NExt ApplicationS of Quantum Computing

D5.9: Benchmarking of QAOA-based algorithms for mesh segmentation, against K-Means, normalized and randomized cuts, and core extraction methods

Document Properties

Contract Number	951821
Contractual Deadline	30-04-2024
Dissemination Level	Public
Nature	Report
Editors	Yagnik Chatterjee, TotalEnergies, Université de Montpellier
Authors	Yagnik Chatterjee, TotalEnergies, Université de Montpellier Henri Calandra, TotalEnergies Eric Bourreau, Université de Montpellier
Reviewers	Vedran Dunjko, ULEI Alfons Laarman , ULEI Alexis Gais, EVIDEN
Date	18-03-2024
Keywords	Quantum variational algorithms, combinatorial optimization, mesh segmentation
Status	Submitted
Release	0.2

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 951821

History of Changes

Release	Date	Author, Organisation	Description of Changes
0.1	31/01/2024	Yagnik Chatterjee, TotalEnergies, Université de Montpellier Henri Calandra, TotalEnergies Eric Bourreau, Université de Montpellier	Submitted for first review
0.2	18/03/2024	Yagnik Chatterjee, TotalEnergies	Submitted for second review

Table of Contents

1 Executive Summary

This report constitutes the Deliverable 5.9 for Task 5.4, namely the benchmarking of QAOA-based algorithms for mesh segmentation, of the Work Package 5 of the NEASQC Project.

In this report we demonstrate and benchmark a quantum algorithm that solves Quadratic Unconstrained Binary Optimization (QUBO) problems using a logarithmic encoding. In other words, an algorithm that solves a combinatorial optimization problem of size n using $\log_2(n)$ qubits, as opposed to linear scaling algorithms like the Quantum Approximate Optimization Algorithm (QAOA). The algorithm used was initially proposed as a Maximum Cut algorithm (Rančić, 2023). It was eventually generalized for a larger pool of problems (Chatterjee et al., 2023).

A triangular mesh consists of a collection of triangles that together form a surface or solid representation of a 2D or 3D object. Each triangle is defined by its three vertices, and the entire mesh is formed by connecting these vertices. Each node of the mesh has an associated color or property. In this deliverable we will use only 2D triangular meshes. In order to solve the problem of mesh segmentation using our algorithm, we first need to convert it into a graph optimization problem consisting of an objective function and constraints if necessary.

To do this we create a graph from the mesh. The nodes of the graph are the nodes of the mesh and the edges of the graph are the sides of the triangle. The edges are then weighted with a weight equal to the difference in color between the nodes. Hence we have an edge-weighted graph.

Once we have a graph, we need to define an objective function that will help us get the different segments of the mesh. Here we will use the maximization of modularity (Newman, 2006) for meshing. Modularity is a measure of how well a network is split into different groups or clusters. It helps us see if these groups have more connections within themselves than we'd expect just by random chance.

We define the modularity objective function in form of a QUBO problem and then use our algorithm to get the solution.

Despite the algorithm being able generate multiple segments as required, the performance is still quite inferior compared to the classical algorithm. However this serves as a good initial attempt to solve the mesh segmentation problem using our quantum algorithm.

2 Description of the project task

The task for this deliverable is to generate the segmentation of a given mesh using a quantum variational optimization algorithm. In this deliverable we will use our algorithm which we will henceforth refer to as the LogQ encoding. In this chapter we will first describe mesh segmentation, then convert the problem of mesh segmentation into a graph optimization problem. We the proceed to define the objective function for the problem. This objective function is in the form of a Quadratic Unconstrained Binary Optimization (QUBO) in order to be compatible with the LogQ encoding.

2.1 Mesh Segmentation

A mesh is a representation of a surface or object formed by connecting vertices, edges, and faces. Each face of the mesh is defined by the vertices and the edges connecting them. Meshes are widely used in computer graphics and simulations because they provide a simple and efficient way to approximate complex surfaces. The vertices store positional information (x, y, z coordinates), edges connect pairs of vertices, and faces define the surface geometry. Figure 1 (taken from (Shamir, 2008)) gives an example of an object and its corresponding mesh.

