“Comparing brute force to quantum annealing for solving combinatorial optimization problems”

By

Suleman Hersi
s188218

Introduction

Since the dawn of man, traveling has been a part of daily life. Either traveling for food, work, or leisure. Travel is a big part of our life. Traveling is also a stressful and logistical nightmare. All the different things that need to be scheduled, remembered, and packed. Packing is the most stressful part of the experience. What do you bring? You have a limited capacity and usually own more things than you can bring. This is a conundrum that's plagued mankind for a long time. More so in a later year when airlines take a hefty penalty fee for being a kg or two over the weight limit. How do you solve this issue? Simply put, the problem is bringing the most value with you without exceeding the weight limit. This sounds like an optimization problem.

In this report, we will take a look at the famous knapsack problem. This is a classical optimization problem that revolves around optimization given boundary conditions.

The knapsack problem has been around for a long time. The first iteration of this can be traced back to the 19th century. The problem is stated as such:

Given a knapsack with a limited carrying capacity and items of varying weight and value. How do you maximize the value of the items chosen without exceeding the weight limit on the knapsack?

This problem has been solved many times in a lot of different ways. For example, it can be done by hand or using computers. In this report, we will use a quantum computer and compare it to regular brute force to see if we can gain insights into how quantum computing optimization works.

As the problem states, we are given a set of items with a given weight and limit. For this project, we have three different scenarios: seven items, 21 items, and 50 items. Each of these scenarios has different max weight limits. To see the weight and value distribution. See Appendix 2.

In this report, we will discuss how the different methods work, emphasizing how the quantum computer works. We will then look at some interesting results that compare the two methods and see how the different scenarios play out with the quantum computer.

Solving the knapsack problem

As mentioned in the introduction, we will solve the knapsack problem in two ways. First, we will implement a brute force method and then, secondly, use a quantum computer to solve the problem.

Brute Force

There is several ways to solve optimization problems, but the brute force method is the simplest. This simply means trying all the possible combinations for the problem and then evaluating them. This doesn't involve math, science, or anything other than time. And in that lies its weakness. As might be apparent, any problem can be solved using this method given enough time; however, as soon as the problem reaches a certain complexity, the time needed will exponentially increase. At a certain point, the time will be astronomical and beyond our lifetime.

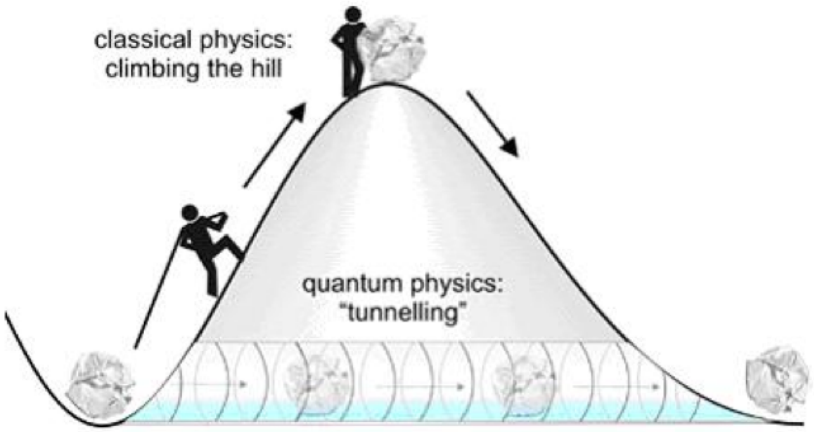
Quantum Computer

Presume brute force is the simplest way of solving an optimization problem. Using a quantum computer has to be the most complex way. This might be up for discussion, but it is on a different level.

The quantum computer used for this solution was created by D-wave. D-Wave is a Canadian company that creates a type of quantum computer that uses quantum annealing. So to understand how we solve the knapsack problem with a quantum computer, we first need to understand what quantum annealing is.

Quantum annealing

Quantum annealing derives its name from something called simulated annealing, which in turn derives its name from the ancient art of annealing, still practiced to this day. Annealing is the process of increasing the temperature of a metal so much that it changes its properties and becomes softer to work with. The metal is then cooled down and now keeps its new shape. The analogy that both quantum annealing and simulated annealing take from physical annealing is in regards to the fact that temperature over time will go down. This is an observable fact about most systems. They seek a lower state. Objects roll down hills. Hot things cool down. This is also true for energy and is why we can use the properties of quantum mechanics to solve optimization problems by framing the problem as an energy minimization problem.



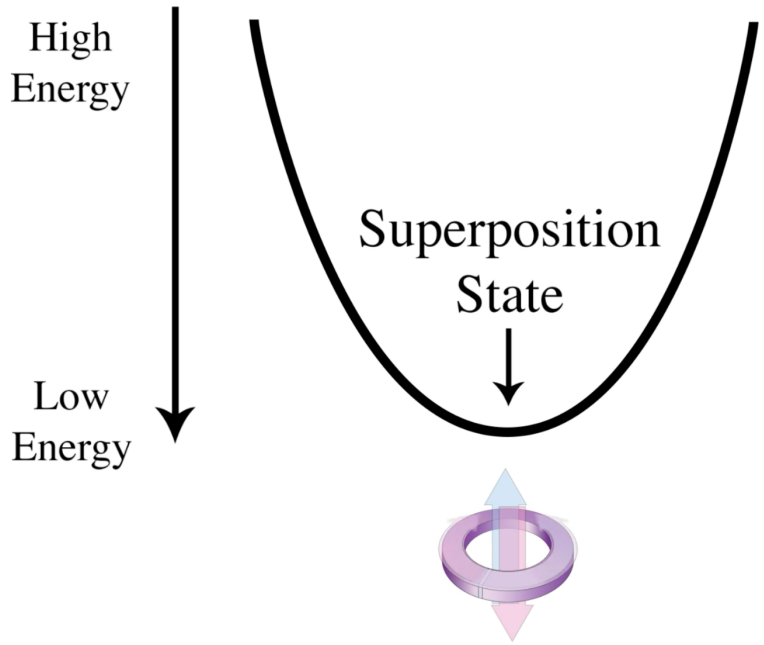
So quantum annealing is used to solve optimization problems where the goal is to reach a global minimum. We have today many techniques for solving such problems, but one of the interesting concepts that quantum physics allows is quantum tunneling. This is a property we get from working in the quantum world. It allows us to escape a local minimum much more efficiently than other methods.

To get a better understanding of how quantum annealing works, let's take a look at the process of quantum annealing.

