecting the node, N, which minimizes:

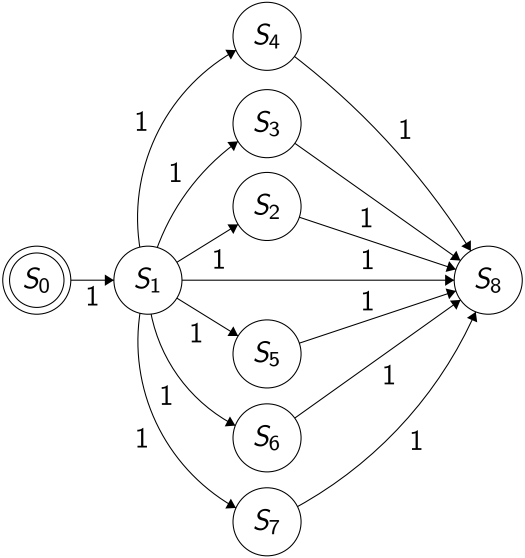
$$f(N) = g(N) + h(N)$$

- $g(N)$ represents the cost to move from the start node to N and $h(N)$ is a heuristic which estimates the cost of moving from N to the goal node.
- If A^* extends a path to the goal node it will return that path. As long as the heuristic function is not an overestimate of the costs, A^* will return the lowest-cost path.

Classical A^* Search

- A^* relies on a priority queue which it uses to select the node, N, with the minimum value $f(N)$.
- That priority queue is constructed using a binary insert function which runs in $\mathcal{O}(\log n)$.
- Items are removed from this priority queue in constant time.
- However, this can get unnecessarily expensive if we are adding many nodes to the priority queue but only actually need to remove a few in order to get the optimal path.
- One theoretical example of a problem with high insertion and low removal would be a graph with an optimal path with lots of neighbors for each node on the optimal path (given a perfect heuristic).
- So the two drivers of these types of problems depend on how good your heuristic is and how many neighbors you need to explore along the optimal path(s).

Problems with High Insertion and Low Removal



Quantum A^* Search

- We propose a modified A^* Search algorithm which will replace the priority queue with an unsorted list.
- Durr and Hoyer have proposed a grover's based algorithm to find the minimum element of a list which runs in $\mathcal{O}(\sqrt{n})$.
- We use this algorithm to find the node, N, with the minimum $f(N)$ value.
- This allows us to construct our list of possible node expansions in just $\mathcal{O}(n)$ instead of the priority queue's $\mathcal{O}(n \log n)$.
- Our removal from the list will run in $\mathcal{O}(\sqrt{n})$ instead of constant time.
- Our hypothesis is that this quantum approach may prove to be more effective at solving problems with a high number of insertions and a low number of removals.

Minimum Algorithm with Grover's

Durr and Hoyer (1996) [DH]

- Let T be an unsorted list of size N. Durr and Hoyer propose an algorithm to find the index of T's minimum in $\mathcal{O}(\sqrt{n})$ probes.
- This approach utilizes Grover's as a subroutine to find the index of some element smaller than a threshold.
- That index is then set to be the new threshold and the process is repeated until the probability that the new threshold is the list's minimum is sufficiently high.
- That probability $(1 - \frac{1}{2}^{c})$ increases as the number of iterations, c, increases.

Minimum Algorithm with Grover's

1. Choose threshold index $0 \leq y \leq N-1$ uniformly at random.

2. Repeat the following and interrupt it when the total running time is more than $22.5\sqrt{N} + 1.4 \lg^2 N$.[1] Then go to stage 2(2c).

   (a) Initialize the memory as $\sum_j \frac{1}{\sqrt{N}} |j\rangle|y\rangle$. Mark every item j for which $T[j] < T[y]$.

   (b) Apply the quantum exponential searching algorithm of [2].

   (c) Observe the first register: let y' be the outcome. If $T[y'] < T[y]$, then set threshold index y to y'.

3. Return y.

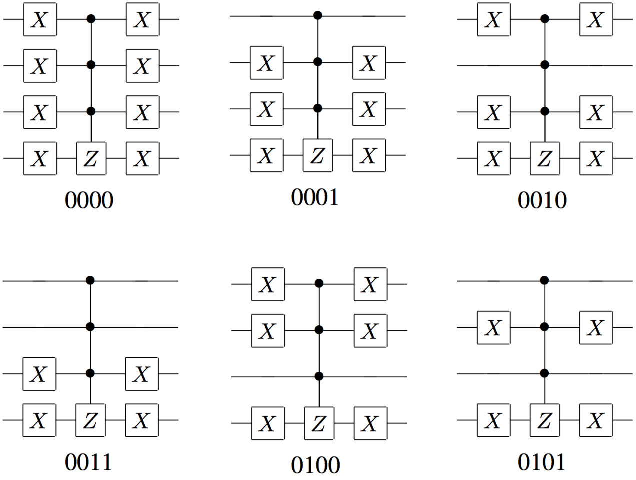
Why Do We Use *THIS* Method

Why not simply create an oracle that marks the minimum element?

- *Conceivable:* a logical gate sequence on qubits that can compare an element to another element
- *Hard To Conceive (Maybe Impossible ?!?!?!):* a logical gate sequence on qubits that can compare an element to a set of other elements in order to determine whether it is the smallest

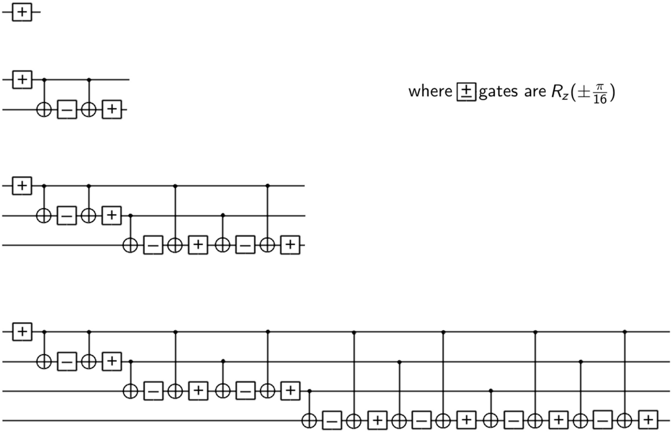
Oracle Construction

Six Example Four-Qubit Oracles [SK]:



Controlled Z-Gate Implementation

1-4 Qubit Case [NS] [SS]:



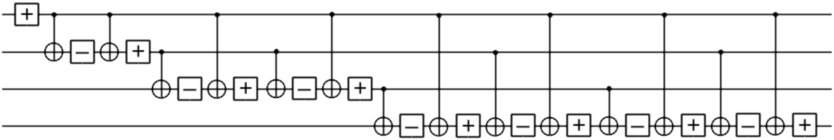
where $\boxed{\pm}$ gates are $R_z(\pm\frac{\pi}{16})$

Controlled Z-Gate Implementation

Removing excess information such as the R_z gates and the first controlled not gates simplifies the problem we have to solve.

Simplifying the problem leads to:

- 0
- 0,1,0
- 0,1,0,2,0,1,0
- 0,1,0,2,0,1,0,3,0,1,0,2,0,1,0



Controlled Z-Gate Implementation

```
def pattern(maxi, iterator, curList):
    if iterator == maxi:
        return curList
    else:
        midList = curList+[iterator]+curList
        qc.cx(qr[iterator], qr[iterator+1])
        qc.rz(-piFrac, qr[iterator+1])

        for i in range(len(midList)):
            qc.cx(qr[midList[i]], qr[iterator+1])
            if i % 2 == 0:
                qc.rz(piFrac, qr[iterator+1])
            else:
                qc.rz(-piFrac, qr[iterator+1])

        return pattern(maxi, iterator+1, midList)
```

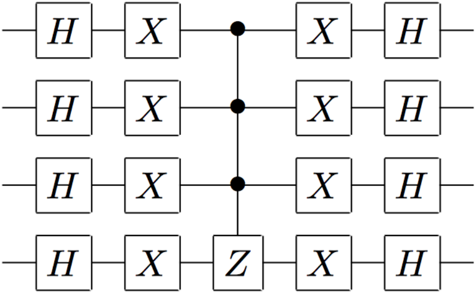
Implementing Minimum Grover's

Marking the Minimum Values

- Using this oracle construction and our controlled z gate we can now mark the minimum items in our list.
- In order to do this we iterated classically through the list to find items less than y

