Title of the article

Giuseppe Sorrentino, Marco Tonnarelli, Marco Venere

Politecnico di Milano
Milan, Italy
giuseppe.sorrentino@mail.polimi.it
marco.tonnarelli@mail.polimi.it
marco.venere@mail.polimi.it

Abstract

In recent years, quantum annealers have been of great interest to researchers and scholars. Our contribution, with the present paper, is a comparison of the performance of the two most important D-Wave quantum annealers: 2000Q and Advantage, as well as the development of a Set Packing Problem algorithm. We test its efficiency on both systems using increasingly complex instances of the problem. We present our results, making important considerations on time and space complexity, as well as the energy of the solutions provided by the quantum annealers.

Keywords: 2000Q, Advantage, Set Packing, quantum annealing, complexity, D-Wave

1 Introduction

Over the past forty years, the frontier of information technology is focused on developing more and more effective architectures with an extremely high computational power: quantum computers. From the very beginning of the ‘80s, when Paul Benioff proposed the first model of the quantum touring machine, several improvements have been achieved. Nowadays, one of the most promising machines is the quantum annealer, which aims to heuristically solve difficult combinatorial optimization problems[10]. To show the capabilities of this technology, in the present article a possible solution for the Set Packing Problem is shown. Even if it is extremely notorious in literature and several solutions have been proposed using classical computers, in the algorithm here presented two different kinds of quantum annealers have been used, and their results have been compared. To execute the algorithm, the quantum annealers of D-Wave have been employed [1].In particular, we used Leap, a real-time Quantum Application Environment. It is a cloud-based platform giving application developers real-time access to a quantum computer. Here, two powerful quantum annealers are available: the 2000Q and the Advantage. They are controllable supercomputers that work at extremely low temperatures. In particular:

- the **2000Q** has a footprint of approximately 10’ x 7’ x 10’ (L x W x H). Its physical enclosure houses sophisticated cryogenic refrigeration, shielding, and I/O systems to support a single thumbnail-sized QPU. Most of the physical volume of the system is required to accommodate the refrigeration plant and to provide easy service access. For quantum effects to play a role in computation, the QPU requires an extreme, isolated environment. It works at a temperature of 15 mK and it has up to 2048 qubits and 6016 couplers;
- the **Advantage** system is the first and only quantum computer designed for business. It is the 5th-generation Advantage quantum computer that was built from the ground up with a new processor with over 5,000 qubits and a 15-way qubit connectivity, empowering enterprises to solve their

largest and most complex business problems. The main strength of the advantage is the higher number of qubits, which makes it computationally more powerful than the 2000Q, while the main drawback is the high noise which affects its data.

In addition to supercomputers, in the first phase of the project, we also used the D-Wave hybrid sampler [8] to test the algorithm. Essentially, it is a framework which allows solving an optimization problem by decomposing it into two or more solutions which run in parallel. These solutions, using the D-Wave hybrid sampler, may or may not involve the supercomputers previously mentioned. The use of this framework allowed us to be certain about the feasibility of the algorithm. Then, we left the hybrid framework to exclusively focus on 2000Q and Advantage.

In the first section of this paper, the Set Packing Problem, the specific formalization provided by Glover for quantum computing[7] is deeply described. Afterwards, there is a description of the algorithm and of the format used to represent instances of the problem. The latter was essential to implement a proper problem generator for testing. Subsequently, in the fourth chapter, the results of our experiments have been shown to highlight the effectiveness of both machines. Finally, in the last section, there is a comparison between them, to put a spotlight on their strengths and drawbacks.

2 The Set Packing Problem

The main focus of our research is the acceleration of the resolution of a set packing problem. This is a combinatorial programming problem which has been extensively studied in recent years[2]. The formulation of the problem is the following: given a set of N finite subsets, a pack is a collection of all the subsets among the N ones which are mutually disjoint. The main objective of the problem is to find the packing of maximum size[5]. Set packing is an NP-complete maximization problem. Here follows the problem formalization given by Glover [1]:

$$max \sum_{j=1}^{n} w_j x_j$$
$$\text{st}$$
$$\sum_{j=1}^{n} a_i j x j \leq 1 \forall j \in 1..m$$

where a_ij are 0/1 coefficients, w_j are weights and x_j variable are binary.

In order to make the algorithm work on quantum annealers, it is necessary to reformulate the problem as a QUBO model (Quadratic Unconstrained Binary Optimization). The QUBO model is a special framework specifically designed for Combinatorial Optimization tasks. In this kind of algorithms, a large number of decisions must be made and these decisions yield a corresponding objective function value. Being the Set Packing problem NP-complete, finding the optimal solution might be unfeasible as the size of the instance increases. However, it is possible to find very good but not necessarily optimal solution in a reasonable amount of computational time.