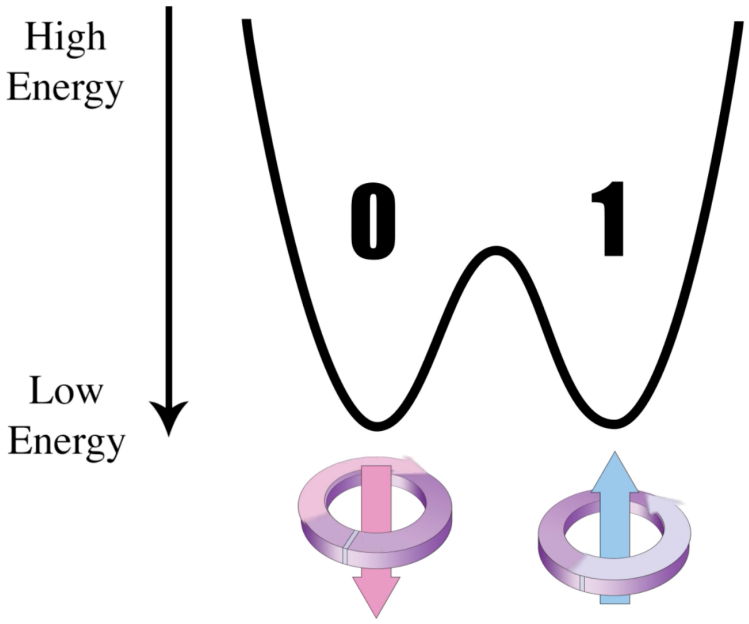
Qubit

Let us start by just looking at what we call a qubit. A qubit can be in the state of 1 or 0. Like a classical computer bit, but given that it's a quantum bit, it can also be in both states simultaneously. This is what's called a superposition. After going through the quantum annealing, we get the qubit from the superposition to either 1 or 0. Then, giving us a result we can read out.

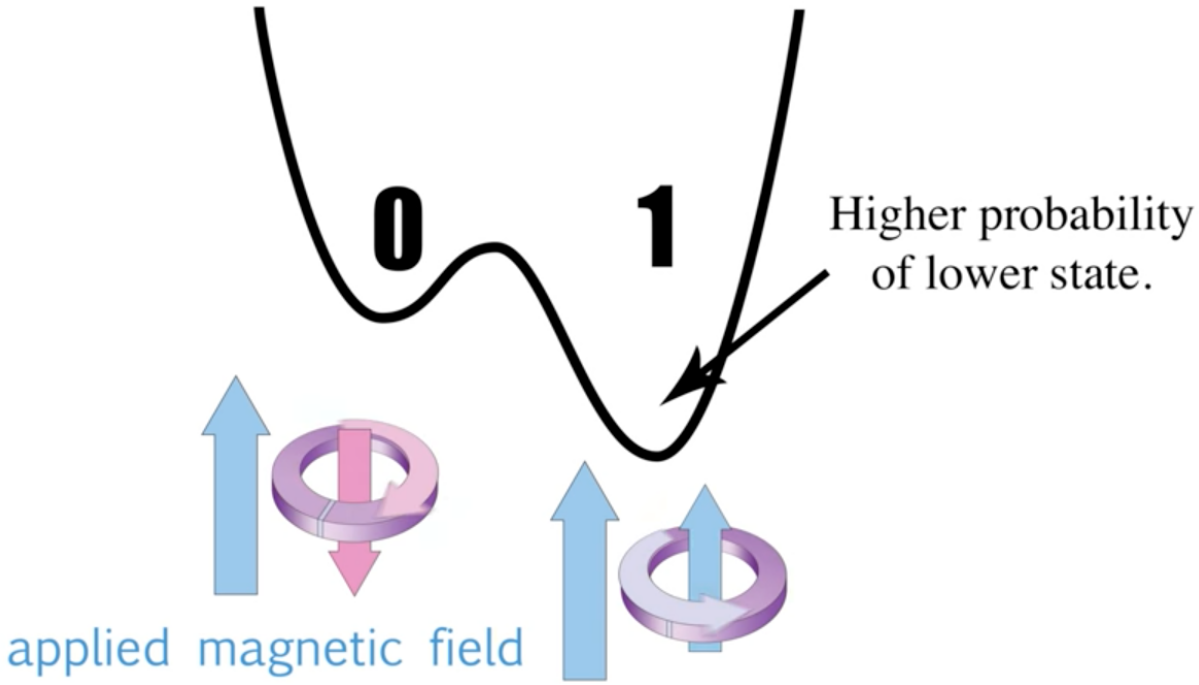
This can be seen demonstrated on what's called an energy diagram.



In this diagram, we can see that the lowest energy state is at the superposition state. As mentioned above, after the quantum annealing, we get the qubit in either state 1 or 0.

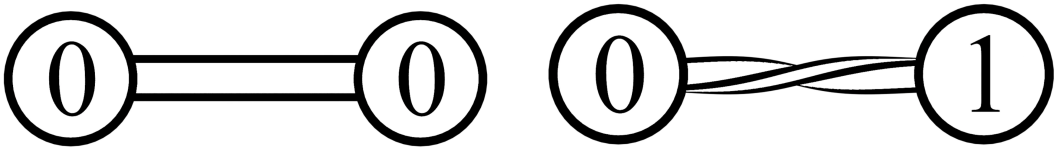


The energy diagram shows us that after the anneal, we end up with something called the double-well potential. That means given all else being equal, the chance of our qubit ending in a state of 1 or 0 is 50/50. An interesting phenomenon is that we can influence which state it ends up in. By applying external magnetic fields, we can increase the probability that the qubit will end up in the lower state. This is called a bias.

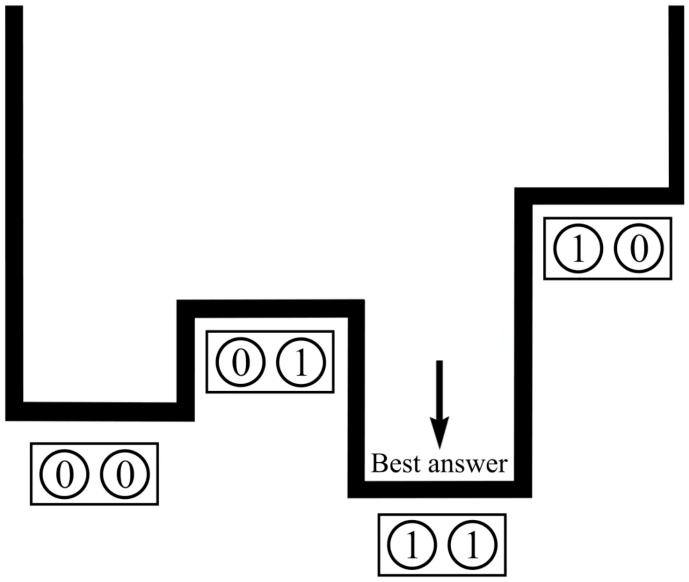


Let's now look at something called a coupler. A coupler is a device that introduces something called entanglement into our system. We have discussed how our qubits can be in a state of 1 or 0, but when combined with a coupler, we can now create a new qubit that can be in four states. As the name suggests, we use this coupler to couple two qubits together.

