Multiway Number Partitioning

Part I : Classical & Quantum Annealing

Lai, Chia-Tso

Fraunhofer
IISB

Problem Formulation

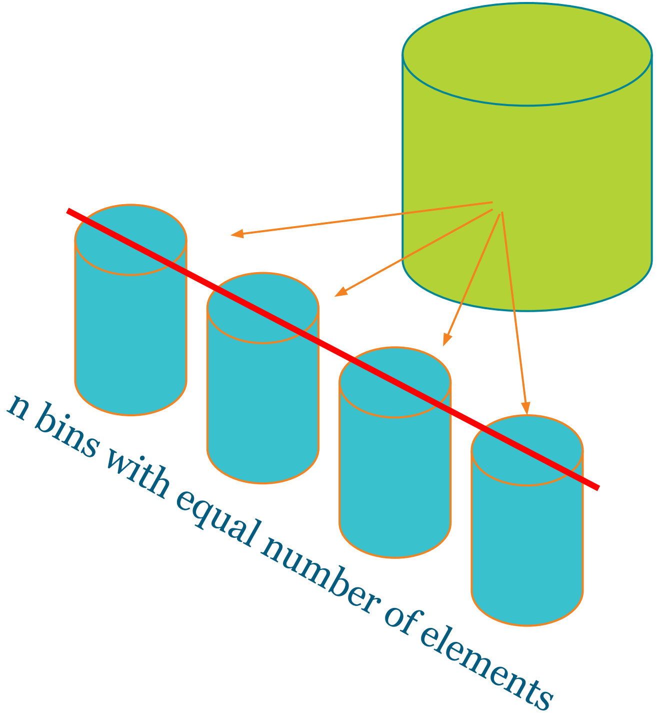
An array with N weighted elements **W**=[w1,w2,w3,...,wN]

Assign the elements uniformly to n bins

Compute the total weight of each bin

Objective:

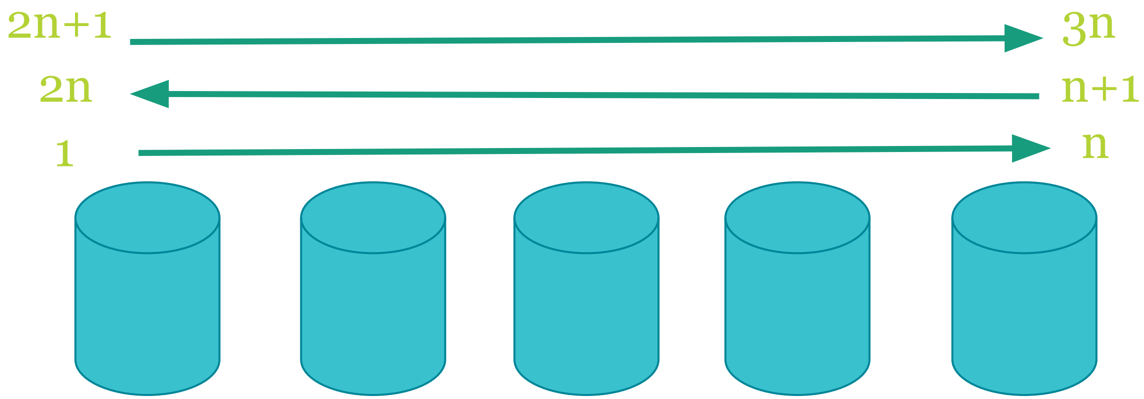
Minimize the standard deviation of the weights