The idea behind the QUBO formulation is to use the following objective function:

$$minimize/maximize\ y = x^t Q x$$

where x is a vector of binary decision variables and Q is a square matrix (symmetric or upper triangular) of constraints. The constraints of a traditional optimization problem become, in the QUBO model, a set of quadratic penalties introduced in the previously mentioned objective function. The penalties are structured in a way that they are equal to zero for feasible solutions or equal to some positive amount for unfeasible solutions. For the Set Packing problem, we obtained the following QUBO representation:

$$max\ x^t Q x \quad .$$

In order to achieve our objectives, we implemented the set packing problem using the Leap platform provided by D-wave systems, where we used a built-in function which is able to map a traditional optimization problem into a QUBO model in such a way that it can be run on quantum annealers.

3 The Algorithm

The definition of the algorithm for the solution of a generic Set Packing problem has been based on the platform and libraries developed by D-Wave (i.e., Leap IDE [14] and D-Wave System [11] respectively). The testing has been focused on two different quantum annealers: 2000Q and Advantage.

Firstly, it was necessary to define a formal descriptive paradigm of a Set packing problem, able to rigorously express all the subsets belonging to the set and the related constraints, according to the Grover formalization.

Secondly, the first phase of the development of the algorithm has started, in which the code has been tested on a single instance of the problem, for both architectures, and we analyzed the spatial and time complexities.

Subsequently, the second phase has taken place, in which we tuned the parameters of the algorithm by executing several experiments, for both architectures. We, therefore, studied their space and time complexity and compared them to the previous ones.

3.1 The problem format and generator

In order to elaborate an effective algorithm for the solution of a generic Set Packing problem, we defined a formal paradigm, aiming at describing the instance of the problem, with the subsets and their weights, as well as all the constraints that exist among them.

During the definition of such a format, priority has been given to two important language characteristics:

1. **expressiveness**: the chosen language should be able to express, clearly and concisely, all the features of the problem, so as to be, at the same time, human-readable and unambiguous;
2. **scripting-compliance**: the chosen language should be thoroughly supported by the standard libraries of the majority of the scripting languages, in order to ease its use and spreading. In addition, being scripting-compliant also reduces the probability of the presence of vulnerabilities and errors in the code related to translation, for standard libraries are widely tested against such risks;
3. **parsing speed**: the chosen language should be parsable very fast, as it will be used very extensively.

After an extensive analysis, we decided that the most convenient format to adopt is JSON (JavaScript Object Notation, [3]), as it is strongly expressive, widely supported by the majority of scripting languages, and parsable at high velocity [4].

In particular, the generic format is defined as an array of objects, each of which describes an instance of a problem. The usage of arrays allows for multiple instances to be considered at the same time. Each of these objects contains two fields: *subsets*, which is an array of the subsets belonging to the set, and *constraints*, which is an array of the constraints among the subsets.

The subsets are defined as objects, containing a *name*, which is a string, and a *weight*, which is a number, and is optional (its default value is 1). The constraints are objects as well, containing an attribute *sets*, that is an array of strings, each referring to a previously defined set.

An example of an instance of the problem is the following:

```
[{
    "subsets":
        [
            {"name": "1E8", "weight": 8},
            {"name": "Z7MF"},
            {"name": "F", "weight": 6},
```

```
            {"name": "B", "weight": 6},
            {"name": "1MS8", "weight": 5},
            {"name": "ZI"}
        ],
    "constraints":
        [
            {"sets": ["B", "ZI"]},
            {"sets": ["1MS8", "ZI", "F"]},
            {"sets": ["1E8", "B", "Z7MF", "F", "1MS8", "ZI"]}
        ]
}]
```

Once the format has been defined, we wrote a script in order to generate random instances of a set packing problem. Since, for complexity analysis, it is necessary to control the size of the instance (i.e., the number of subsets considered in the problem), such a parameter is the input of the script. The procedure returns a file containing the instance of the problem, written according to the format described above. Thanks to this generator, it has been possible to automatically produce several instances of the

Algorithm 1 Set Packing Problem generator

```
1:  procedure GenerateSetPackingProblem(filename, N)
2:      global variables
3:          set S := ∅                                   ▷ Set of subsets names
4:          set W := ∅                                   ▷ Set of subsets weights
5:          set C := ∅                                   ▷ Set of constraints
6:      end global variables
7:      while size(S) ≠ N  do
8:          S ← random_string()                          ▷ Create random subset name
9:          W ← random_number_or_null()                  ▷ Create random optional subset weight
10:     int size_of_C ← random_positive_up_to_(N)        ▷ Generate random size of C
11:     while size(C) ≠ size_of_C do
12:         C ← random_constraint_from_subsets(S)        ▷ Create random constraint from defined subsets
13:     json J ← create_JSON(S, W, C)                    ▷ Create JSON object from given sets, by using defined format
14:     file F ← create_file(filename, J);               ▷ Create file with content J
15:     return F                                         ▷ return file
```

problem, of different sizes, in order to evaluate the space and time complexity of the algorithm for the resolution of the problem.

