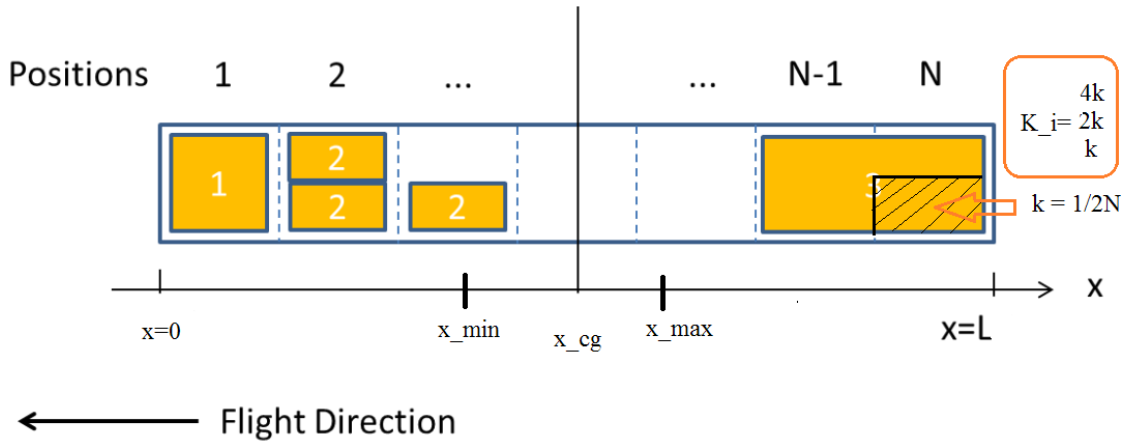
Airbus QC Challenge: Aircraft Loading Optimization (problem no5)

Author: Sorin Moldoveanu
Affiliation: home

Introduction

The aircraft loading problem is described as a simpliffied version of this kind of problems. Cargo bay is a box with N cargo bay spaces. The cargo boxes (containers) are of 3 types. The optimization target (set of containers which mass has to be maximized within a maximum payload limit) has constraints which are implemented in 3 steps:



Step 1: keep the loaded container mass within the limit and take into accont that it must be physically possible to place the containers on the aircraft
Step 2: place the containers in order to respect the centre of gravity limits
Step 3: the selected set has to respect shear limits of the fuselage

Problem description and formalization; Notations and variables

With respect of container storage, given the N locations, we will consider a new value Ns = 2N (given). We will consider a unit space k as 1/Ns. As a consequence, the space occupied by a given container can be:

$K_i \in \{k, 2k, 4k\}$ where k = 1/N

In this way, the total space occupied by the loaded containers = 1

We will define the following variables:
m_max - maximum payload; scalar; noted Wp into given statement
n - number of mass elements to be loaded - boxes/containers
Ns - number of spaces available for boxes; an unit space is = 1/2 of original_space
M - mass vector n x 1
K - space took by each mass boxes; vector n x 1
x - decision variable vector n x 1; values are 0 or 1
Z - Pauli Z operator
p - penalty

APPROACH #01

Step 1

Load the boxes into cargo bay such as loaded mass to be maximized but less that maximum payload AND it must be physically possible to place the boxes in cargo bay. This means

$max \sum_{i=1}^{n} m_i$

In [1]:
```
#IMPORT
from qiskit import BasicAer
from qiskit_aqua import QuantumInstance
from qiskit_aqua import Operator, run_algorithm
from qiskit_aqua.input import EnergyInput
from qiskit_aqua.translators.ising import aircargo
from qiskit_aqua.algorithms import VQE, QAOA, ExactEigensolver
from qiskit_aqua.components.optimizers import COBYLA
from qiskit_aqua.components.variational_forms import RY
import numpy as np
```

Setup token to run the experiment on a real device (IBMQ) - if case

In [2]:
```
#set for real device if case
#device = 'ibmq_16_melbourne'
from qiskit import IBMQ
IBMQ.load_accounts()
#backend = IBMQ.get_backend(device)
```

Define the problem (instance). Operator instance is created for Issing Hamiltonian as it is translated from the problem. The translation module is "aircargo.py". We will test with a set of 5 boxes and 4 places in cargo bay.