As mentioned, the couplers introduce something called entanglement. Quantum entanglement is a phenomenon where two or more qubits are linked together to either always be in the same state or be in opposite states.



Now, how do these concepts help solve our optimization problem? Well, let's apply it to our knapsack problem. We were given seven items. And we can either pack an item or not. Let's say that if an item is 1, it's packed, and 0 if it's not. Now we can represent our solution as 1 or 0.



Given that we have seven items let's represent each item as a single qubit. This gives us seven qubits. If we now entangle them using our coupler, we have seven entangled qubits. And are now defined as a single qubit. Given that before we go through our anneal, the qubits are in a superposition, we have seven entangled qubits in superpositions. Now our entangled qubit system, if you may, represents all the different solutions that can be. Each solution is represented by a state of 1's and 0's. And each of these states has an energy state connected to it. As we do in our energy diagram, we can introduce our bias so that the qubit system with the lowest overall energy state is the one that reflects the best solution.

As stated before, in order to use quantum annealing, we need to state our problem as an energy minimization problem. In the previous section, we explained that quantum anneal will give us the state of the qubits with the lowest energy state. How it gives us this is by using a Hamiltonian. Without going into too much detail, a hamiltonian is a mathematical description of a physical system expressed in terms of energy. It allows us to plot the state of a system and read out the energy state of said system. So for our use, we enter the state of all our qubits system, that is, our seven qubits that are entangled, and read out their energy state. The energy readout will then be sorted from lowest to highest. And we have our solution.

The Hamiltonian is built up in two parts consisting of two hamiltonians. We have our initial Hamiltonian, which has its lowest energy state when all the qubits are in a superposition, and we have our problem hamiltonian, which has our solution as its lowest energy state. The quantum anneal will start by first only containing the initial Hamiltonian and then slowly and controlled introduce our problem hamiltonian. This allows for the energy read out to only contain our problem hamiltonian as its input.

$$H = Hi + Hp$$

So our knapsack problem is being defined in the problem Hamiltonian. At the same time, the D-Wave system takes care of the initial Hamiltonian. Now how do we define our problem, Hamiltonian? We will have to get familiar with something called QUBO.

QUBO stands for "quadratic unconstrained binary optimization," and it's a way to describe optimization problems mathematically. One of the things that make QUBOs worthwhile in quantum computing is that they look almost identical to hamiltonians when the Hamiltonian is expressed as an Ising Hamiltonian.

$$H = -\sum_{i,j} J_{ij}\sigma_i\sigma_j + h\sum_i \sigma_i$$

$$f(x) = \sum_{i<j}^{N} Q_{i,j} x_i x_j + \sum_{i}^{N} Q_{i,i} x_i$$

This allows us to formulate our problem as a QUBO and then insert it into our problem hamiltonian. For this problem, our QUBO had two components. The first was the maximization of the value we chose to bring in our knapsack. That can be expressed like this:

$$-\sum_{i}^{N} v_i x_i$$

Where $i = number\ of\ items$, $v = value\ of\ item$ and $x = state$ ($1\ or\ 0$). In other words, this sums up the value of all the chosen items from 1 to N, which in this case is 7. The negative operator put in front is there since in we are formulating our maximization problem as a minimization problem. So this allows the smallest value to be the biggest value.

Now for this image to be complete, we need to add our constraint. As previously stated, the knapsack can only carry a certain number of weight units. This we also define as a QUBO:

$$\sum_{i}^{N} w_i x_i - W$$

Here $w = weight\ of\ item$ and $W = maximum\ weight$. So we add up all the weights chosen, and we subtract the max weight. This will ensure that our definition of the best solution is the one that has the lowest energy state.

Adding these two together will give us our problem hamiltonian:

$$-\left(\sum_{i}^{N} v_i x_i\right) + \left(\sum_{i}^{N} w_i x_i - W\right)$$

This expression now allows us to find the highest value we can carry that also is equal to the weight capacity. This means that this expression sees the optimal solution as the solution that uses all of the allowed weight. There is an inherent flaw here, given that using less weight and gaining more value is a better solution. However, for this paper, this expression, as stated, will be used.

Now that we have our QUBO, we can implement this on D-waves quantum computers. The way that this takes place is by using their cloud solution and programming our problem in python using their libraries. The code used in this rapport is added as appendix 2.

Results

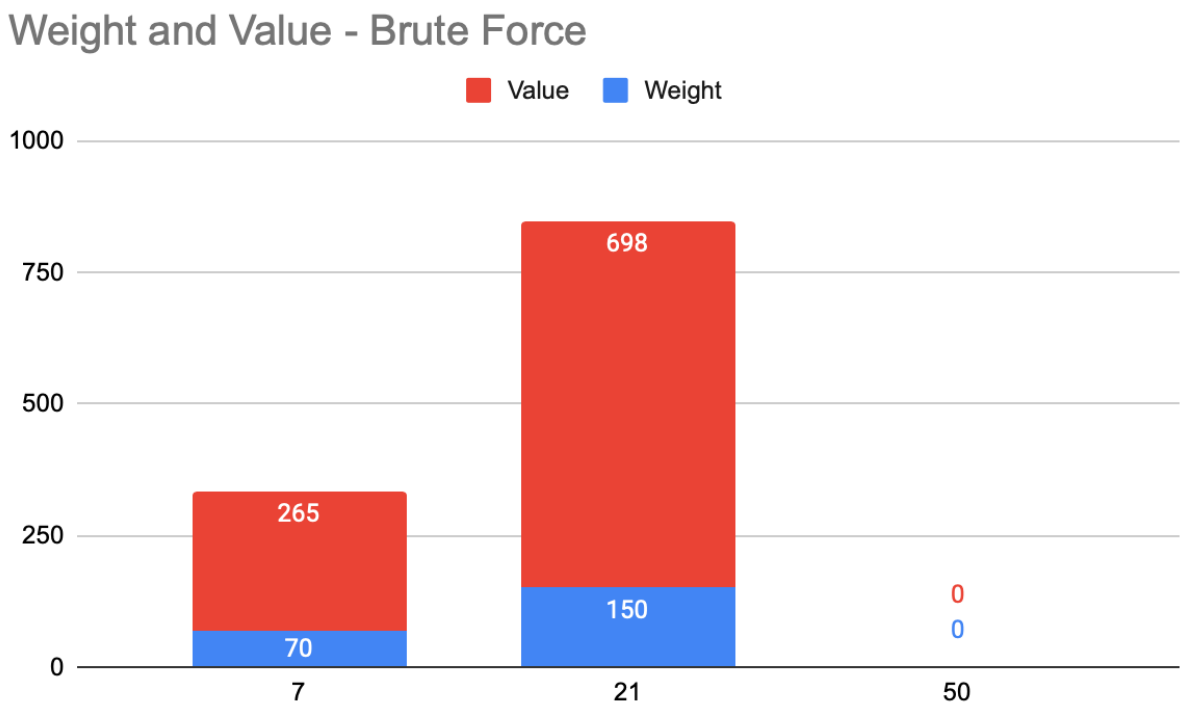
For this project, the parameters were as follows. Find the maximum value that can be carried while also maximizing the weight carried without crossing the weight limit. This was done to reduce complexity. As mentioned before, we solved this problem in two ways. In this section, let's take a look at some of the results.