3.2 The first phase

In the first phase of the development of the algorithm, we performed a study of the D-Wave Leap IDE platform and decided to adopt the Python programming language, since it is the language chosen by the D-Wave environment.

The Set Packing problem has been modelled as an object of the *SetPacking* class, which embodies all the data related to subsets and constraints, and paves the way for the usage of the *BinaryQuadraticModel* [6], that is the main representation of a Binary Quadratic Model in the D-Wave System library. In particular, all the subsets are represented as *variables* of the problem, and the constraints as *interactions* among variables.

After the program is correctly represented in memory, the sampling stage begins, by choosing one of the three topologies on which this study is based:

1. **2000Q**: It is invoked by the *EmbeddingComposite* object, by specifying *chimera* as topology type.

2. **Advantage**: it is invoked by the *EmbeddingComposite* object, as a default choice, thus no topology type needs to be specified.

3. **Hybrid**: it is invoked by the *LeapHybridSampler* object. It is a hybrid architecture.

Once the sampler has been chosen [13], the sampling stage can take place, and its results, received as a sample set, can be analyzed. The platform allows the use of a specific tool, called *Inspector* [9], made available for the 2000Q and Advantage architectures, that let developers visualize the problems received from the machine and control several parameters, in addition to the quantum bits involved in the computation. We considered such parameters for the evaluation of the space and time complexity of the algorithm, and because of the absence of an inspector screen for the *LeapHybridSampler*, for such a topology it was not possible to show any computational trend.

As a matter of completeness, the pseudocode of the resolution of a set packing problem is here reported. The input of the *SolveSetPackingProblem* is a file name, corresponding to a file that has been previously written in accordance with the aforementioned format.

Algorithm 2 Set Packing Problem solver

Require: a file with name $filename$, consistent with the defined format

procedure SolveSetPackingProblem($filename$, N)
  parameters
    $penalty$: the penalty associated to the violation of a constraint
    $sampler$: the sampler, and related solver, associated to the desired architecture.
  end parameters
  global variables
    P, SetPackingProblem
    bqm, BinaryQuadraticModel
  end global variables
  P := read_sanitized_file($filename$) ▷ Read a file, verify the format and instantiate P
  for each subset ∈ P.subsets **do**
    bqm.add_variable(subset.name, subset.weight) ▷ Add subset and weight as a variable in the binary quadratic model
  for each constraint ∈ P.constraints **do**
    for each i ∈ constraint **do**
      for each j ∈ constraint, j < i **do**
        bqm.add_interaction(i, j, $penalty$) ▷ Add an interaction for each pair of sets in a constraint
  sampleset := $sampler$.sample(bqm) ▷ Sample the binary quadratic model with $sampler$
  show_inspector(sampleset) ▷ Show the inspector for the sampleset, if available
  return sampleset ▷ Return the sampleset

2000Q: space and time complexity

For what concerns the 2000Q architecture, the maximum size of a set packing problem that has been successfully computed is circa 60. In order to consider the space complexity, we checked the inspector for increasing sizes of the problem (N = 10, 20, 30, 40, 50, 60). For these values, we here report the maximum chain length that has been obtained (figures 1 to 6).

In order to evaluate the time complexity of the algorithm, we tested it on increasing sizes of the problem, and analyzed the QPU time spent by the machine (i.e., the time spent by the QPU, Quantum Processing Unit, to solve the problem [15]). The following plots show the trends of the time complexity:

In figure 7, it can be noticed that the overall trend is a linear increase, even though some spikes are present, due to the intrinsic noise belonging to the machines [12]. By removing such outliers, the

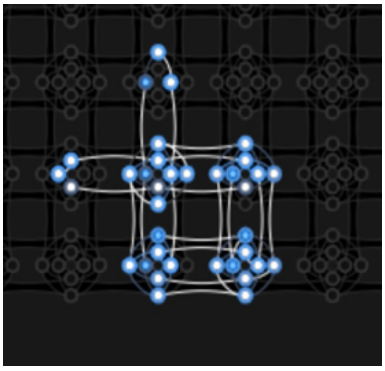

Figure 1: N=10 subsets
max chain length = 4

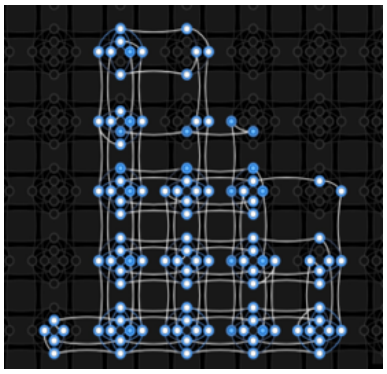

Figure 2: N=20 subsets
max chain length = 7

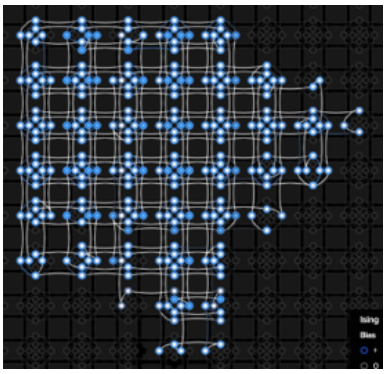

Figure 3: N=30 subsets
max chain length = 12

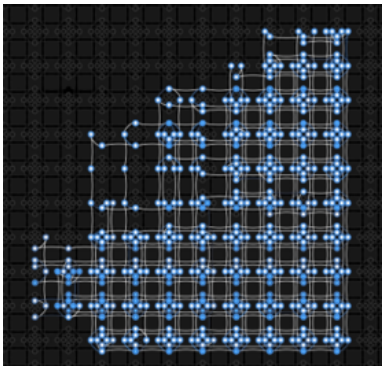

Figure 4: N=40 subsets
max chain length = 15

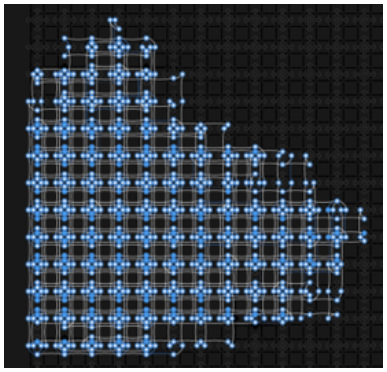

Figure 5: N=50subsets
max chain length = 21

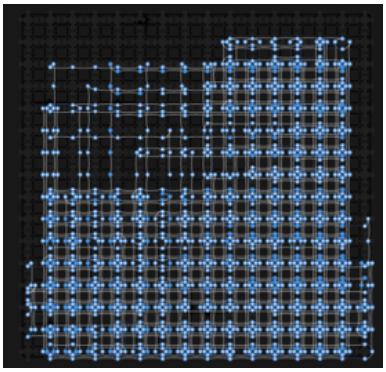

Figure 6: N=60subsets
max chain length=31

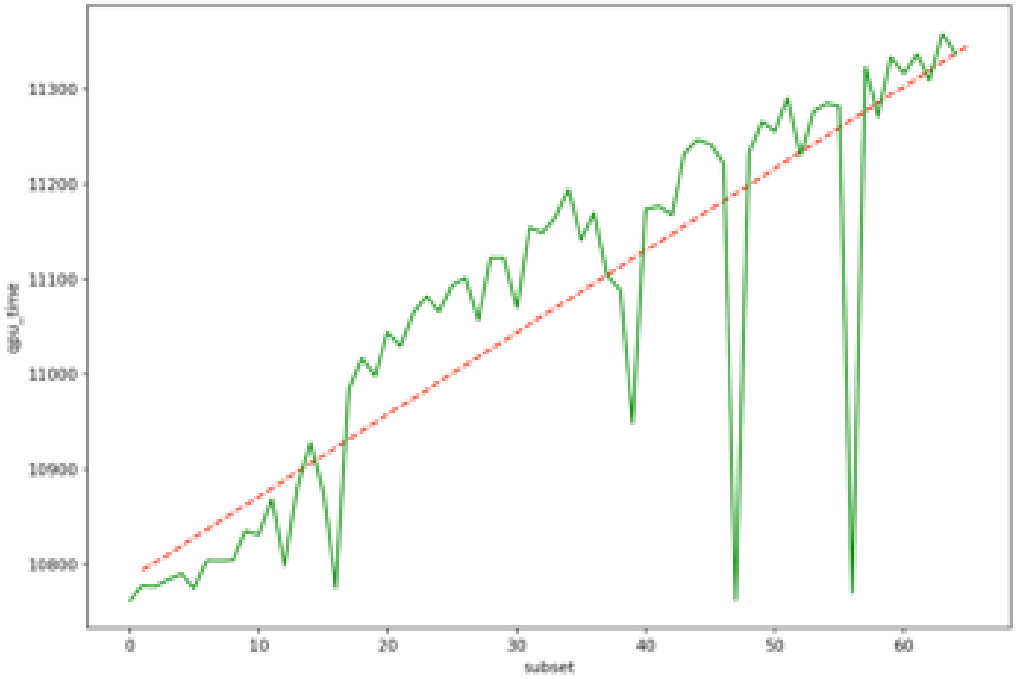