In [8]:
```
# define the problem
# variables
# m_max - maxumum payload; scalar
# n - number of mass elements to be loaded - boxes
# N - number of spaces available for boxes; an unit space is = 1/2 of origin
al_space
# M - mass vector n x 1
# K - space took by each mass boxes; vector n x 1
# p - penalty
# e - identity vector
# E - identity matrix

m_max=6
n=5
N=4  # as defined into the original problem there are 2 storage locations; h
ere we use a location = 1/2 of original
p = 1/5.0
M=np.array([2.1, 3.2, 0.9, 2.8, 1.2])
#M=np.array([2.1/6, 3.2/6, 0.9/6, 0.8/6, 1.2/6]) # in this case is reported
to m_max; m_max becomes = 1
K=np.array([0.5,0.5,0.25,0.25,0.25])
print('M',M)
print('K',K)
e=np.ones(n)
E = np.matmul(np.asmatrix(e).T, np.asmatrix(e))

qubitOp, offset = aircargo.get_aircargo_qubitops_01(M, K, m_max, p)
#qubitOp, offset = aircargo.get_aircargo_qubitops_fiacco(M, K, m_max, p)
algo_input = EnergyInput(qubitOp)
print('offset=',offset,'qubitOp=', qubitOp)
```

```
M [2.1 3.2 0.9 2.8 1.2]
K [0.5  0.5  0.25 0.25 0.25]
offset= 14.575500000000003 qubitOp= Representation: paulis, qubits: 5, size:
15
```

In order to format the results, here are some functions.

In [4]:
```
#prepare some printing format
def index_to_selection(i, n):
    s = "{0:b}".format(i).rjust(n)
    x = np.array([1 if s[i]=='1' else 0 for i in reversed(range(n))])
    return x

def print_result(result):
    selection = aircargo.sample_most_likely(result['eigvecs'][0])
    value = aircargo.aircargo_value_01(selection, M, K, m_max, p)
    print('Optimal: selection {}, value {:.4f}'.format(selection, value))

    probabilities = np.abs(result['eigvecs'][0])**2
    i_sorted = reversed(np.argsort(probabilities))
    print('\n----------------- Full result ---------------------')
    print('selection\tvalue\t\tprobability')
    print('---------------------------------------------------')
    for i in i_sorted:
        x = index_to_selection(i, n)
        value = aircargo.aircargo_value_01(x, M, K, m_max, p)
        probability = probabilities[i]
        if value>=0: print('%10s\t%.4f\t\t%.4f' %(x, value, probability))
```

Exact eigen values (can be used as classical reference)

In [9]:
```
#exact eigensolver - as reference
exact_eigensolver = ExactEigensolver(qubitOp, k=1)
result = exact_eigensolver.run()
print('\n','Exact_Eigensolver')
print_result(result)
```

```
 exact_eigensolver
Optimal: selection [0 1 0 1 0], value 5.9875

----------------- Full result ---------------------
selection       value           probability
---------------------------------------------------
[0 1 0 1 0]     5.9875          1.0000
[1 1 1 1 1]     6.5595          0.0000
[0 1 1 1 1]     7.2055          0.0000
[0 1 0 0 0]     1.5820          0.0000
[1 1 0 0 0]     5.2020          0.0000
[1 0 1 0 0]     1.1875          0.0000
[0 1 1 0 0]     3.3655          0.0000
[1 1 1 0 0]     6.1795          0.0000
[0 0 0 1 0]     0.6395          0.0000
[1 0 0 1 0]     4.6455          0.0000
[1 1 0 1 0]     7.2055          0.0000
[0 0 1 1 0]     2.5920          0.0000
[1 0 1 1 0]     5.7920          0.0000
[0 1 1 1 0]     6.7380          0.0000
[1 1 1 1 0]     7.1500          0.0000
[1 0 0 0 1]     1.8295          0.0000
[0 1 0 0 1]     3.8755          0.0000
[1 1 0 0 1]     6.4375          0.0000
[1 0 1 0 1]     3.5520          0.0000
[0 1 1 0 1]     5.2020          0.0000
[1 1 1 0 1]     6.9580          0.0000
[0 0 0 1 1]     3.1500          0.0000
[1 0 0 1 1]     6.0980          0.0000
[0 1 0 1 1]     6.9120          0.0000
[1 1 0 1 1]     7.0720          0.0000
[0 0 1 1 1]     4.6455          0.0000
[1 0 1 1 1]     6.7875          0.0000
```

The **VQE** algorithm. Here we specify the classical optimizer (COBYLA) and the variational form (RY) to be used.