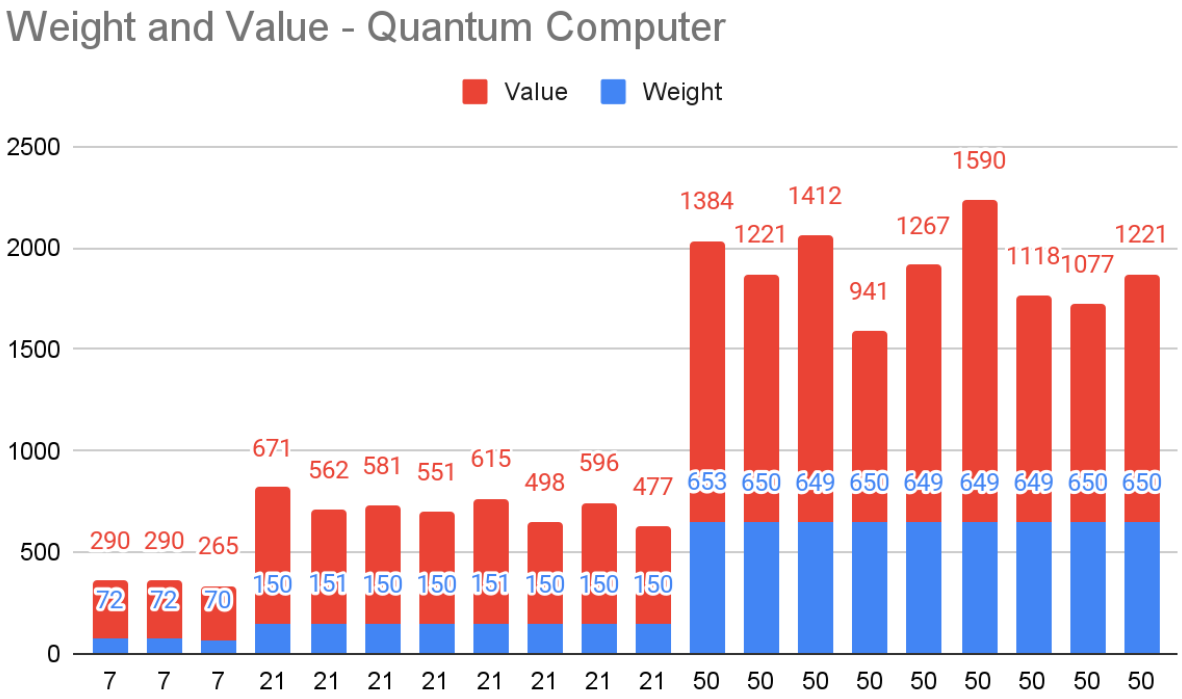
In order to gauge the efficiency of the different methods, the initial problem was extended to include scenarios with more items. The value and weight of each item were randomly assorted, and a random max capacity was set. These values were the same for both methods. This left us with three scenarios:

1. Seven items, with a max weight of 70
2. 21 items, with a max weight of 150
3. 50 items, with a max weight of 650

There was also another parameter that varied during this project. The Lagrange multiplier used was varied to see if this impacted the results from the quantum computer. The Lagrange multiplier will work as a penalty for exceeding the constraint. The theory is that the higher the penalty, the closer the lowest energy state will be to the optimal solution to our problem.

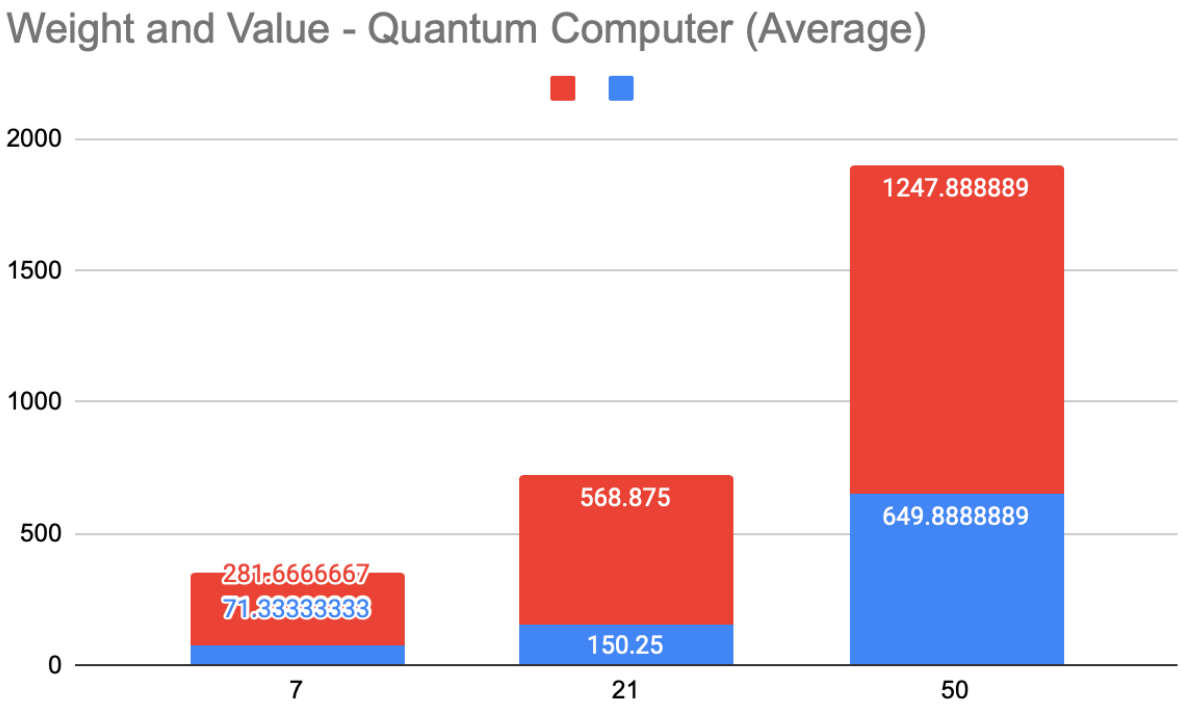
Below are graphs that depict the varying result from the two methods for the different scenarios:

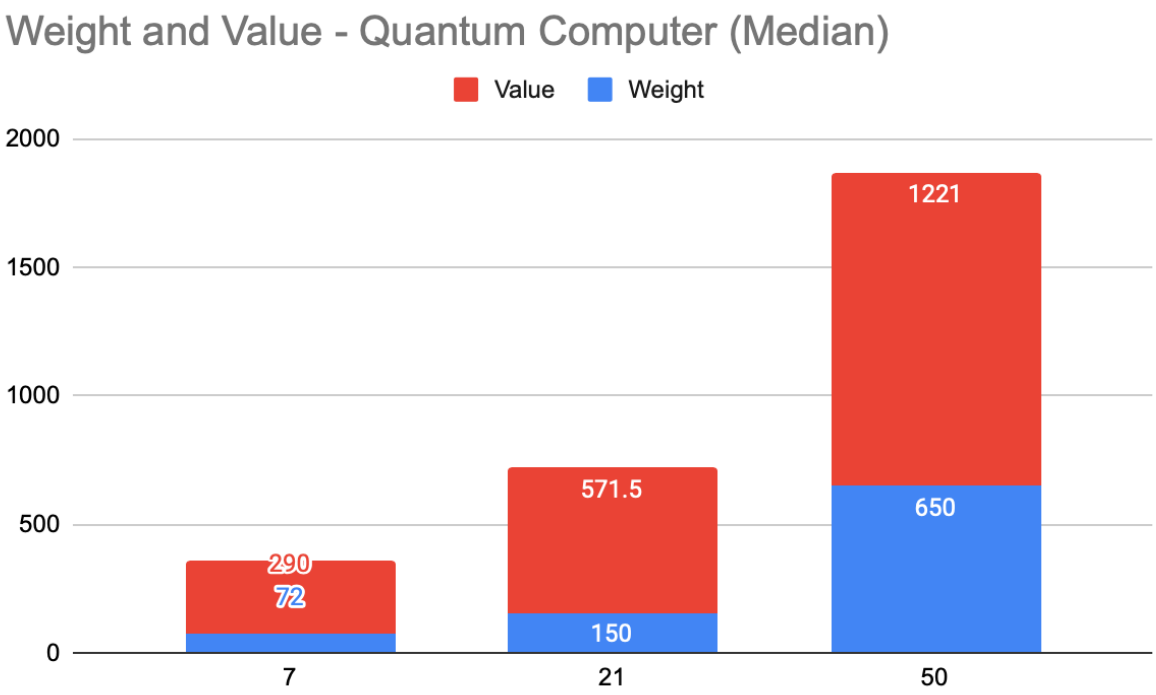




As can be seen from the graphs above, both methods gave us the satisfactory result for our problem when they were able to complete the calculation. The brute force method had issues completing the 50-item scenario. It timed out. Therefore we have no result from those calculations.

From the quantum computer, we can see that, on average, you will get a close enough solution but not necessarily the exact solution on every iteration. The median, however, seems to average out to respect the weight limit.

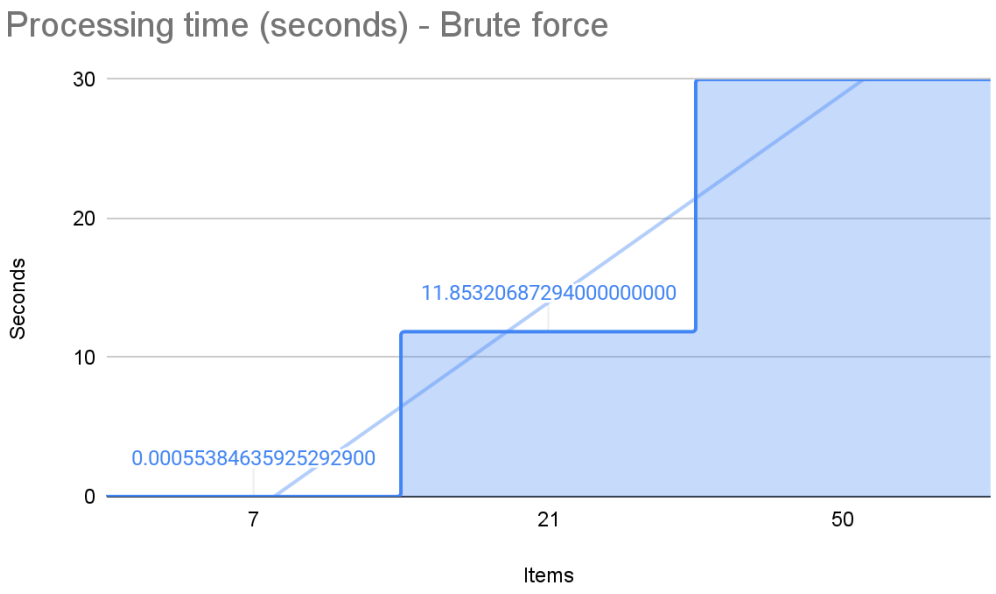

Weight and Value - Quantum Computer (Average)
2000
1500
1000
500
0
281.6666667
71.3333333
568.875
150.25
1247.888889
649.8888889
7
21
50

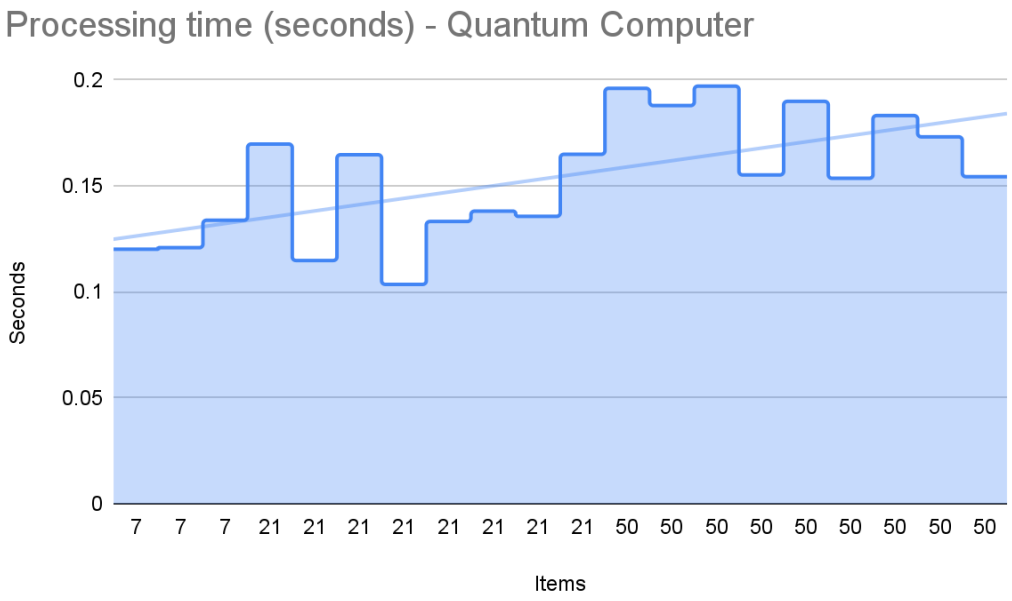

Weight and Value - Quantum Computer (Median)
Value
Weight
2000
1500
1000
500
0
290
72
571.5
150
1221
650
7
21
50

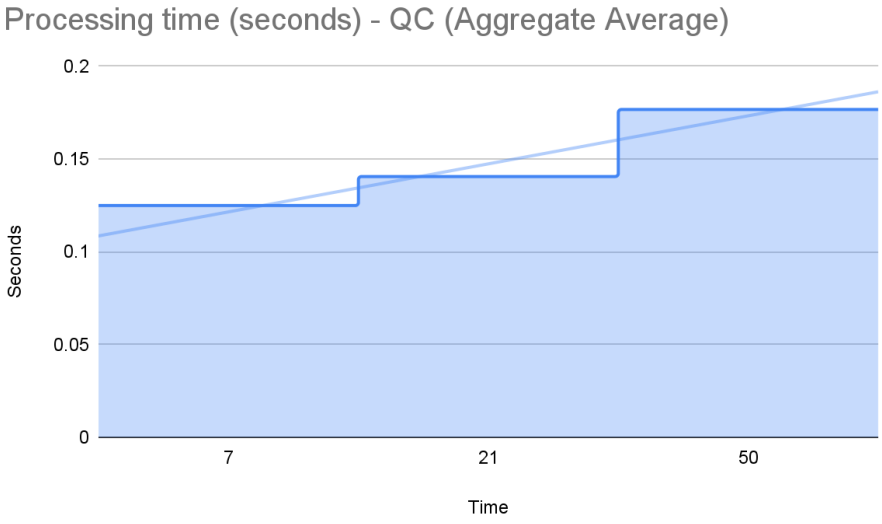
Processing Time

Regarding which method was the quickest, the data shows that at a small enough problem, the difference between these two is minimal. But as can be seen, the larger the problem, the more significant this difference gets. Do notice, however, that at a small enough problem percentage-wise, the brute force method was significantly faster, but for this scenario, it's practically the same.

Below we can see how the different methods and processing times developed throughout the different scenarios. Note here that the brute force method timed out on the 3. scenario.