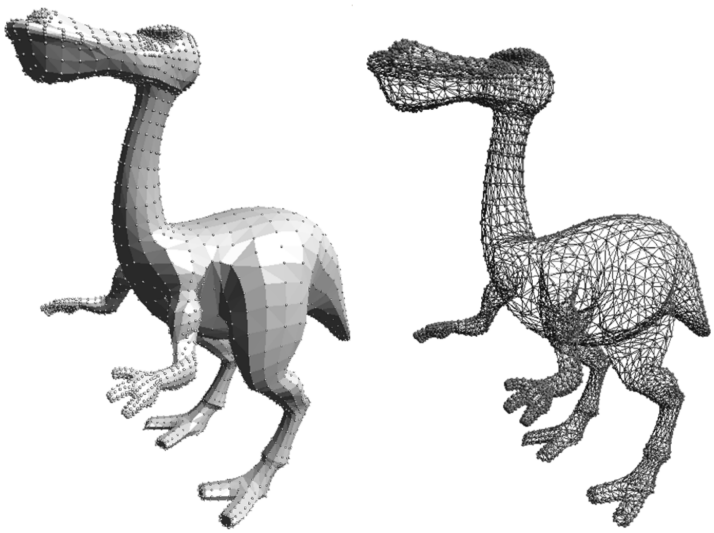

***Figure 1**: An example of a 3D mesh of an object*

Mesh segmentation refers to the process of dividing a complex mesh (composed of vertices, edges, and faces) into meaningful and semantically coherent parts or regions. The goal is to identify and isolate distinct components within the mesh, which may correspond to different objects, substructures, or functional units. Take for example figure 2, wherein by the segmentation of the mesh of the human body, we can decipher the different parts of the body.

Meshes can be in various types, such as triangle meshes, quadrilateral meshes, or polygonal meshes, depending on the type of polygons used to construct the faces. In this deliverable we will deal with only 2D triangular meshes.

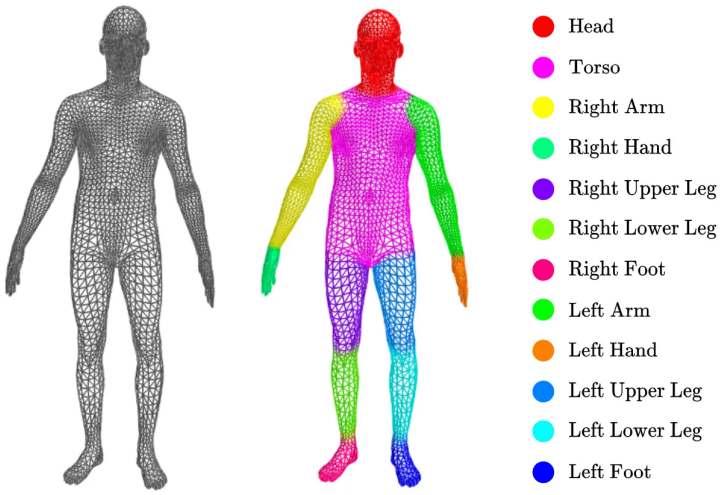

Figure 2: An example for the segmentation of a 3D mesh

2.2 Mesh Segmentation as a graph optimization problem

In order to tackle the problem of mesh segmentation, we first convert it into a graph optimization problem. Figure 3 shows a 2D triangular mesh. It has 64 nodes and every node has a color (or property) associated with it. The color is a number ranging from 10 to 110. In order to translate this problem of mesh segmentation into a graph problem, we create a graph of 64 nodes having the same edges as the edges of the triangular mesh, as shown in Figure 4. Then the edges of the graph are assigned weights equal to the difference in color between the nodes of the edge. The goal is to divide the graph into clusters of nodes. The number of segments or clusters required needs to be specified.