Heuristic Approach

Serpentine draft

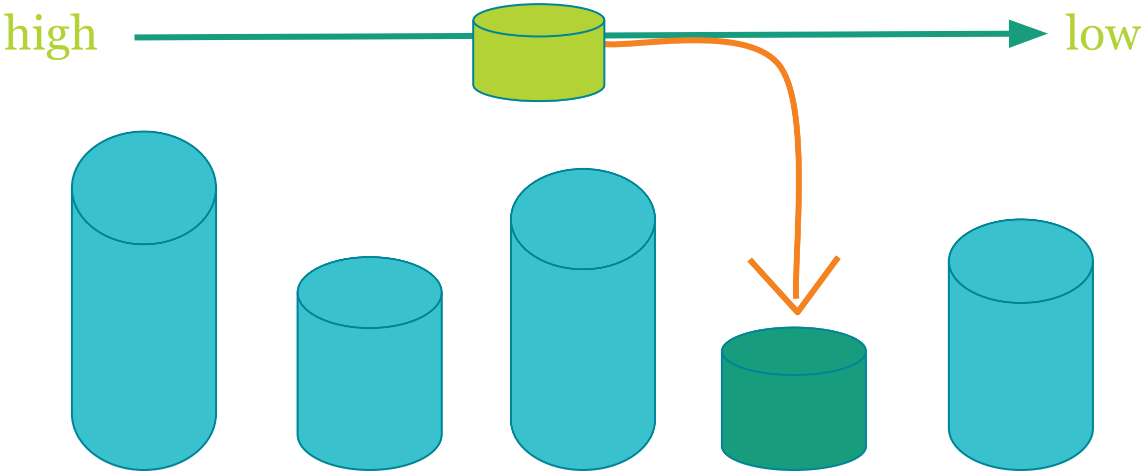
1. Sort the elements from low to high weights
2. Order of assignment alternates between ascent and descent



Fraunhofer IISB

Greedy Number Partitioning

1. Sort the elements from high to low weights
2. Assign the next element to the bin (not full yet) with the lowest current sum



Fraunhofer IISB

Quantum Annealing Optimization

Define binary variables $\mathbf{X}_{ij}$ $i=0,1,\ldots,n \quad j=0,1,\ldots,N$

i labels the bin while j labels the element

If the jth element is assigned to the ith bin $\Rightarrow$ $\mathbf{X}_{ij}=1$ otherwise, $\mathbf{X}_{ij}=0$

Cost Function:

$$f(\mathbf{X}) = \Sigma_i(\underbrace{\boxed{\Sigma_j \mathbf{X}_{ij} \mathbf{W}_j}}_{\text{sum of each bin}} - \underbrace{\boxed{\Sigma \mathbf{W}_i / N}}_{\text{mean}})\hat{\ }2$$

Constraints:

each element only assigned once $\boxed{\Sigma_i \mathbf{X}_{ij} = 1}$

$\boxed{\Sigma_j \mathbf{X}_{ij} = N/n}$ Each bin has the equal number of element

Quantum Annealing Optimization

1. Encode the cost function and constraints as **ConstrainQuadraticModel**
2. Use **LeapHybridCQMSampler** to sample the solutions
3. Take the lowest energy sample among all feasible samples (samples which satisfies the constraints)
4. Convert the solution matrix **Xij** into n arrays of element indices

Example 1: 20 elements to 4 bins

W = [3.2, 0.33, 6.1, 5.5, 3.17, 4.3, 0.1, 0.1, 2.5, 10.8,

0.06, 0.15, 4, 3, 0.14, 1, 0.25, 4, 0.22, 1.08]

(element values are not of equal order of magnitude)

Optimizer	Heuristic	Greedy	Quantum Annealer
Sum of each bin	[9.69, 10.93, 12.12, 17.26]	[12.53, 12.5 , 12.5 , 12.47]	[12.5, 12.5, 12.5, 12.5]
standard deviation	2.88	0.0212	0

Example 2: 40 elements to 5 bins

W = [19.9, 12.111, 0.009, 3.5, 4.08, 0.3, 0.01, 0.09, 30.234, 5.123, 1.643, 0.007, 0.013, 1.9, 1, 0.08, 0.2, 0.5, 0.299, 0.001, 37.75, 1, 0.2, 0.05, 5.5, 4.5, 3.7, 6.3, 0.004, 15.996, 2, 2, 10.5, 10.91, 12, 0.15, 2.33, 4.01, 0.07, 0.03]

(element values are not of equal order of magnitude)