Implementing The Grover Diffusion Operator

- Trivial given the controlled Z Gate
- Add Hadamard gates and X gates for every rail on either side of the Oracle [SK]:



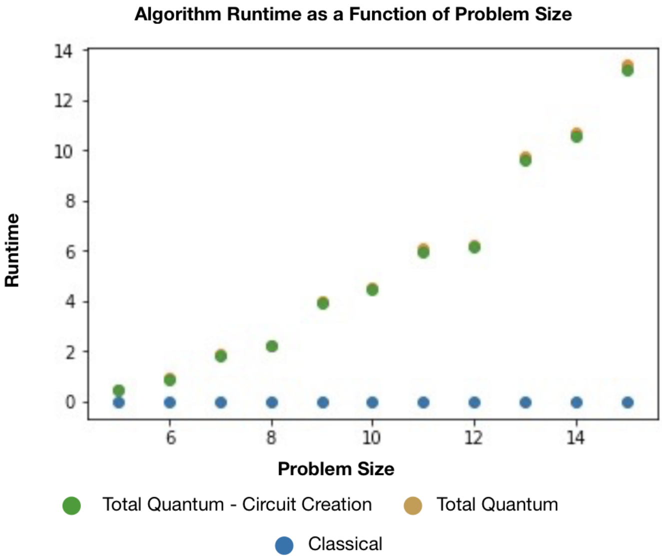
How Many Times Do We Run Grovers?

- Given k answers and N Qubits we run grovers g times according to the following formula [LMP]:

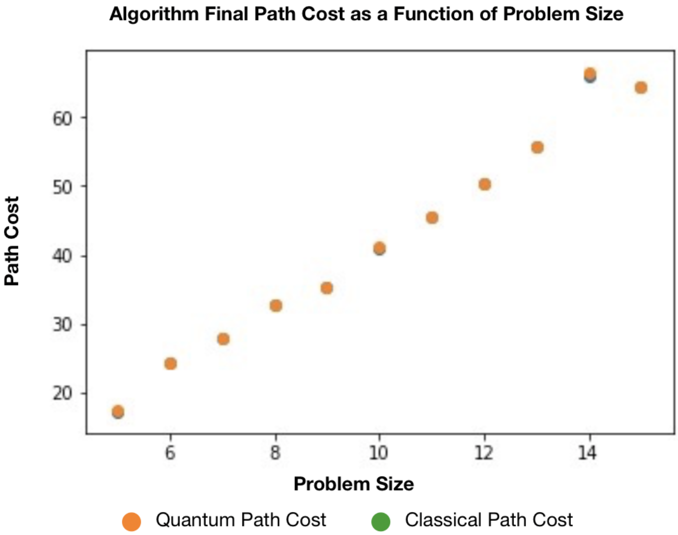
$$g = \frac{\pi}{4}(\frac{N}{k})^{\frac{1}{2}}$$

- In this case we get k classically
- You can also get the answer without the value of k