In [10]:
```
#VQE
backend = BasicAer.get_backend('statevector_simulator')
seed = 50

cobyla = COBYLA()
cobyla.set_options(maxiter=250)
ry = RY(qubitOp.num_qubits, depth=3, entanglement='full')
#num_of_parameters = d x q x (q+1)/2 +q where d = circuit depth and q = numb
of qubits
#depth example: d=1 means one step quantum algo - can be expressed as a unit
ary operator
vqe = VQE(qubitOp, ry, cobyla, 'matrix')
vqe.random_seed = seed

quantum_instance = QuantumInstance(backend=backend, seed=seed, seed_mapper=s
eed)

result = vqe.run(quantum_instance)
print('\n','VQE')
print_result(result)

#in order to know circut width and depth: how many qubits are required (circ
uit width)
# or how large the maximum number of gates applied to a single qubit (circui
t depth) is

#print("circuit_width", result['circuit_info']['width'])
#print("circuit_depth", result['circuit_info']['depth'])
```

```
 VQE
Optimal: selection [1 1 0 0 0], value 5.2020

----------------- Full result ---------------------
selection       value           probability
---------------------------------------------------
[1 1 0 0 0]     5.2020          0.9987
[1 0 1 0 0]     1.1875          0.0004
[1 0 0 0 1]     1.8295          0.0002
[1 0 1 0 1]     3.5520          0.0002
[1 1 0 1 0]     7.2055          0.0002
[1 1 1 0 0]     6.1795          0.0001
[1 1 1 0 1]     6.9580          0.0001
[0 1 1 0 1]     5.2020          0.0000
[1 0 0 1 0]     4.6455          0.0000
[1 1 0 0 1]     6.4375          0.0000
[0 1 1 0 0]     3.3655          0.0000
[0 1 0 0 0]     1.5820          0.0000
[0 1 0 0 1]     3.8755          0.0000
[1 1 1 1 0]     7.1500          0.0000
[1 1 0 1 1]     7.0720          0.0000
[1 0 1 1 1]     6.7875          0.0000
[1 0 0 1 1]     6.0980          0.0000
[1 0 1 1 0]     5.7920          0.0000
[1 1 1 1 1]     6.5595          0.0000
[0 0 1 1 1]     4.6455          0.0000
[0 0 0 1 1]     3.1500          0.0000
[0 1 1 1 1]     7.2055          0.0000
[0 0 1 1 0]     2.5920          0.0000
[0 1 1 1 0]     6.7380          0.0000
[0 1 0 1 1]     6.9120          0.0000
[0 1 0 1 0]     5.9875          0.0000
[0 0 0 1 0]     0.6395          0.0000
```

The **QAOA** algorithm

In [11]:
```
#QAOA
backend = BasicAer.get_backend('statevector_simulator')
seed = 50

cobyla = COBYLA()
cobyla.set_options(maxiter=250)
qaoa = QAOA(qubitOp, cobyla, 3, 'matrix')
qaoa.random_seed = seed

quantum_instance = QuantumInstance(backend=backend, seed=seed, seed_mapper=s
eed)

result = qaoa.run(quantum_instance)
print('\n','QAOA')
print_result(result)
```

```
 QAOA
Optimal: selection [0 1 0 1 0], value 5.9875

----------------- Full result ---------------------
selection       value           probability
---------------------------------------------------
[0 1 0 1 0]     5.9875          0.1073
[1 1 0 0 0]     5.2020          0.0946
[1 0 0 1 0]     4.6455          0.0860
[0 1 0 0 1]     3.8755          0.0798
[0 1 1 0 0]     3.3655          0.0763
[0 0 0 1 1]     3.1500          0.0728
[0 0 1 1 0]     2.5920          0.0697
[1 0 0 0 1]     1.8295          0.0649
[1 0 1 0 0]     1.1875          0.0622
[1 1 0 1 0]     7.2055          0.0406
[0 1 0 1 1]     6.9120          0.0296
[0 1 1 1 0]     6.7380          0.0267
[1 1 0 0 1]     6.4375          0.0242
[1 1 1 0 0]     6.1795          0.0218
[1 0 0 1 1]     6.0980          0.0211
[1 0 1 1 0]     5.7920          0.0189
[0 1 1 0 1]     5.2020          0.0153
[0 0 1 1 1]     4.6455          0.0131
[1 0 1 0 1]     3.5520          0.0103
[0 1 0 0 0]     1.5820          0.0018
[0 0 0 1 0]     0.6395          0.0012
[1 1 1 1 1]     6.5595          0.0005
[1 1 0 1 1]     7.0720          0.0003
[1 1 1 1 0]     7.1500          0.0002
[0 1 1 1 1]     7.2055          0.0001
[1 1 1 0 1]     6.9580          0.0001
[1 0 1 1 1]     6.7875          0.0001
```