Optimizer	Heuristic	Greedy	Quantum Annealer
Sum of each bin	[50.087, 44.258, 37.737, 33.985, 31.933]	[39.999, 40.007, 40, 39.996, 39.998]	[40.003, 39.994, 40.002, 39.999, 40.002]
standard deviation	7.36	0.0037	0.0033

Example 3: 100 elements to 10 bins

W = numpy.random.random(100)

(element values are mostly of equal order of magnitude)

Optimizer	Heuristic	Greedy	Quantum Annealer
standard deviation	0.0276	0.027	0.01856

Example 4: 1000 elements to 20 bins

W = numpy.random.random(1000)

(element values are mostly of equal order of magnitude)

Optimizer	Heuristic	Greedy	Quantum Annealer
standard deviation	0.014	0.002	0.036
Runtime	0.001 secs	0.007 secs	13 mins

Example 5: 400 elements to 100 bins

W = numpy.ones(400)+numpy.random.normal(0,1,400)

(element values sampled from unity+noise of normal distribution with mean=0 & variance=1)

Optimizer	Heuristic	Greedy	Quantum Annealer
standard deviation	0.16	3.3	0.16

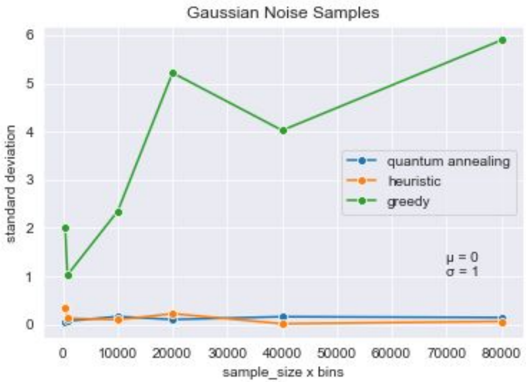
Example 6: 800 elements to 100 bins

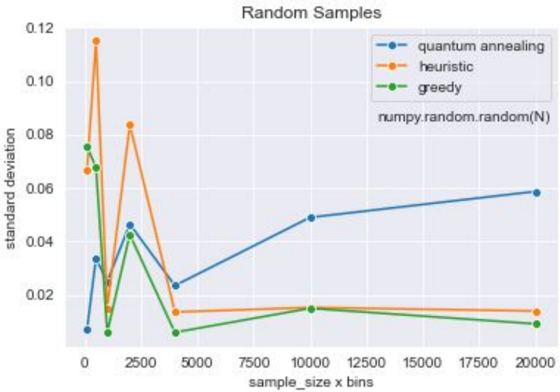
W = numpy.ones(800)+numpy.random.normal(0,1,800)

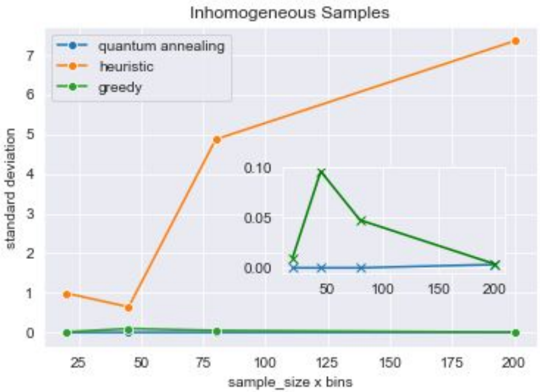
(element values sampled from unity+noise of normal distribution with mean=0 & variance=1)

Optimizer	Heuristic	Greedy	Quantum Annealer
standard deviation	0.063	5.9	0.144

Benchmarking







Summary

- **Quantum annealing**'s performance is robust and consistent across different sample types and sizes. However, the **runtime** can scale up quickly as the qubits number scales up as Nxn.
- **Heuristic** (serpentine draft) algorithm performs well and extremely fast across all sample sizes. However, for certain samples containing elements with **uneven order of magnitude**, it performs less well compared to quantum annealing.
- **Greedy Number Partitioning** also performs extremely well and fast in most cases except when the samples follow a **normal distribution**.