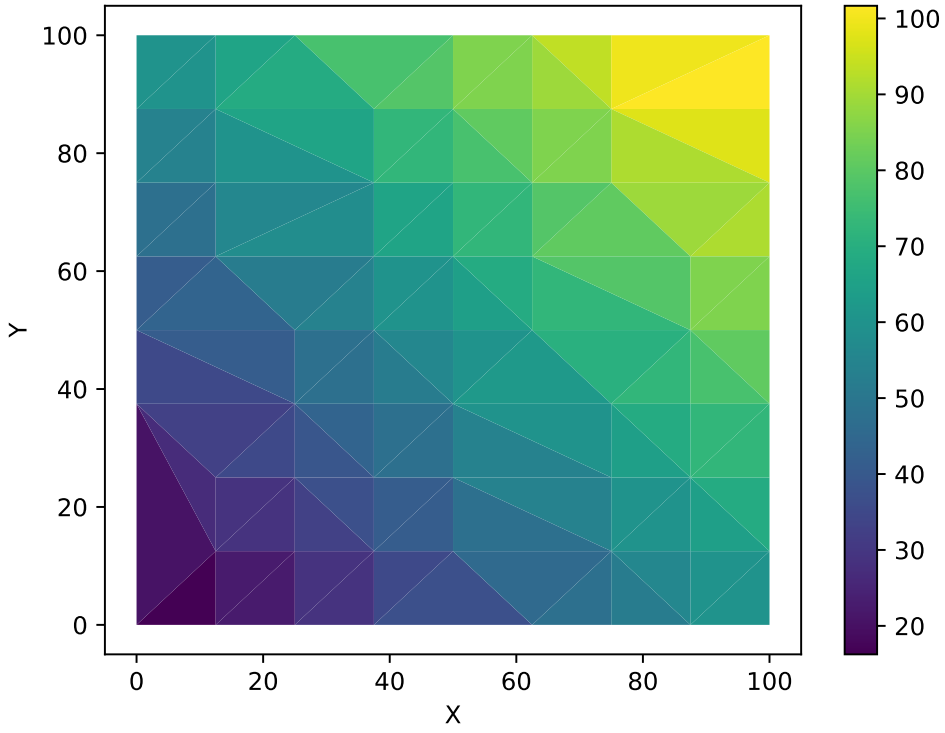

Figure 3: 2D Triangular Mesh of Size 64

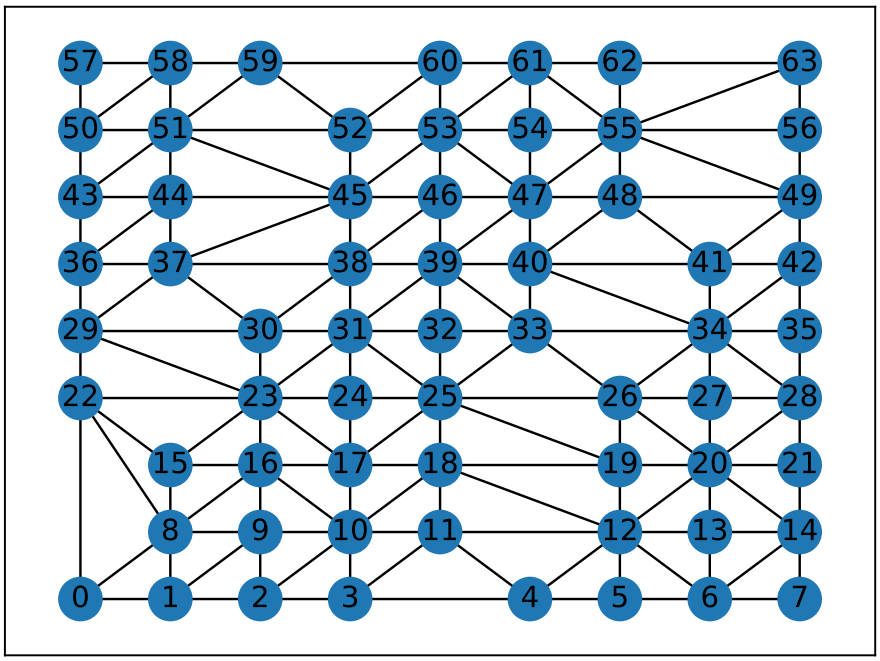

Figure 4: 2D Triangular Mesh of Size 64 as a Graph Problem

2.3 Description of the Objective Function

Once the mesh is represented as an edge-weighted graph, we need to define an objective function to divide it into segments. Here we will use the maximization of modularity (Newman, 2006) as our objective.

Modularity is a measure to evaluate the quality of a partition of a network into communities or clusters (also called communities or groups). The idea of maximizing the modularity is to maximize the number of connections between the nodes within clusters while having sparse connections between nodes in different clusters.

Consider an edge-weighted graph $G(V, E, w)$ with weights $w_{ij}, i, j \in V$. Let the variable $x_i \in \{1, -1\}$ decide whether the node i is in one subgroup or the other. Then its modularity for division into 2 subgroups is defined as follows :

$$M = \frac{1}{4m} \sum_{i,j \in V} \left(A_{ij} - \frac{k_i k_j}{2m} \right) x_i x_j \tag{2.1}$$

where k_i is the weighted degree of node i, A_{ij} is the adjacency matrix of G,

$$A_{ij} = \begin{cases} w_{ij} & \text{if } (i,j) \in E \\ 0 & \text{if } (i,j) \notin E \end{cases} \tag{2.2}$$

$$m = \frac{1}{2} \sum_{i \in V} k_i \tag{2.3}$$

Equation (2.1) is in the form of QUBO can be written as :

$$M = \frac{1}{4m} x^T Q x \tag{2.4}$$

$$Q_{ij} = A_{ij} - \frac{k_i k_j}{2m} \tag{2.5}$$

Q is the required QUBO matrix that can be used in LogQ.

In order to get more than two divisions, we use the algorithm repeatedly on the graph.