Step 2

Add constraints against the mass center positioning (gravity center limits). Let's define an elemetary length - l:
$l = L/N$
Having given the limits of gravity center as x_max and x*min, the constraints for the limits are:*

$ /\sum{i=1}^{N}iXm*i - x*{min} \sum_{i=1}^{N}Xm*i \geqslant 0*$< br >< br > and < br >< br >/\sum*{i=1}^{N}iXm*i + x*{max} \sum_{i=1}^{N}Xm_i \geqslant 0$< br >< br > where :< br >x \in \left { 0,1 \right }^n$

Step 3

Now we have to add the constraints with regards on share limits. Having defined the max for share as S_{max} the constraint is:

$lx^T M \leqslant S_{max}$

NOTE

The constraints at steps 2 & 3 are to be implemented in the same manner as for the step 1

APPROACH #02

We will take another approach for the objective function, vs: we will consider the cost function related to x_{cg} - the taget centre fo gravity. In this way the function is:

min $\left(l\sum_{i=1}^{N} im_i - m_{max}x_{cg} \right)$

Subject to

$\sum_{i=1}^{n} m_i \leqslant m_{max}$

$\sum_{i=1}^{n} k_i \leqslant 1$

$lx^T M \leqslant S_{max}$

$l\sum_{i=1}^{N} im_i - x_{min}\sum_{i=1}^{N} m_i \geqslant 0$

$-l\sum_{i=1}^{N} im_i + x_{max}\sum_{i=1}^{N} m_i \geqslant 0$

We will use the same decision variable $x \in \{0,1\}^n$. The constraints are mapped to penalty terms and scaled by a parameter p (as we did for approach#01)

References

[1] Nielsen M. and Chuang I. "Quantum Computation and Quantum Information", Cambridge University Press (2010)
[2] Qiskit open-source framework (qiskit, qiskit-aqua, qiskit-aer, qiskit-terra) at https://qiskit.org/ (https://qiskit.org/) and https://github.com/Qiskit (https://github.com/Qiskit)
[3] IBMQ at https://quantumexperience.ng.bluemix.net (https://quantumexperience.ng.bluemix.net)
[4] Patrik J. Coles et al, "Quantum Algorithm Implementation for Beginners", arXiv:1804.03719 (Apr 2018)
[5] Lucas Andrew, "Ising formulations of many NP problems", arXiv"1302.5843 (Jan 2014)
[6] Egger J. Daniel et al, "Qunatum optimization using variational algorithms on near-term quantum devices", arXiv:1710.01022 (Oct 2017)
[7] Edward Fahri et al, "A quantum approximate optimization algorithm", arXiv:1411.4028 (Nov 2014)
[8] Parwadi Moengin, "Polynomial penalty method for solving linear programming problems", International Journal of Applied Mathematics, 40:3, IJAM_40_3_06 (Aug 2010)
[9] Nannicini G, "Performance of hybrid quantum/classical variational heuristics for combinatorial optimization", arXiv:1805.12037 (Dec 2018)
[10] Harrow W Aram et al, "Quantum algorithm for linear systems of equations", arXiv:0811.3171 (Sept 2009)
[11] Lucas Andrew, "Hard combinatorial problems and minor embeddings on lattice graphs", arXiv:1812.01789 (Dec 2018)
[12] Kevin C Young, "Adiabatic quantum optimization with the wrong Hamiltonian", arXiv:1310.0529 (Oct 2013)
[13] Preskill John, "Quantum Computing in the NISQ era and beyond", arXiv:1801.00862 (Jul 2018)
[13] Airbus, "Aircraft Loading Optimization", Airbus Quantum Computing Challenge

In [ ]: