IonQHACK 2025 · 2 February 2025

Parallelized QAOA to optimize resource-bound firefighting

Akanksha Chablani[1*], Shannen Espinosa[2*], Corey Ferrier[3*], Aarav Mande[1*]
[1]University of Illinois Urbana-Champaign, Champaign, IL
[2]Northeastern University, Boston, MA
[3]University of Maryland, College Park, MD

Challenge 1: "Parallel Processing" of the Spanning Trees

The score is partially based on the number of CNOT gates used within the circuit. The benefit of fewer gates is reducing the accrued error rate as the circuit progresses. By parallel processing the circuit in the sense of running the circuit on the spanning tree and then the non-spanning tree leftover nodes in two separate circuits, we may have a similar number of total CNOT, but less error. This is because the number of CNOTs *after* register initialization is now much less, up to 50% less than that of the full combined circuit. We say parallel processing, but of course, this is happening in sequence but split into two tasks. Less gates after initialization means less accrued error and also benefits the computation time as seen by our results. The combined circuit would take several times longer than running the circuit two times (on the spanning and non-spanning). We used the CNOT count of the longest of the two circuits to factor into the score since now the error rate contribution by gate count is influenced by the larger of the two circuits.

Challenge 2: Balancing MAX-CUTs

1. When resources are sparse, we wish to have at least approximately equal subsets to distribute resources more efficiently
2. Going beyond the standard max-cut Hamiltonian, we add a balancing term to the Hamiltonian that punishes the algorithm for having unbalanced subsets
3. $H_{total} = H_{max-cut} + \lambda(\sum_{i=1}^{n} Z_i)^2$
4. We chose this form for the balancing term because the sum of Z_i's is zero for a balanced graph whose representative qubit state has the same number of 0's and 1's, but it is either negative or positive for an unbalanced graph. Hence, the square of this term reaches its minimum at zero only when the graph is balanced and is >0 in all other cases

*All authors have contributed equally.

5. Optimizing this Hamiltonian with a large enough lambda makes sure the graph is balanced

Challenge 3: Connected MAX-CUT

1. It is important to note that every connected graph has a minimum-spanning tree within itself. Any separating of vertices into two groups will involve slicing this minimum-spanning tree at least once. To ensure that the subgraphs are connected, we wish to make a maximum of one cuts
2. We implemented this idea by punishing every cut to the minimum spanning tree by increasing the weights of every edge in the minimum spanning tree
3. $H_{connected-max-cut} = \frac{1}{2}|E| - \frac{1}{2}\sum_{(v,w)\in(E/T)} Z_v Z_w + \alpha \sum_{(t,k)\in T} Z_t Z_k$
4. If T is the minimum spanning tree, this algorithms punishes cutting edges of T more than once
5. While this didn't fully work for large graphs just yet, we believe with more time and clever programming, we can make a good approximation algorithm

Unique Graph Generation

With the recent tragic events in Los Angeles California, our group decided to utilize our solution in a manner that could potentially target the wildfire disaster over on the west coast.

1. Taking satellite images from Sentinel 2, Google, and other open-source data repositories provided us with high-resolution map data that we could then process into graphs and feed into our quantum algorithm.
    a. The logic behind this is that one of the troubles faced when fighting fires was the lack of resources (both water and firefighting manpower) present to combat the massive area that the fires impacted. We began to think of whether or not there would be a decent way to properly allocate resources in this circumstance.
2. By thinking of each node on a graph as locations in an area of Los Angeles CA and the edges representing if those nodes were connected via a path of some sort and an arbitrary max distance away, we could apply Max Cut. Every node was assumed to represent a location "on fire" or "at risk" so therefore all needed the firefighting resources.
3. The Max-Cut solution of two groups with the maximum amount of edges between them represents two groups of nodes where we can then apply the few resources to as many nodes in one group such that when the fires are fought at that node, resources would quickly move via edge to the node in the other group.

   a. Challenge 3 would be helpful too as if the groups are strongly connected within, then the few resources can quickly travel within that group and throughout.

4. In `graph8_api.ipynb`, we introduce a Google Maps API-based graph generator that would make a working application of our solution. While we got an API-based application to work, repeatedly calling Google Maps was expensive and we implemented OpenStreetMap and Contextily as free open-source alternatives.