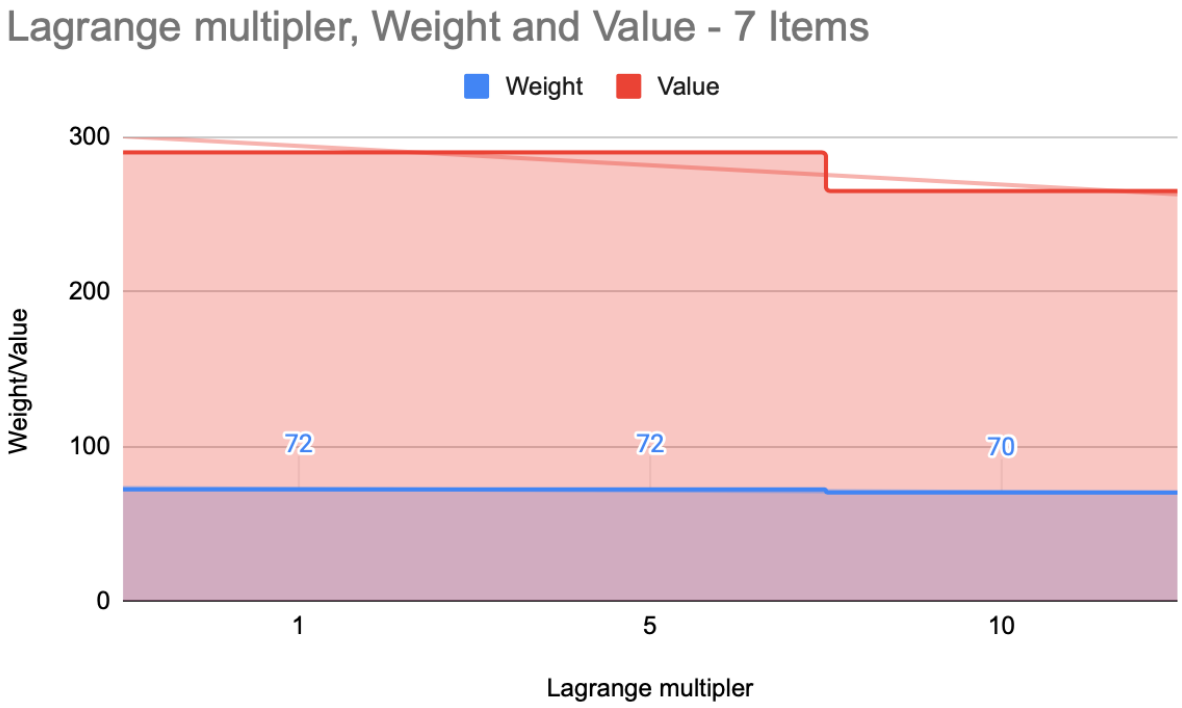


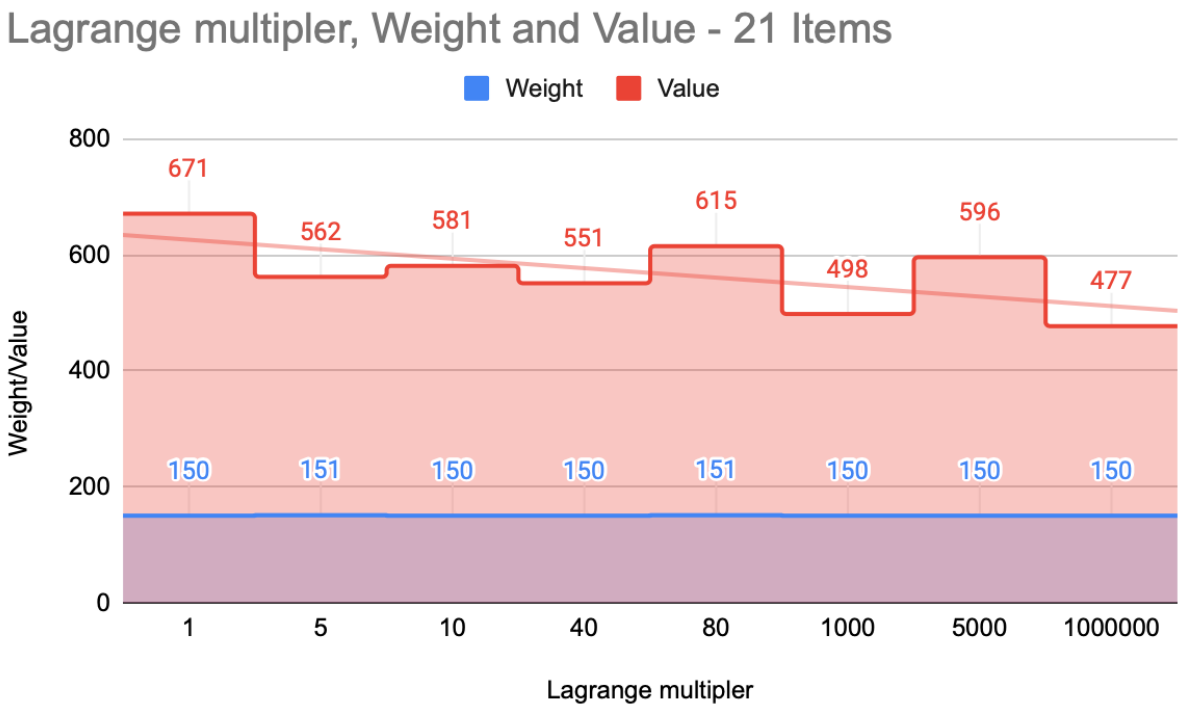

Processing time (seconds) - Quantum Computer
0.2
0.15
0.1
0.05
0
Seconds
7 7 7 21 21 21 21 21 21 21 21 50 50 50 50 50 50 50 50 50
Items

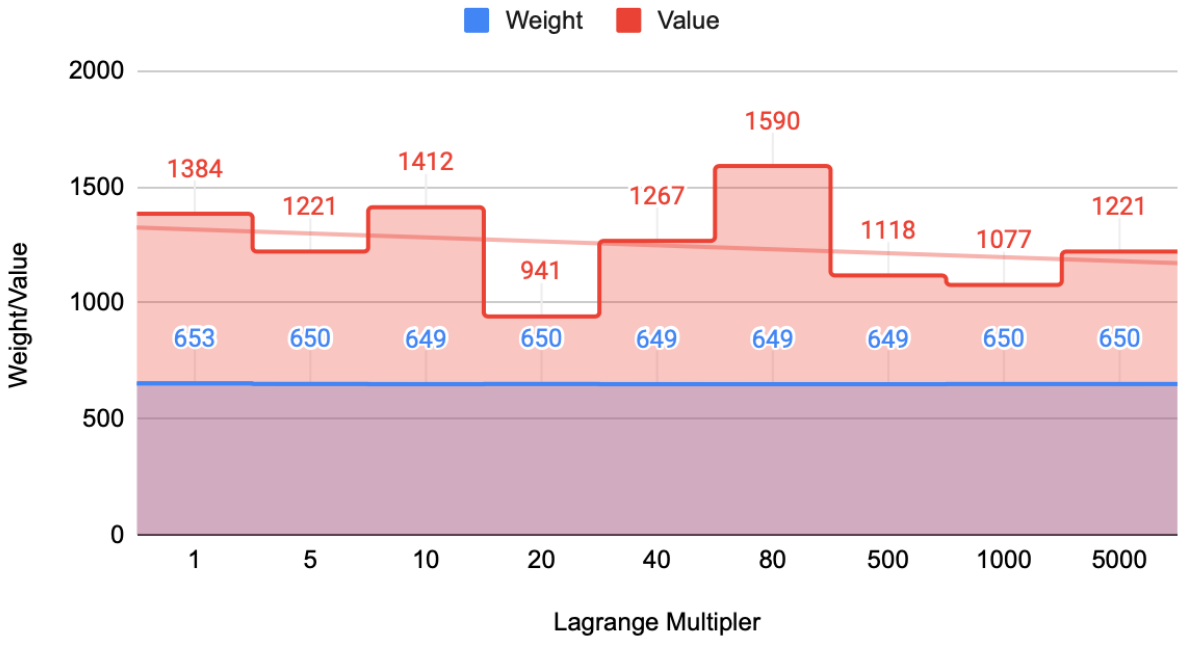

Processing time (seconds) - QC (Aggregate Average)
0.2
0.15
0.1
0.05
0
Seconds
7
21
50
Time

Lagrange Multiplier for Quantum Annealing

In these graphs, we can see that the higher Lagrange multiplier has the desired effect of keeping the weight restraint. However, there does not seem to be a correlation between the Lagrange multiplier and the value variable. There were cases where a lower Lagrange multiplier gave a better solution.




Lagrange multipler, Weight and Value - 21 Items
Weight
Value
800
600
400
200
0
Weight/Value
671
562
581
551
615
498
596
477
150
151
150
150
151
150
150
150
1
5
10
40
80
1000
5000
1000000
Lagrange multipler


Lagrange multipler, Weight and Value - 50 Items
Weight
Value
2000
1500
1000
500
0
Weight/Value
1384
1221
1412
941
1267
1590
1118
1077
1221
653
650
649
650
649
649
649
650
650
1
5
10
20
40
80
500
1000
5000
Lagrange Multipler

Conclusion

We have in this paper takes a look at how quantum computing can be used to solve optimization problems. We have compared it to the simple method of using brute force and see the difference between them. I believe that we can safely say that using quantum annealing to solve optimization problems is a more efficient way. However, because of the nature of quantum computing, the results are not necessarily as accurate as those from the brute force method. This means that if accuracy is valued, then this may not be the best solution. However, if a close approximation is acceptable, then using quantum computers to solve optimization problems is the way to go.

Appendix 1 - References

References

Hanly, B. (2021, February 2). Implementing the Knapsack problem on D-Wave's DQM Solver. *MIT 6.S089—Intro to Quantum Computing*. https://medium.com/mit-6-s089-intro-to-quantum-computing/implementing-the-knapsack-problem-on-d-waves-dqm-solver-c210d22e0622

Lucas, A. (2014). Ising formulations of many NP problems. *Frontiers in Physics*, *2*. https://doi.org/10.3389/fphy.2014.00005

Ruiz, A. de la F. (2014, April 9). *Quantum annealing*. ArXiv.Org. https://arxiv.org/abs/1404.2465

Systems, D.-W. (2015a). What is Quantum Annealing? [Video]. In *YouTube*. https://www.youtube.com/watch?v=zvfkXjzzYO0&t=183s

Systems, D.-W. (2015b). How the quantum annealing process works [Video]. In *YouTube*. https://www.youtube.com/watch?v=UV_RlCAc5Zs&t=179s

Systems, D.-W. (2016). Physics of quantum annealing - Hamiltonian and eigenspectrum [Video]. In *YouTube*. https://www.youtube.com/watch?v=tnikftltqE0&t=305s

Systems, D.-W. (2020). Quantum Computing Tutorial Part 1: Quantum annealing, QUBOs and more [Video]. In *YouTube*. https://www.youtube.com/watch?v=teraaPiaG8s&t=2245s

Systems, D.-W. (2021). Quantum programming tutorial [Video]. In *YouTube*. https://www.youtube.com/watch?v=jTDnGoxoc9Y&t=1503s