Multiway Number Partitioning

Part II : Gate-based Quantum Solution

Lai, Chia-Tso

Fraunhofer
IISB

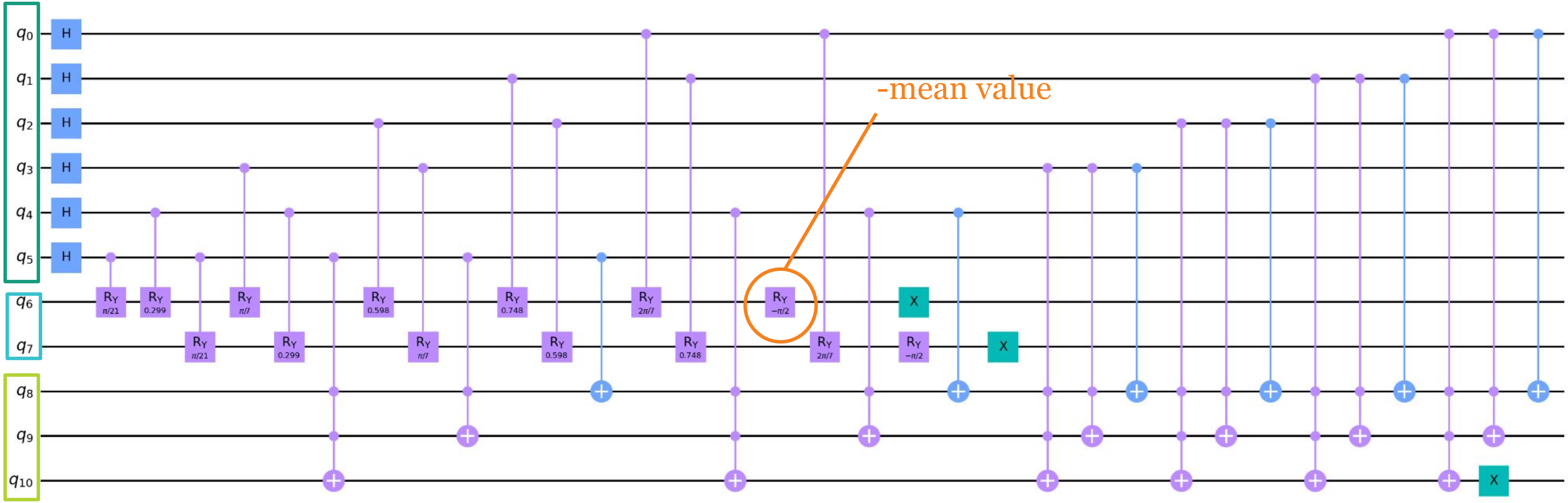
Grover's Search Method

1. At least $\mathbf{N+log_2(N)+1}$ qubits required ($\log_2(N)+1$ ancilla qubits)
2. Binary variables $\mathbf{X_i}$ represent the elements ($i = 1,2,..,N$)
3. Initialize the state by applying Hadamard gates to every element qubit
4. Set up the objective by **C-Ry** and **Ry** gates. The rotational angles of C-Ry gates correspond to the elements' weights (mapped to $[0, \pi]$), while the angle of Ry gate encodes the **mean value** of each bin
5. The constraint of N/n elements in each bin is imposed by **CNOT** gates, which implement addition of 1 if the element qubit is chosen ($\mathbf{X_i = 1}$)
6. Amplify the amplitudes of the desired states with Grover's operator:

$$\mathbf{G = AS_0A^{\dagger}S_x}$$

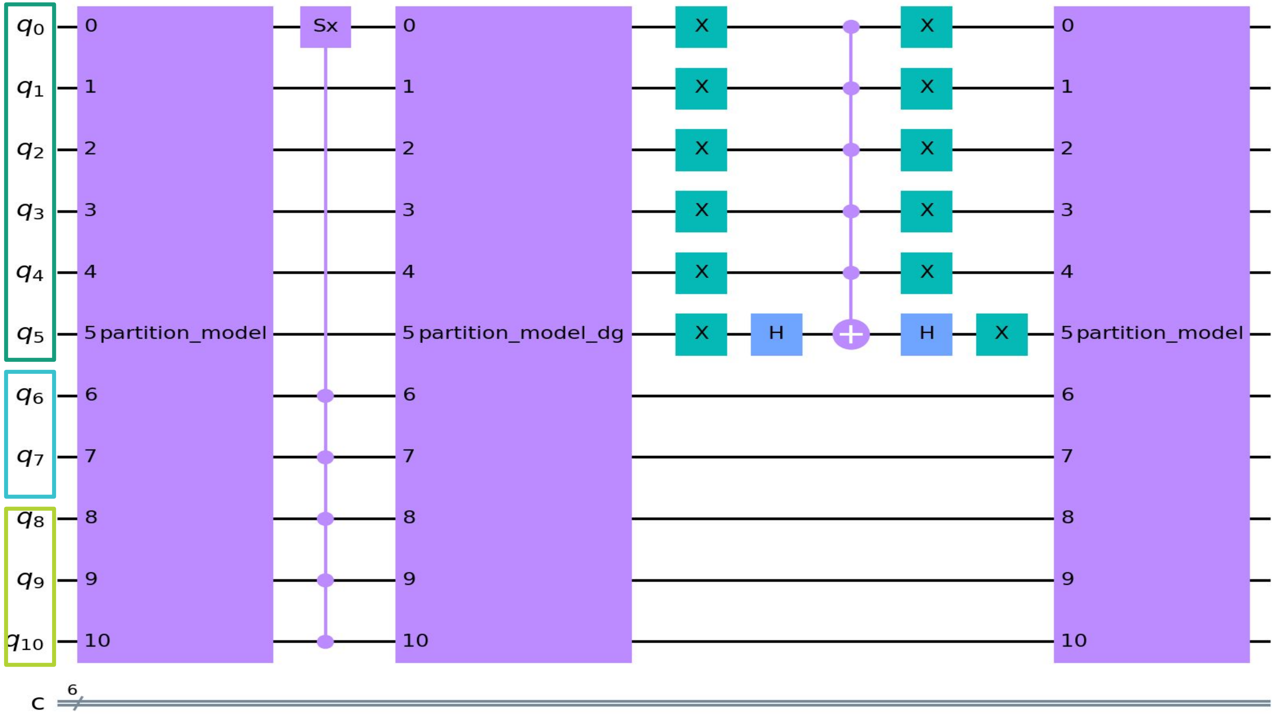
Quantum Number Partitioning Model

Example: 6 elements [1,2,3,4,5,6] into 2 bins



element qubits : 6 objective qubits : 2 constraint qubits : 3

Grover’s Operator



element qubits : 6
objective qubits : 2
constraint qubits : 3

Post Processing

1. Extract the top measurement results
2. Filter out results that violate the constraint
3. Pick the best solution (closest to mean value) and keep the remaining elements
4. Execute another round of Grover'search with the remaining elements. Namely, assigning N-N/n elements to n-1 bins
5. After **n-1** rounds of search, a set of solution could be formed
6. Improve the solution by pairing up two bins according to their deviations from the mean and then exchange certain elements such that the difference of two bins got reduced

Implementation

W = np.ones(N) + np.random.normal(0,1,N)

(element values sampled from unity+noise of normal distribution with mean=0 & variance=1)

Samples	Heuristic	Grover's Search
9 elements to 3 bins	0.199	0.079
16 elements to 4 bins	0.2095	0.018
20 elements to 5 bins	0.535	0.212
22 elements to 2 bins	0.39	0.025

Implementation

W = np.random.random(N)

(samples are random numbers between 0 and 1)

Samples	Heuristic	Grover's Search
12 elements to 6 bins	0.1032	0.1046
14 elements to 7 bins	0.065	0.071
20 elements to 4 bins	0.0512	0.0469
22 elements to 2 bins	0.0364	0.0042

Summary

- Number Partitioning by Grover's search works really well in both sample types on a small scale
- The number of deployable qubits is currently limited. The maximum problem size is 22 elements using the qasm simulator
- Questionable potential quantum speedup due to the qubit number O(N)
- Accuracy can be improved with more ancilla qubits and precise rounds of amplitude amplification