3 A Quantum algorithm for solving QUBO problems

NP-hard optimization problems are problems that are expected to not admit algorithms that can give an exact solution in polynomial time, whereas it is 'easy' to verify the solution if it is known. While finding exact solutions to large problems is difficult, there exist many algorithms that find approximate solutions to these problems. In the scope of quantum computing, a huge amount of research has been carried out on hybrid quantum-classical algorithms (Callison & Chancellor, 2022; Cerezo et al., 2021; Farhi et al., 2014; Hadfield et al., 2019; Moll et al., 2018; Nakanishi et al., 2020; Peruzzo et al., 2014; Stokes et al., 2020). In such algorithms, quantum circuit measurements are used in tandem with a classical optimization loop to obtain an approximate solution.

In this deliverable we use the LogQ encoding (Chatterjee et al., 2023), a hybrid quantum-classical algorithm that scales logarithmically with problem size. This means that if the problem size is n then the number of qubits required is of the order of $\log_2 n$. This is in contrast with one of the most commonly used hybrid algorithms, the Quantum Approximate Optimization Algorithm (QAOA) (Farhi et al., 2014), which scales linearly with problem size, meaning that a graph of n nodes would require an n-qubit quantum computer to be solved.

The goal of LogQ is to represent larger problems using a smaller number of qubits, thus potentially achieving faster solutions compared to classical algorithms for sufficiently large instances, while maintaining reliability in the results produced. LogQ was developed in-house in the TotalEnergies Quantum Computing group. It was originally proposed as a MAXIMUM CUT algorithm, which is presented in (Rančić, 2023). Later, we generalized this algorithm for Quadratic Unconstrained Binary Optimization (QUBO) problems in (Chatterjee et al., 2023).

LogQ takes as input a Quadratic Unconstrained Binary Optimization (QUBO) matrix Q as generated in the section 2.3. Then the algorithm proceeds as follows:

1. Let the size of the matrix be $n \times n$, then the number of qubits required will be $N = \lceil \log_2 n \rceil$. A N-qubit parameterized state $|\Psi(\theta)\rangle$ is created as follows:

$$|\Psi(\theta)\rangle = U(\theta) H^{\otimes N} |0\rangle^{\otimes N} \tag{3.1}$$

The equation above represents the application of N Hadamard (H) gates on N qubits, all initially in state $|0\rangle$; followed by the application of a diagonal gate $U(\theta)$ which is of the following form:

$$U(\theta) = \begin{bmatrix} e^{i\pi R(\theta_1)} & 0 & 0 & .... \\ 0 & e^{i\pi R(\theta_2)} & 0 & .... \\ .... & .... & .... & .... \\ 0 & 0 & 0 & e^{i\pi R(\theta_n)} \end{bmatrix} \tag{3.2}$$

where

$$R(\theta_k) = \begin{cases} 0 & \text{if } 0 \leq \theta_k < \pi \\ 1 & \text{if } \pi \leq \theta_k < 2\pi \end{cases} \tag{3.3}$$

Using (3.2) and (3.3) in (3.1), we have:

$$|\Psi(\theta)\rangle = \frac{1}{\sqrt{n}} \begin{bmatrix} e^{i\pi R(\theta_1)} \\ e^{i\pi R(\theta_2)} \\ ..... \\ e^{i\pi R(\theta_n)} \end{bmatrix} \tag{3.4}$$

$|\Psi(\theta)\rangle$ is therefore a vector whose terms belong to the set $\{1, -1\}^n$ subject to the normalization factor of $\frac{1}{\sqrt{n}}$.

2. Since $|\Psi(\theta)\rangle \in \{1, -1\}^n$, we need to define our QUBO matrix using variables $x \in \{1, -1\}$. If the QUBO formulation of the optimization problem is done using variables $x \in \{1, -1\}$ (as is the case for us) we can use the matrix Q in the following steps as is. If the variables used in the formulation are $\{0, 1\}$ binary variables, the QUBO matrix can be converted to the spin-QUBO or sQUBO matrix (Chatterjee et al., 2023) and we can then follow the next steps.

3. Equation (2.4) can have the following quantum equivalent:

$$M(\theta) = -\frac{1}{4m} \langle \Psi(\theta)| Q |\Psi(\theta)\rangle \tag{3.5}$$

where $\theta = \{\theta_1 ... \theta_n\}$ is a set of parameters and $|\Psi(\theta)\rangle$ is a parameterized ansatz that represents that vector x. The interesting thing about this representation is that we need only $\log n$ qubits to represent Q of size $2n \times 2n$. Therefore, only $N = \lceil \log n \rceil$ qubits will be required if we have a problem of size n. We put the negative sign as we want to make it a minimization problem.

4. Measure the expectation value (3.5) on a quantum computer or a simulator. In order to calculate this expectation value we need to decompose Q into a sum of Pauli strings (see Appendix 6.1).

5. Using a classical optimizer such as the genetic algorithm (GA), optimize the parameters θ.

$$M^*(\theta^*) = \min M(\theta) \tag{3.6}$$

From the optimized parameters we can get the binary solution using:

$$x_i = \exp(i\pi R(\theta_i^*)) = \begin{cases} 1 & \text{if } 0 \leq \theta_i^* < \pi \\ -1 & \text{if } \pi \leq \theta_i^* < 2\pi \end{cases} \tag{3.7}$$

4 Results

In this section we show the results obtained for meshes of sizes 32, 64 and 128. The meshes are generated synthetically using packages *numpy* and *matplotlib* on Python. Results of the KMeans clustering method (Jin & Han, 2010) are used as a benchmark. KMeans clustering is also an NP-Hard problem and therefore has exponential complexity. The KMeans algorithm of the *scikit-learn* Python package is used here. This package uses Lloyd's algorithm (Lloyd, 1982) as the default algorithm and this is what was used here. The worst case complexity is given by $O(n^{\frac{k+2}{p}})$ where n is the number of nodes, k is the number of segments and p is the number of attributes. Here we have only one attribute for every node (color) so $p = 1$.

The quantum simulations as well as the classical algorithm runs are carried out using the following computational resources: Chip: Apple M2 Pro, RAM: 32 GB. For the LogQ runs, the meshes of sizes 32, 64 and 128 required 5, 6 and 7 qubits respectively. The genetic algorithm is used as the classical optimizer for the variaitonal loop in LogQ. The parameters of the genetic algorithm such as the *population size* and the *maximum number of iterations* have an impact on the quality of the solutions and the execution time. In order to choose these parameters, one must consider a balance between execution time and the quality of solution. Increasing the population size and number of iterations will improve the solution up to a certain point after which it will stabilize. Conversely, augmenting these parameters also leads to a linear increase in execution time. In our case, both the population size and maximum number of iterations are set to 40.

In figures 5 and 6 we can visualize the mesh segmentation results for the instance of size 128 for 8 segments, using KMeans and the quantum algorithm respectively. We can see that in KMeans it is perfectly separated into 8 different clusters while for the quantum algorithm it is less consistent, with some of the clusters being a bit scattered. This might be due to the repeated bi-partitioning and sometimes the algorithm has no choice but to split the clusters.

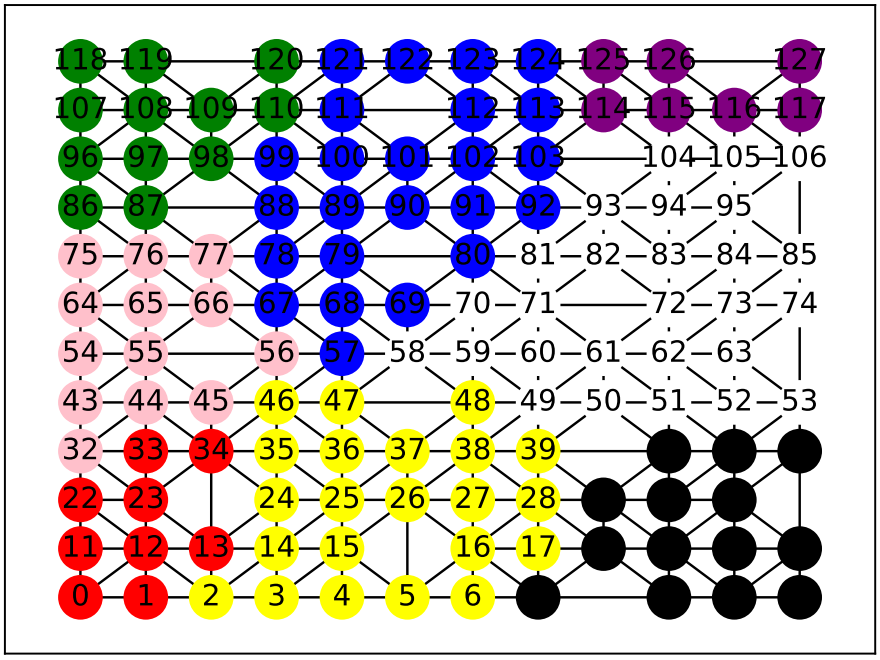

***Figure 5**: Mesh Segmentation of 128-Node Mesh into 8 segments using KMeans*

The benchmarks shown in tables 1, 2 and 3, namely cut discrepancy (Huang & Dom, 1995), consistency error (Martin et al., 2001), adjusted Rand index (Rand, 1971) and Hamming distance (Huang & Dom, 1995) are generated by comparing the segments generated by the quantum algorithm and the segments generated by KMeans. For every

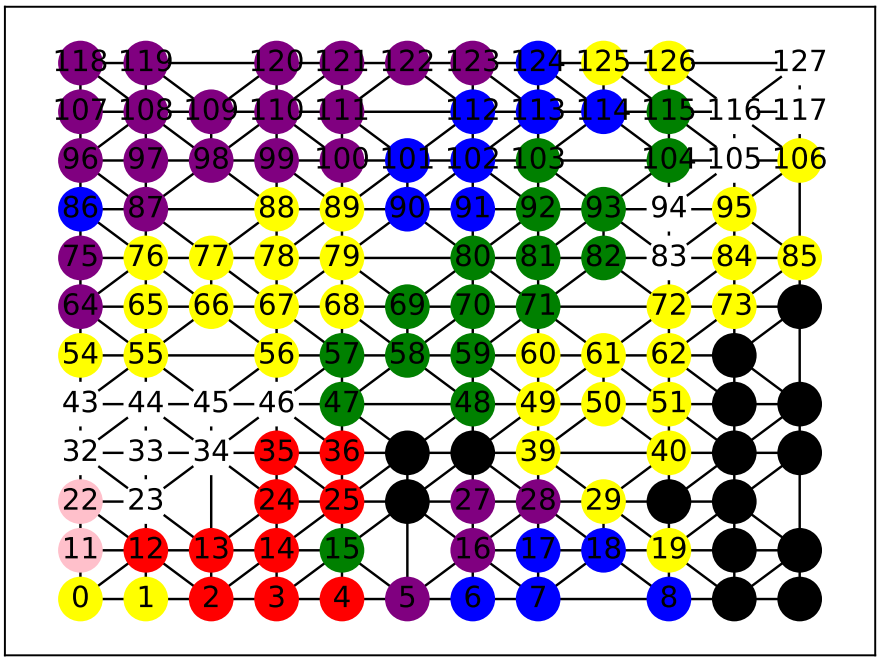

***Figure 6**: Mesh Segmentation of 128-Node Mesh into 8 segments using LogQ*

instance, we generate the segments using both LogQ and KMeans. Then, we consider the KMeans solution to be the true solution and the LogQ solution to be the predicted solution. Note that KMeans does not necessarily provide the optimal solution, but we consider it as a classical benchmark.

For cut discrepancy, consistency error and hamming distance, smaller values are better while for rand index, larger is better. The Rand index ranges from 0 to 1, where 1 indicates perfect agreement between the two clusters, and 0 indicates no agreement beyond what would be expected by random chance. Consistency error quantifies the discrepancy or inconsistency between clusters from different partitions. It ranges from 0 (perfect consistency) to 1 (complete inconsistency). Cut discrepancy is the difference in normalized cuts between the two clusters. It varies from 0 to a positive number, 0 being the best.

Hamming distance measures the number of points of disagreement between 2 vectors. The range of the Hamming distance depends on the length of the vectors being compared. If the concerned mesh has a size n, then the Hamming distance can range from 0 to n, inclusively.

Cut discrepancy depends heavily on the accuracy of boundaries. This is why as the number of segments increases, it becomes harder to maintain a low cut discrepancy as the precision of the boundary recognition plays an important role. For all 3 mesh sizes, the cut discrepancy is close to 0 for $k = 2$ and gets significantly worse as we increase the number of segments. This indicates that both KMeans and LogQ find a very similar boundary for $k = 2$ and then the boundaries of the segments grow more and more different.

Rand index measures the likelihood that a pair of faces are either in the same segment in both clusters, or in different segments in both clusters. Note that rand index is actually getting better with higher number of segments. Therefore with increasing number of segments while segment boundaries are not as consistent, as demonstrated by the cut discrepancy, the number of pairs belonging to the same segment increases.

Finally, Hamming distance and consistency error give rather poor results. The consistency error is around 0.9 for almost all the instances (with 1 being the worst). As for Hamming distance, for a vector of size 128, we see disagreements on 100 and 107 elements respectively for $k = 4$ and $k = 8$, while it is a bit better for $k = 2$. Similar values are found for sizes 32 and 64 as well. This could to be due to the fact that Hamming distance and consistency error

depend on the labeling of the segmentations. This means that, despite the segments being the same, it could be that in one of the segments it is labeled 1 and in another 2, resulting in poor benchmark values.