Figure 7: Time complexity, with outliers (real values in green, general trend in red)

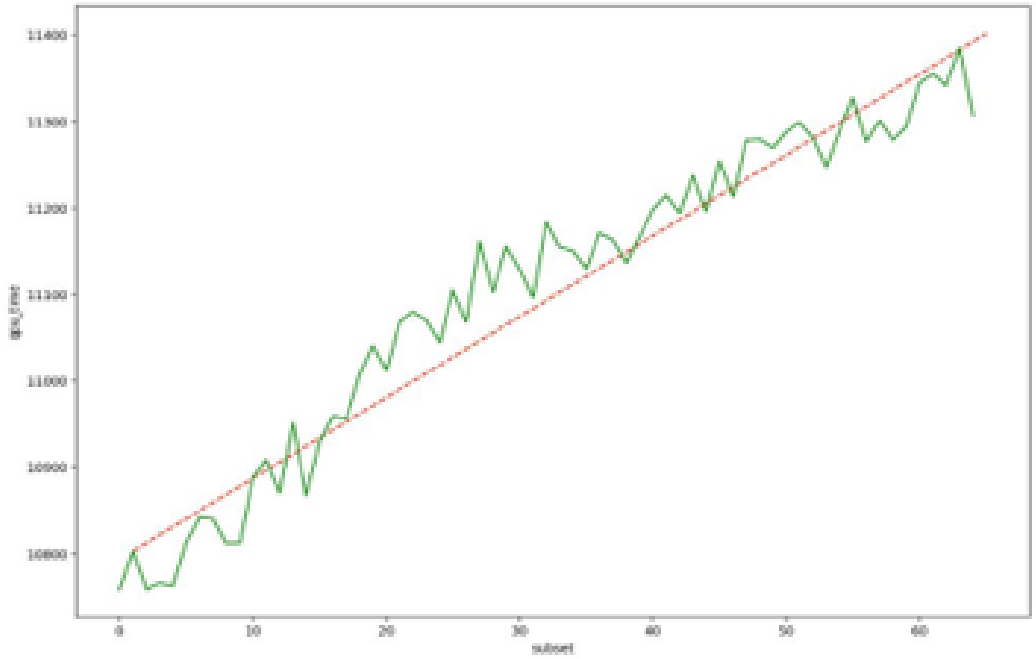

Figure 8: Time complexity, without outliers (real values in green, general trend in red)

linearity is even more noticeable, as figure 8 shows.

Advantage: space and time complexity

We made analogous considerations for the Advantage architecture. In this case, the maximum size for an instance to be successfully computed is about 150. The space complexity has been observed for N = 10, 40, 70, 100, 130, 150.

