Quantum Approximate Optimization Algorithms - Knapsack Problem

Esther Wang
AEOP Undergraduate Apprentice

Knapsack Problem - Background

- Combinatorial optimization problem
  - Finding an optimal object from a finite set of objects
- Given a knapsack with a limited capacity
- Given a finite set of items
  - Weight
  - Value

https://medium.com/algobox/knapsack-problems-4ccee7c295ea

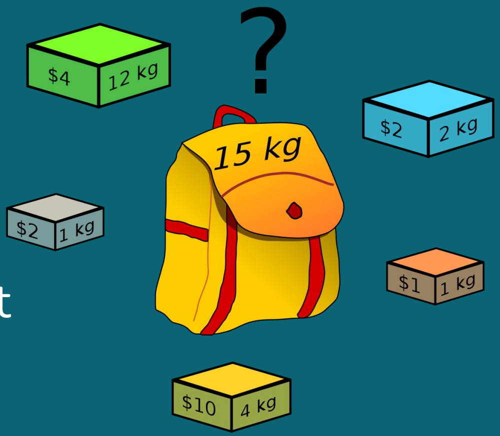
Knapsack Problem - Background

- Determine a subset of items to include in the knapsack
  - Total weight is less than or equal to the capacity of the knapsack
  - Total value of the items is as large as possible
  - Example
    - 4kg ($10) + 2kg ($2) + 1kg ($1) + 1kg ($1)
    - = 8kg < 15kg
    - Total value is $15, which is the max value

https://medium.com/algobox/knapsack-problems-4ccee7c295ea

Knapsack Problem - Background

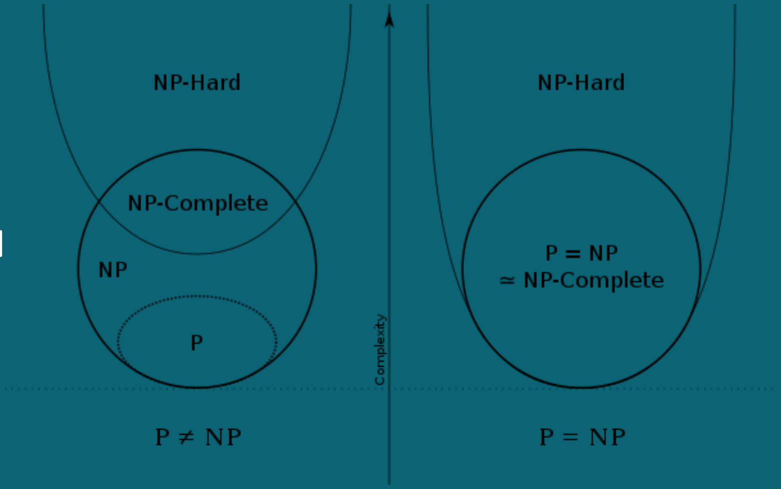
- Derived from a commonplace problem of packing the most valuable items without overloading the luggage
- Often appears in real-word decision-making processes
  - Example
    - Resource allocation problem
    - Given a set of non-divisible tasks
      - Under a fixed budget or time constraint



https://en.wikipedia.org/wiki/Knapsack_problem

Knapsack Problem - Computational Complexity

- Decision problem form of knapsack
  - NP (non-deterministic polynomial-time) Complete
    - Brute-force algorithm by testing all possible cases to find the solution
    - Correctness of the solution can be verified in polynomial time
- Optimization problem form of knapsack
  - Not NP Complete
  - No known polynomial algorithm that can tell whether a given solution is optimal



https://commons.wikimedia.org/wiki/File:P_np_np-complete_np-hard.svg

Classical Solutions

- Classical Algorithms
  - Brute force recursion
  - Dynamic programming
    - Optimization method to solve a class of problems that have overlapping subproblems
  - Branch and bound
    - Algorithm that explores the entire search space to find the optimal solution
  - Hybridization of dynamic programming and branch and bound

Classical Solution - Dynamic Programming

- Dynamic programming solution

```python
def knapSack(w, wt, val, n):
    K = [[0 for x in range(w+1)] for y in range(2)]

    # We know we are always using the  current row or
    # the previous row of the array/vector . Thereby we can
    # improve it further by using a 2D array but with only
    # 2 rows i%2 will be giving the index inside the bounds
    # of 2d array K
    for i in range(n + 1):
        for w in range(w + 1):
            if (i == 0 or w == 0):
                K[i % 2][w] = 0
            elif (wt[i - 1] <= w):
                K[i % 2][w] = max(
                    val[i - 1]
                    + K[(i - 1) % 2][w - wt[i - 1]],
                    K[(i - 1) % 2][w])

            else:
                K[i % 2][w] = K[(i - 1) % 2][w]

    return K[n % 2][w]
```