The main takeaway from the benchmarks is that the method works well for bi-partition but the segment boundaries get worse with an increase in k. Further work is therefore required to improve the consistency of the segment boundaries as the number of segments increase.

Number of Segments	Cut Discrepancy	Consistency Error	Adjusted Rand Index	Hamming Distance
2	0.004	0.95	0.22	8
4	0.08	0.83	0.4	23
8	0.77	0.77	0.62	28

***Table 1**: 32-Node Mesh Segmentation.*

Number of Segments	Cut Discrepancy	Consistency Error	Adjusted Rand Index	Hamming Distance
2	0.002	0.86	0.05	24
4	0.03	0.83	0.17	56
8	0.36	0.93	0.37	54

***Table 2**: 64-Node Mesh Segmentation.*

Number of Segments	Cut Discrepancy	Consistency Error	Adjusted Rand Index	Hamming Distance
2	0.0003	0.86	0.07	46
4	0.012	0.92	0.11	100
8	0.32	0.96	0.15	107

***Table 3**: 128-Node Mesh Segmentation.*

While all the algorithm runs above were carried out using a quantum simulator, the effect of noise of an actual quantum computer (from IBM) on the algorithm can be found in (Chatterjee et al., 2023). In addition, a study demonstrating the time taken by the quantum simulator versus the time taken to solve the same instance on real hardware is shown in the appendix of (Chatterjee et al., 2023). Even though this study is on another problem, this can help shed some light regarding how the runtime of the algorithm evolves with increasing size.

5 Conclusions

In this deliverable we demonstrate how we can carry out mesh segmentation using a quantum algorithm. Using this algorithm we are able to perform mesh segmentation of size n using $\lceil \log_2 n \rceil$ qubits. This allows us to represent meshes of sizes up to 128 using only 7 qubits, something that would have taken 128 qubits with QAOA. This would mean that with 30 qubits we can reach a mesh size of 1 billion which is around the industrial size of meshes. However, there are certain challenges that need to be faced in order to do this. The main challenge is that of the overhead due to Pauli decomposition. As shown in appendix 6.1, we need to calculate the coefficients of n^2 terms where n is the size of the mesh. For a mesh of size 1 billion, we will need to calculate 10^{18} terms. There is already some interesting research to mitigate this issue. For example, much more efficient schemes to decompose the Hamiltonian and calculate the expectation value exist but only for d-sparse matrices (Bravo-Prieto et al., 2023).

We benchmark LogQ against the well-known KMeans algorithm. Although the quantum algorithm is able to find clusters, the quality of the results is far below that of KMeans as the number of segments increase. This is understandable for several reasons. One is the choice of objective function. A more appropriate and optimized objective function can improve performance significantly. An issue with the current objective function is that it can only bi-partition a mesh into two clusters and therefore requires multiple iterations to get multiple clusters. This could induce errors in the segmentation. This could be remedied by using an objective function that works for any number of segments and not just two segments. A further work is therefore to tackle the issue with a modularity generalization or using another measure like within-cluster variance.

Secondly, the performance also depends on the population size and number of the iterations of genetic algorithm in our algorithm and hence on computation time. The computation time increases linearly with the increase in either population size or number of iterations or both. As mentioned in the previous section, it is important to find the right balance between the performance and the computation time.

Nevertheless this deliverable serves as a first proof of concept on how the problem of mesh segmentation can be tackled using a quantum variational algorithm.

List of Acronyms

Term	Definition
QAOA	Quantum Approximate Optimization Algorithm
QUBO	Quadratic Unconstrained Binary Optimization

***Table 4**: Acronyms and Abbreviations*

List of Figures

List of Tables

Bibliography

Bravo-Prieto, C., LaRose, R., Cerezo, M., Subasi, Y., Cincio, L., & Coles, P. J. (2023). Variational quantum linear solver. *Quantum*, *7*, 1188.

Callison, A., & Chancellor, N. (2022). Hybrid quantum-classical algorithms in the noisy intermediate-scale quantum era and beyond. *Phys. Rev. A*, *106*, 010101. https://doi.org/10.1103/PhysRevA.106.010101

Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S. C., Endo, S., Fujii, K., McClean, J. R., Mitarai, K., Yuan, X., Cincio, L., et al. (2021). Variational quantum algorithms. *Nature Reviews Physics*, *3*(9), 625–644. https://www.nature.com/articles/s42254-021-00348-9

Chatterjee, Y., Bourreau, E., & Rančić, M. J. (2023). Solving various np-hard problems using exponentially fewer qubits on a quantum computer. *arXiv preprint arXiv:2301.06978*.