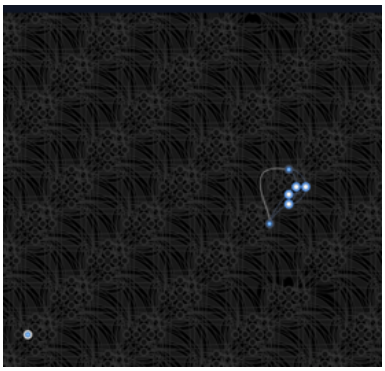

Figure 9: N=10 subsets
max chain length = 2

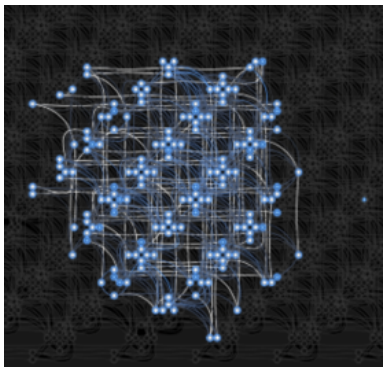

Figure 10: N=40 subsets
max chain length = 7

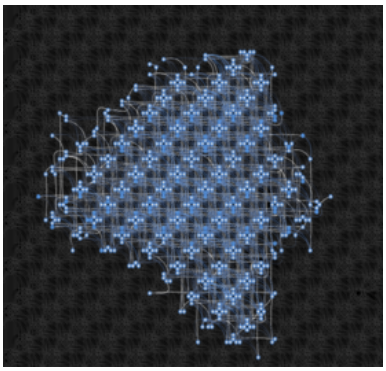

Figure 11: N=70 subsets
max chain length = 13

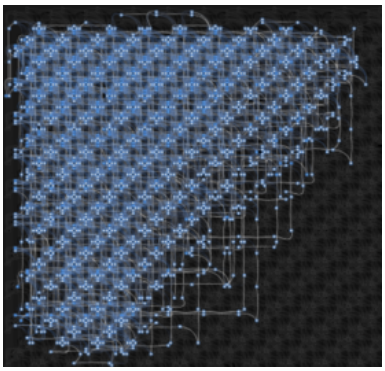

Figure 12: N=100 subsets
max chain length = 29

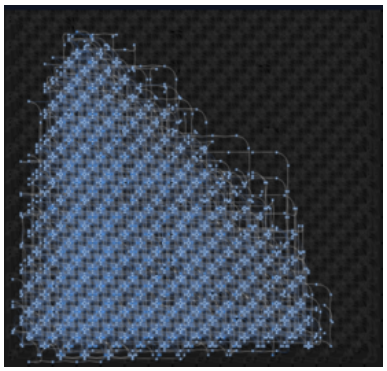

Figure 13: N=130 subsets
max chain length = 29

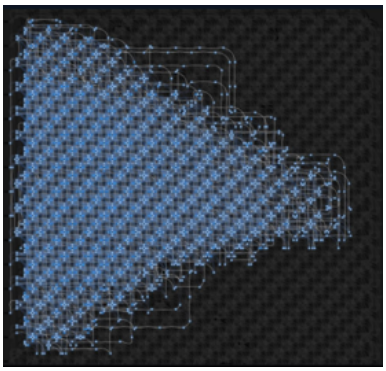

Figure 14: N=150 subsets
max chain length=32

In order to evaluate time complexity, the same plots as for the 2000Q architecture have been taken.

As figure 15 shows, the presence of some particular spikes prevents a correct analysis of the general trend, due to the noise of the Advantage. By removing such outliers, as in figure 16, there can be observed a general linear increase, even though, once again, the great amount of noise of the architecture produces a more irregular pattern.

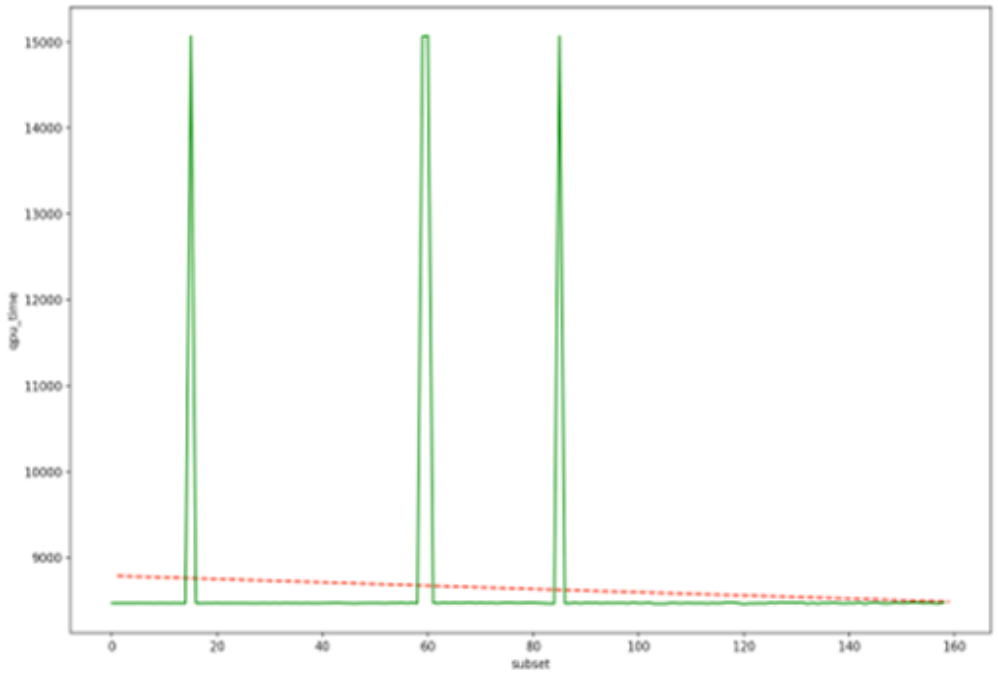

Figure 15: Time complexity, with outliers (real values in green, general trend in red)