Appendix 2 - Code

Brute Force code

```
#!/usr/bin/env python3

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 17 14:56:50 2022

@author: sulemanhersi
"""
import itertools
import more_itertools
import numpy as np
import time


# Set up scenario
z = int(input("Press ´1´ for 7 items ´2´for 21 items and ´3´ for 50 items: "))

if z == 1:

        max_weight = 70
        weight =[12,27,11,17,20,10,15]
        value=[35,85,30,50,70,80,55]
        i = 7
        objects = np.arange(i)
```

```
if z == 2:

        max_weight = 150
        weight =[12,27,11,17,20,10,15,95,70,85,31,100,1,7,53,35,11,64,26,4,34]
        value=[35,85,30,50,70,80,55,77,44,15,67,75,91,2,46,16,68,35,53,84,85]
        i = 21
        objects = np.arange(i)



if z == 3:

        max_weight = 650
        weight
=[12,27,11,17,20,10,15,95,70,85,31,100,1,7,53,35,11,64,26,4,34,12,27,11,17,20,10,15,95,70,85,3
1,100,1,7,53,35,11,64,26,4,34,7,53,35,11,64,26,4,34]

value=[35,85,30,50,70,80,55,77,44,15,67,75,91,2,46,16,68,35,53,84,85,35,85,30,50,70,80,55,77,4
4,15,67,75,91,2,46,16,68,35,53,84,85,2,46,16,68,35,53,84,85]
        i = 68
        objects = np.arange(i)



g = 0

solution_weight = []
solution_value = []

print("START")
```

```
start_time = time.time()
# Run through all possible combinations
n = i

knapsack = itertools.product([0,1], repeat=n)
knapsack_combo = list(knapsack)
valid_combo = []


# Check if the combination is valid

while g < len(knapsack_combo):

        index = more_itertools.locate(knapsack_combo[g], lambda x: x == 1)

        items_picked = list(index)

        i = 0

        check_weight = 0
        check_value = 0

        if not items_picked:
                print()

        else:

                while i < len(items_picked):

                        check_weight = check_weight + weight[items_picked[i]]
                        check_value = check_value + value[items_picked[i]]
```

```
                i = i +1

            if check_weight == max_weight:

                    solution_weight.append(check_weight)
                    solution_value.append(check_value)
                    valid_combo.append(knapsack_combo[g])




        g = g + 1


# Find the best combination
g = 0
best_combo = 0
all_best_combo = []
check_value = 0
check_weight = 0

while g < len(solution_value):


        if check_weight <= solution_weight[g] and check_value <= solution_value[g]:

            check_value = solution_value[g]
            check_weight = solution_weight[g]

            best_combo = g
```

```
        g = g +1

g = 0
while  g < len(valid_combo):

        if check_weight == solution_weight[g] and check_value == solution_value[g]:

                all_best_combo.append(valid_combo[g])

        g = g + 1

print(time.time() - start_time,"seconds")
print(weight)
print()
print(value)
print()
print(all_best_combo)
print()
print(valid_combo[best_combo])
print()
print(solution_weight[best_combo])
print()
print(solution_value[best_combo])
```

Quantum annealing code

```
#!/usr/bin/env python3

from dwave.system import DWaveSampler, EmbeddingComposite
from dimod import BinaryQuadraticModel

# Set up scenario
import numpy as np


z = int(input("Press ´1´ for 7 items ´2´for 21 items and ´3´ for 50 items: "))


if z == 1:

        max_weight = 70
        weight =[12,27,11,17,20,10,15]
        value=[35,85,30,50,70,80,55]
        i = 7
        objects = np.arange(i)


if z == 2:

        max_weight = 150
        weight =[12,27,11,17,20,10,15,95,70,85,31,100,1,7,53,35,11,64,26,4,34]
        value=[35,85,30,50,70,80,55,77,44,15,67,75,91,2,46,16,68,35,53,84,85]
        i = 21
        objects = np.arange(i)
```

```
if z == 3:

        max_weight = 650
        weight
=[12,27,11,17,20,10,15,95,70,85,31,100,1,7,53,35,11,64,26,4,34,12,27,11,17,20,10,15,95,70,85,3
1,100,1,7,53,35,11,64,26,4,34,7,53,35,11,64,26,4,34]

value=[35,85,30,50,70,80,55,77,44,15,67,75,91,2,46,16,68,35,53,84,85,35,85,30,50,70,80,55,77,4
4,15,67,75,91,2,46,16,68,35,53,84,85,2,46,16,68,35,53,84,85]
        i = 50
        objects = np.arange(i)

# Build a variable for each object
x=[]

for o in objects:
   x.append(f'O{o}')


# Create bqm
bqm = BinaryQuadraticModel("BINARY")


#Objective
for o in objects:
   bqm.add_variable(x[o], -1*value[o])



# Add Constraint
c1 = [(x[o], weight[o]) for o in objects]
```

```
bqm.add_linear_equality_constraint(c1,constant = -max_weight, lagrange_multiplier=100)


sampler = EmbeddingComposite(DWaveSampler())
sampleset = sampler.sample(bqm, num_reads=1000)

sample = sampleset.first.sample
energy = sampleset.first.energy



total_value = 0
total_weight = 0
printout = ""

# Creat print out
for o in objects:
    printout += str(sample[x[o]])
    total_value += sample[x[o]]*value[o]
    total_weight += sample[x[o]]*weight[o]


print(printout)
print("\nTotal weight:\t", total_weight)
print("\nTotal Value:\t", total_value)
print("\nEnergy:\t", energy)
```

Appendix 3 - Raw Data

Tool	Items	Lagrange multipler	Processing time (seconds)	Weight	Value	Energy
BF	7	0	0.00055384635925292900	70	265	0
BF	21	0	11.85320687294 00	150	698	0
BF	50	0	30.0000000000	0	0	0
QA	7	1	0.1201195700	72	290	-286
QA	7	5	0.1208395700	72	290	-270
QA	7	10	0.1337995700	70	265	-265
QA	21	1	0.1697595700	150	671	-671
QA	21	5	0.1147999700	151	562	-557
QA	21	10	0.1647203700	150	581	-581
QA	21	40	0.1035195700	150	551	-551
QA	21	80	0.1331999700	151	615	-535
QA	21	1000	0.1381003700	150	498	-498
QA	21	5000	0.1356195700	150	596	-596
QA	21	1000000	0.1649207700	150	477	-477
QA	50	1	0.1960631700	653	1384	-1375
QA	50	5	0.1879215700	650	1221	-1221
QA	50	10	0.1970995700	649	1412	-1402
QA	50	20	0.1551799700	650	941	-941
QA	50	40	0.1898991700	649	1267	-1227
QA	50	80	0.1536007700	649	1590	-1510
QA	50	500	0.1832039700	649	1118	-618
QA	50	1000	0.1731631700	650	1077	-1077

QA	50	5000	0.1543611700	650	1221	-1221