Farhi, E., Goldstone, J., & Gutmann, S. (2014). A quantum approximate optimization algorithm. *arXiv:1411.4028*. https://arxiv.org/abs/1411.4028

Hadfield, S., Wang, Z., O'Gorman, B., Rieffel, E. G., Venturelli, D., & Biswas, R. (2019). From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms*, *12*(2). https://doi.org/10.3390/a12020034

Huang, Q., & Dom, B. (1995). Quantitative methods of evaluating image segmentation. *Proceedings of the 1995 International Conference on Image Processing (Vol. 3)-Volume 3 - Volume 3*, 3053.

Jin, X., & Han, J. (2010). K-means clustering. In C. Sammut & G. I. Webb (Eds.), *Encyclopedia of machine learning* (pp. 563–564). Springer US. https://doi.org/10.1007/978-0-387-30164-8_425

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, *28*(2), 129–137. https://doi.org/10.1109/TIT.1982.1056489

Martin, D., Fowlkes, C. C., Tal, D., & Malik, J. (2001). A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. *2*, 416–423. https://doi.org/10.1109/ICCV.2001.937655

Moll, N., Barkoutsos, P., Bishop, L. S., Chow, J. M., Cross, A., Egger, D. J., Filipp, S., Fuhrer, A., Gambetta, J. M., Ganzhorn, M., et al. (2018). Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Science and Technology*, *3*(3), 030503. https://iopscience.iop.org/article/10.1088/2058-9565/aab822

Nakanishi, K. M., Fujii, K., & Todo, S. (2020). Sequential minimal optimization for quantum-classical hybrid algorithms. *Physical Review Research*, *2*(4), 043158. https://journals.aps.org/prresearch/abstract/10.1103/PhysRevResearch.2.043158

Newman, M. E. J. (2006). Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, *103*(23), 8577–8582. https://doi.org/10.1073/pnas.0601602103

Peruzzo, A., McClean, J., Shadbolt, P., Yung, M.-H., Zhou, X.-Q., Love, P. J., Aspuru-Guzik, A., & O'brien, J. L. (2014). A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, *5*(1), 1–7. https://www.nature.com/articles/ncomms5213

Rančić, M. J. (2023). Noisy intermediate-scale quantum computing algorithm for solving an n-vertex maxcut problem with log (n) qubits. *Physical Review Research*, *5*(1), L012021.

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, *66*(336), 846–850. https://doi.org/10.1080/01621459.1971.10482356

Shamir, A. (2008). A survey on mesh segmentation techniques. *Computer graphics forum*, *27*(6), 1539–1556.

Stokes, J., Izaac, J., Killoran, N., & Carleo, G. (2020). Quantum natural gradient. *Quantum*, *4*, 269. https://quantum-journal.org/papers/q-2020-05-25-269/

6 Appendix

6.1 Calculating the Expectation value of an observable

Given the observable $\mathcal{O}$, we first need to convert into a sum of tensor products of Pauli strings.

Let $\mathcal{O}$ be a $n \times n$ matrix and $S = \{I, X, Y, Z\}^N = \{S_1, S_2, S_3, S_4\}^N$ be the set of Pauli matrices. We can consider n to be a power of 2 without any loss of generality. If the size of the Hamiltonian matrix is n' which is not a power of 2, we can easily convert it to a size of $n = 2^{\log_2(n')}$, which is a power of 2. The extra space in the matrix is filled with 0's.

This Hamiltonian can now represented on $N = \log_2(n)$ qubits. Consider the set $J = \{S_{i_1} \otimes S_{i2} ... \otimes S_{iN} | i_1, i_2 .... i_N \in \{0, 1, 2, 3\}\}$ which consists of all tensor product combinations of the Pauli matrices.

Then the Hamiltonian can decomposed as:

$$\mathcal{O} = \sum_{i=1}^{4^N} c_i J_i \tag{6.1}$$

where the coefficients are:

$$c_i = \frac{1}{n} Tr(J_i \cdot \mathcal{O}) \tag{6.2}$$

The Hamiltonian therefore becomes:

$$\mathcal{O} = \frac{1}{n} \sum_{i=1}^{4^N} Tr(J_i \cdot \mathcal{O}) J_i \tag{6.3}$$

Therefore we need to calculate $4^N = 4^{log_2 n} = n^2$ terms in order to get the full decomposition of the observable into Pauli matrices. The expectation value then becomes a sum of the expectation values of all the terms.

$$\langle \Psi | \mathcal{O} | \Psi \rangle = \frac{1}{n} \sum_{i=1}^{4^N} Tr(J_i \cdot \mathcal{O}) \langle \Psi | J_i | \Psi \rangle \tag{6.4}$$