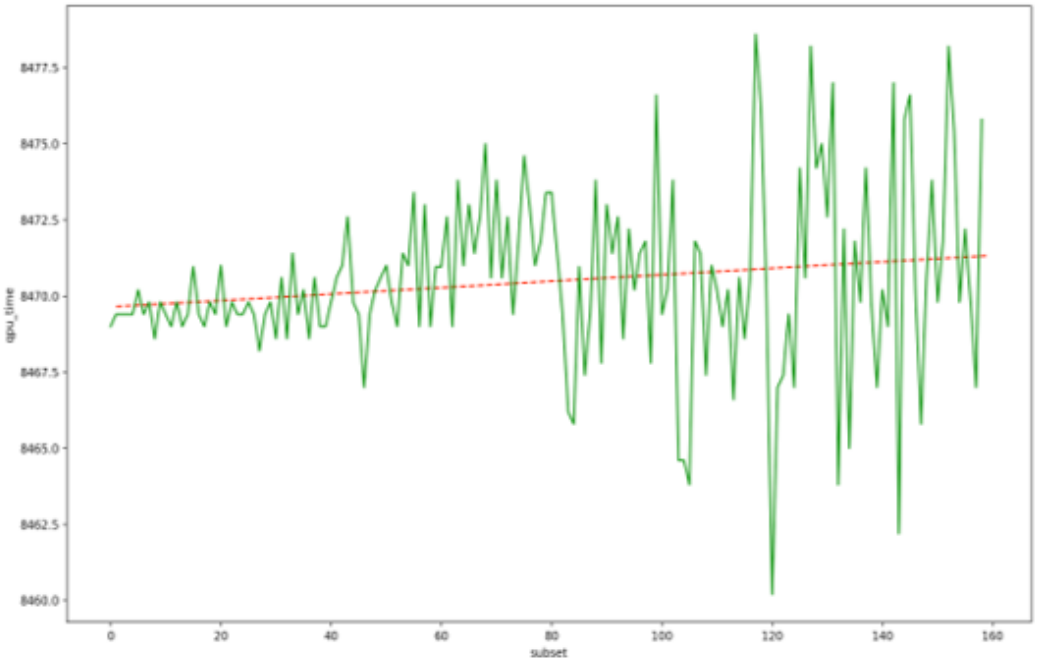

Figure 16: Time complexity, without outliers (real values in green, general trend in red)

3.3 The second phase

After the first phase, in which the algorithm and the problem generator have been tested, we focused on improving the simulations and the experiments modelling some useful parameters. In particular, we modelled the following values:

- the **num_reads** parameter. It represents the number of samples asked to the solver. By default, it is set to 1 so the solver tries to find just one sample. To get a more reliable result, we set this param to 100.
- the **chain_strength** parameter. It represents the strength of the chain. For the sake of clarity, let us specify that a solution is said to be correct and trustable if it has the minimum possible number of chain breaks. Considering this, the chain_strength parameter "forces" the solver to find a solution which is at least as strong as specified. We must specify that choosing an excessively low or high value of chain strength has negative effects. In particular, a too low value leads to a too high number of chain breaks while a too high value leads to a change in the nature of the problem[16]. Therefore, to choose a proper value, we used the method uniform_torque_compensation [11], which computes the chain strength that attempts to compensate for the torque that would cause the chain breaks.
- the **pre_factor** parameter. It is a parameter of uniform_torque_compensation and it allows the method to find the best value of chain_strength. By default, its value is set to 1.414 but, normally, it belongs to a range [0.5,2.0]. By doing several experiments, the optimal value for which the minimum number of chain breaks resulted is 2.0.

Here follows the result of the experiment performed in order to analyze the space complexity of each instance.

The problems with a number of subsets belonging to [1,60] have been solved with 2000Q and Advantage while the problems with a number of subsets belonging to [61,150] have been solved with Advantage only, since they required an excessive amount of resources to be successfully computed, more than what 2000Q can provide.