Quantum Approximate Optimization Algorithm

- Algorithm that finds approximate solutions to combinatorial optimization problems
- Performs better than classical computers
- Use the quantum superposition states to compute solutions faster
    - Apply to many possible inputs simultaneously

Unitary Quantum Operators

$$|\beta, \gamma\rangle \equiv U(B, \beta_p)U(C, \gamma_p).....U(B, \beta_1)U(C, \gamma_1)\,|s\rangle$$

where $|\psi_0>$ is a suitable initial state

- $U(\beta,\gamma)$ Unitary is characterized by its parameters to prepare a quantum state
- Composed of $U(\beta)=e^{-i\beta HB}$ and $U(\gamma)=e^{-i\gamma HP}$ where H_B is the mixing Hamiltonian and H_P is the problem Hamiltonian
- Quantum state is prepared by applying the unitaries as alternating blocks applied p times
- Find the optimal parameters so that the quantum state encodes the solution

https://arxiv.org/pdf/1908.02210.pdf

Knapsack Cost Function

$$C(X') = M(W_{max} - sum_{i=0}^{n-1} x_i w_i - S)^2 - sum_i^{n-1} x_i v_i$$

where S = sum(2^j * y[j]), j goes from n to n + log(W_{max}). M is a number large enough to dominate the sum of values.

The minimum value will be where the constraint is respected and the sum of the values are maximized.

https://qiskit.org/documentation/stable/0.24/stubs/qiskit.optimization.applications.ising.knapsack.html

Hamiltonian Operator

- Hamiltonian mapped from the cost function

General function form: $H = H_A + H_B,$

Formula: $$H_A = A\left(1 - \sum_{n=1}^{W} y_n\right)^2 + A\left(\sum_{n=1}^{W} n y_n - \sum_{\alpha} w_\alpha x_\alpha\right)^2$$

$$H_B = -B \sum_{\alpha} c_\alpha x_\alpha.$$

https://www.frontiersin.org/articles/10.3389/fphy.2014.00005/full

```python
prob = Knapsack(values=[3,4,5,6,7], weights=[2,3,4,5,6], max_weight=10)
qp = prob.to_quadratic_program()
print(qp.prettyprint())
```

```
Problem name: Knapsack

Maximize
  3*x_0 + 4*x_1 + 5*x_2 + 6*x_3 + 7*x_4

Subject to
  Linear constraints (1)
    2*x_0 + 3*x_1 + 4*x_2 + 5*x_3 + 6*x_4 <= 10  'c0'

  Binary variables (5)
    x_0 x_1 x_2 x_3 x_4
```

```python
# QAOA
qins = QuantumInstance(backend=Aer.get_backend("aer_simulator"), shots=100)
meo = MinimumEigenOptimizer(min_eigen_solver=QAOA(reps=1, quantum_instance=qins))
result = meo.solve(qp)
print(result.prettyprint())
print("\nsolution:", prob.interpret(result))
print("\ntime:", result.min_eigen_solver_result.optimizer_time)
```

```
objective function value: 13.0
variable values: x_0=1.0, x_1=1.0, x_2=0.0, x_3=1.0, x_4=0.0
status: SUCCESS

solution: [0, 1, 3]

time: 1.8710551261901855
```

The Quantum Approximate Optimization Algorithm simulated on a specific knapsack problem with weights and values.

```
# intermediate QUBO form of the optimization problem
conv = QuadraticProgramToQubo()
qubo = conv.convert(qp)
print(qubo.prettyprint())
```

```
Problem name: Knapsack

Minimize
  26*c0@int_slack@0^2 + 104*c0@int_slack@0*c0@int_slack@1
  + 208*c0@int_slack@0*c0@int_slack@2 + 156*c0@int_slack@0*c0@int_slack@3
  + 104*c0@int_slack@1^2 + 416*c0@int_slack@1*c0@int_slack@2
  + 312*c0@int_slack@1*c0@int_slack@3 + 416*c0@int_slack@2^2
  + 624*c0@int_slack@2*c0@int_slack@3 + 234*c0@int_slack@3^2
```

```
# qubit Hamiltonian and offset
op, offset = qubo.to_ising()
print(f"num qubits: {op.num_qubits}, offset: {offset}\n")
print(op)
```

```
num qubits: 9, offset: 1417.5

-258.5 * IIIIIIIIZ
- 388.0 * IIIIIIIZI
- 517.5 * IIIIIIZII
- 647.0 * IIIIIZIII
```

Generating Hamiltonian and the result printing Pauli operators.

```
# op is our hamiltonian
# multiplying by parameter theta and finding the exponentiation
evo_time = Parameter('θ')
evolution_op = (evo_time*op).exp_i()
print(evolution_op) # Note, EvolvedOps print as exponentiations
print(repr(evolution_op))
```

```
e^(-i*1.0*θ * (
  -258.5 * IIIIIIIZ
```

```
# approximate e^-iHt using two-qubit gates
trotterized_op = PauliTrotterEvolution(trotter_mode=Suzuki(order=2, reps=1)).convert(evo_and_meas)
# We can also set trotter_mode='suzuki' or leave it empty to default to first order Trotterization.
print(trotterized_op)
```

```
ComposedOp([
  OperatorMeasurement(-258.5 * IIIIIIIZ
  - 388.0 * IIIIIIZI
  - 517.5 * IIIIIZII
  - 647.0 * IIIIZIII
  - 776.5 * IIIZIIII
  - 130.0 * IIZIIIII
  - 260.0 * IZIIIIII
```

Here are the expectation values $\langle\Phi_+|e^{iHt}He^{-iHt}|\Phi_+\rangle$ corresponding to the different values of the parameter

```
h_trotter_expectations.eval()
```

```
[(-0.5+0j),
 (-0.49999999999980105+2.8422e-14j),
 (-0.5000000000000568+5.6843e-14j),
 (-0.49999999999993747+0j),
 (-0.49999999999920425+5.6843e-14j),
 (-0.49999999999997726+2.8422e-14j),
 (-0.5000000000000284+0j),
 (-0.5000000000009948+0j)]
```

```
sampler = CircuitSampler(backend=Aer.get_backend('aer_simulator'))
# sampler.quantum_instance.run_config.shots = 1000
sampled_trotter_exp_op = sampler.convert(h_trotter_expectations)
sampled_trotter_energies = sampled_trotter_exp_op.eval()
print('Sampled Trotterized energies:\n {}'.format(np.real(sampled_trotter_energies)))
```

```
Sampled Trotterized energies:
 [ 27.2265625    31.84765625  31.84765625 -55.953125    64.1953125
 -18.984375   -62.88476563 -30.53710937]
```

Analyzing Results

```python
print("variable order:", [var.name for var in result.variables])
for s in result.samples:
    print(s)
```

```
variable order: ['x_0', 'x_1', 'x_2', 'x_3', 'x_4']
SolutionSample(x=array([1., 1., 0., 1., 0.]), fval=13.0, probability=0.029900000000000003, status=<OptimizationResult
Status.SUCCESS: 0>)
SolutionSample(x=array([0., 0., 1., 0., 1.]), fval=12.0, probability=0.036300000000000001, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([1., 1., 1., 0., 0.]), fval=12.0, probability=0.0338, status=<OptimizationResultStatus.SUCCES
S: 0>)
SolutionSample(x=array([0., 0., 1., 1., 0.]), fval=11.0, probability=0.0417, status=<OptimizationResultStatus.SUCCES
S: 0>)
SolutionSample(x=array([0., 1., 0., 0., 1.]), fval=11.0, probability=0.025500000000000002, status=<OptimizationResult
Status.SUCCESS: 0>)
SolutionSample(x=array([0., 1., 0., 1., 0.]), fval=10.0, probability=0.0324, status=<OptimizationResultStatus.SUCCES
```

```
SolutionSample(x=array([1., 1., 1., 1., 1.]), fval=25.0, probability=0.0221, status=<OptimizationResultStatus.INFEASI
BLE: 2>)
SolutionSample(x=array([0., 1., 1., 1., 1.]), fval=22.0, probability=0.0252, status=<OptimizationResultStatus.INFEASI
BLE: 2>)
SolutionSample(x=array([1., 0., 1., 1., 1.]), fval=21.0, probability=0.039199999999999999, status=<OptimizationResultS
tatus.INFEASIBLE: 2>)
SolutionSample(x=array([1., 1., 0., 1., 1.]), fval=20.0, probability=0.0358, status=<OptimizationResultStatus.INFEASI
BLE: 2>)
SolutionSample(x=array([1., 1., 1., 0., 1.]), fval=19.0, probability=0.028099999999999993, status=<OptimizationResult
Status.INFEASIBLE: 2>)
SolutionSample(x=array([1., 1., 1., 1., 0.]), fval=18.0, probability=0.0331, status=<OptimizationResultStatus.INFEASI
BLE: 2>)
SolutionSample(x=array([0., 0., 1., 1., 1.]), fval=18.0, probability=0.0218, status=<OptimizationResultStatus.INFEASI
```

Analyzing Results

```python
def get_filtered_samples(
    samples: List[SolutionSample],
    threshold: float = 0,
    allowed_status: Tuple[OptimizationResultStatus] = (OptimizationResultStatus.SUCCESS,),
):
    res = []
    for s in samples:
        if s.status in allowed_status and s.probability > threshold:
            res.append(s)

    return res
```

```python
filtered_samples = get_filtered_samples(
    result.samples, threshold=0.005, allowed_status=(OptimizationResultStatus.SUCCESS,)
)
for s in filtered_samples:
    print(s) # val for each configurations; this filters out low prob
```

```
SolutionSample(x=array([1., 1., 0., 1., 0.]), fval=13.0, probability=0.029900000000000003, status=<OptimizationResult
Status.SUCCESS: 0>)
SolutionSample(x=array([0., 0., 1., 0., 1.]), fval=12.0, probability=0.03630000000000001, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([1., 1., 1., 0., 0.]), fval=12.0, probability=0.0338, status=<OptimizationResultStatus.SUCCES
S: 0>)
SolutionSample(x=array([0., 0., 1., 1., 0.]), fval=11.0, probability=0.0417, status=<OptimizationResultStatus.SUCCES
S: 0>)
SolutionSample(x=array([0., 1., 0., 0., 1.]), fval=11.0, probability=0.025500000000000002, status=<OptimizationResult
Status.SUCCESS: 0>)
SolutionSample(x=array([0., 1., 0., 1., 0.]), fval=10.0, probability=0.0324, status=<OptimizationResultStatus.SUCCES
```

Analyzing Results

```
SolutionSample(x=array([1., 1., 0., 1., 0.]), fval=13.0, probability=0.029900000000000003, status=<OptimizationResult
Status.SUCCESS: 0>)
SolutionSample(x=array([0., 0., 1., 0., 1.]), fval=12.0, probability=0.03630000000000001, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([1., 1., 1., 0., 0.]), fval=12.0, probability=0.0338, status=<OptimizationResultStatus.SUCCES
S: 0>)
SolutionSample(x=array([0., 0., 1., 1., 0.]), fval=11.0, probability=0.0417, status=<OptimizationResultStatus.SUCCES
S: 0>)
SolutionSample(x=array([0., 1., 0., 0., 1.]), fval=11.0, probability=0.025500000000000002, status=<OptimizationResult
Status.SUCCESS: 0>)
SolutionSample(x=array([0., 1., 0., 1., 0.]), fval=10.0, probability=0.0324, status=<OptimizationResultStatus.SUCCES
S: 0>)
SolutionSample(x=array([1., 0., 0., 0., 1.]), fval=10.0, probability=0.026300000000000004, status=<OptimizationResult
Status.SUCCESS: 0>)
SolutionSample(x=array([0., 1., 1., 0., 0.]), fval=9.0, probability=0.027200000000000002, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([1., 0., 0., 1., 0.]), fval=9.0, probability=0.0194, status=<OptimizationResultStatus.SUCCESS:
0>)
SolutionSample(x=array([1., 0., 1., 0., 0.]), fval=8.0, probability=0.019399999999999994, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([0., 0., 0., 0., 1.]), fval=7.0, probability=0.026200000000000005, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([1., 1., 0., 0., 0.]), fval=7.0, probability=0.0339, status=<OptimizationResultStatus.SUCCESS:
0>)
SolutionSample(x=array([0., 0., 0., 1., 0.]), fval=6.0, probability=0.024300000000000002, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([0., 0., 1., 0., 0.]), fval=5.0, probability=0.025300000000000003, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([0., 1., 0., 0., 0.]), fval=4.0, probability=0.025699999999999997, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([1., 0., 0., 0., 0.]), fval=3.0, probability=0.0375, status=<OptimizationResultStatus.SUCCESS:
0>)
SolutionSample(x=array([0., 0., 0., 0., 0.]), fval=0.0, probability=0.04229999999999999, status=<OptimizationResultSt
atus.SUCCESS: 0>)
```

References

https://en.wikipedia.org/wiki/Knapsack_problem#:~:text=The%20knapsack%20problem%20is%20a,is%20as%20large%20as%20possible.

https://en.wikipedia.org/wiki/NP-completeness

https://qiskit.org/documentation/stable/0.24/stubs/qiskit.optimization.applications.ising.knapsack.html

https://qiskit.org/documentation/optimization/tutorials/09_application_classes.html#Knapsack-problem

https://www.frontiersin.org/articles/10.3389/fphy.2014.00005/full

https://qiskit.org/documentation/tutorials/operators/01_operator_flow.html

https://arxiv.org/pdf/1908.02210.pdf

https://arxiv.org/pdf/1411.4028.pdf