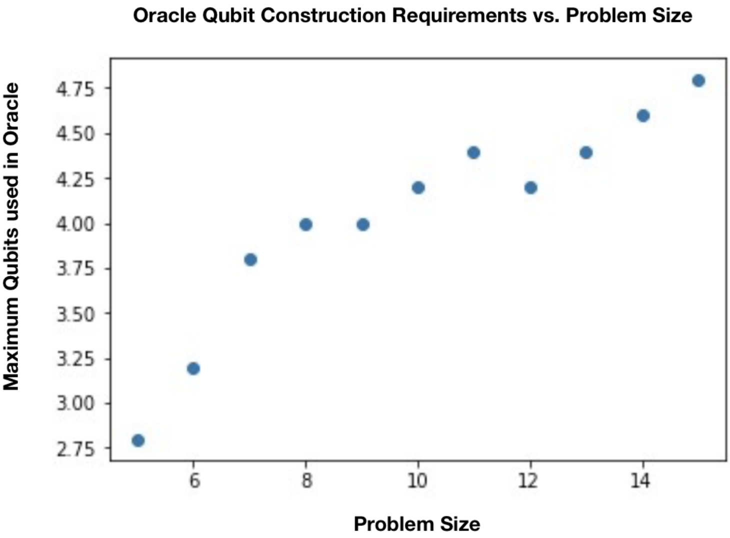
Analysis: Runtime Sensitivity



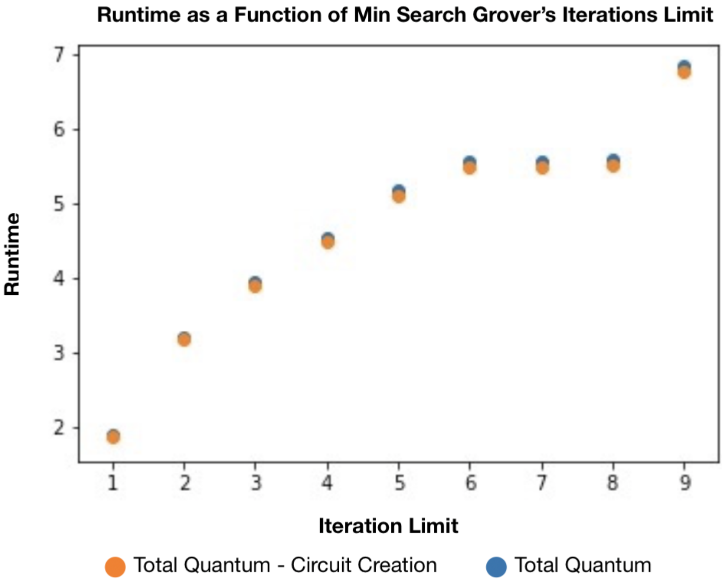
Analysis: Path Cost Sensitivity



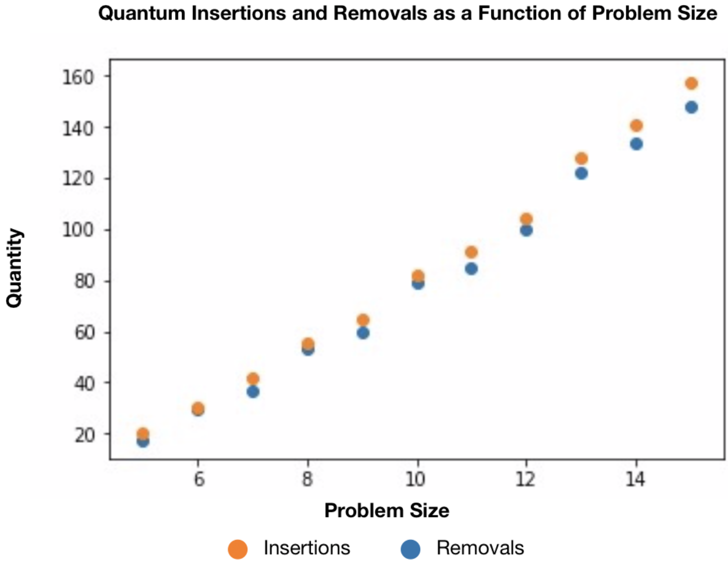
Analysis: Oracle Qubit Requirement Sensitivity



Analysis: Runtime Sensitivity w.r.t. Grover's Iteration Limit



Analysis: Insertion and Removal Rates



References

- Christoph Durr and Peter Hoyer, *A quantum algorithm for finding the minimum*.
- C. Lavor, L. R. U. Manssur, and R. Portugal, *Grover's algorithm: Quantum database search*.
- *How to construct a multi-qubit controlled-z from elementary gates?*
- Philip Strömberg and Vera Blomkvist Karlsson, *4-qubit grover's algorithm implemented for the ibmqx5 architecture*, 46.
- Norbert Schuch and Jens Siewert, *Programmable networks for quantum algorithms*, no. 2, 027902.