2000Q : number of chain breaks = 0

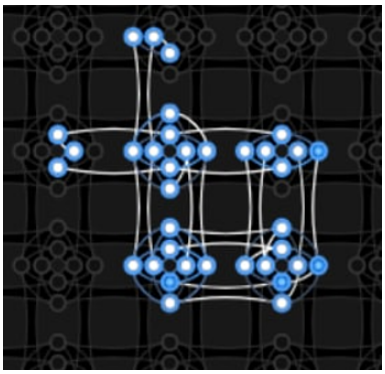

Figure 17: N=10 subsets
max chain length = 4
max chain strength=120.72
physical qubits = 33

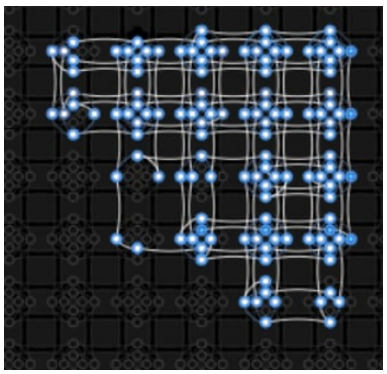

Figure 18: N=20 subsets
max chain length = 8
max chain strength=95.47
physical qubits = 132

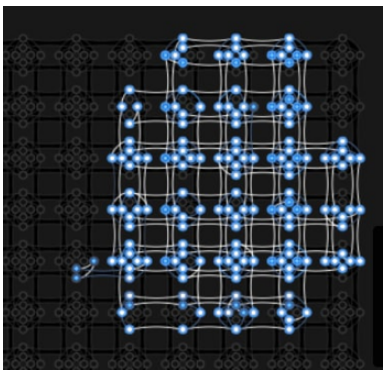

Figure 19: N=30 subsets
max chain length = 9
max chain strength=52.21
physical qubits = 178

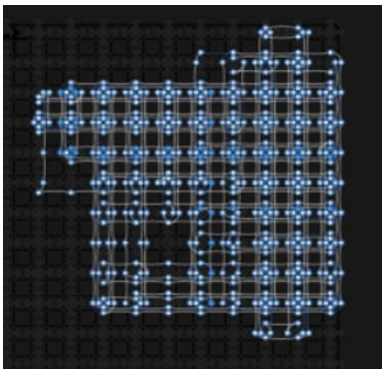

Figure 20: N=40 subsets
max chain length = 17
max chain strength=273.18
physical qubits = 528

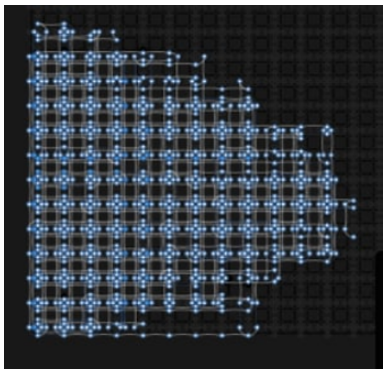

Figure 21: N=50 subsets
max chain length = 25
max chain strength=701.68
physical qubits = 888

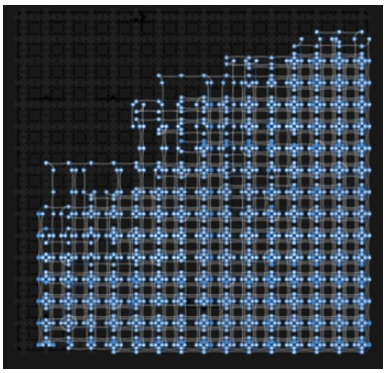

Figure 22: N=60 subsets
max chain length=33
max chain strength=1285.3
physical qubits=1295

The Advantage: number of chain breaks = 0

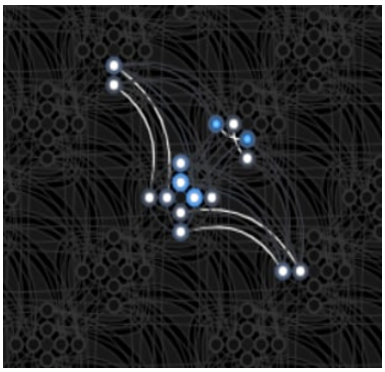

Figure 23: N=10 subsets
max chain length = 2
max chain strength=100.40
physical qubits = 16

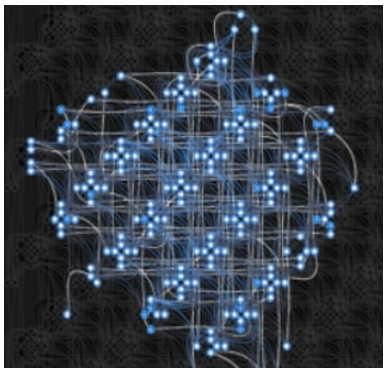

Figure 24: N=40 subsets
max chain length = 7
max chain strength=114.09
physical qubits = 206

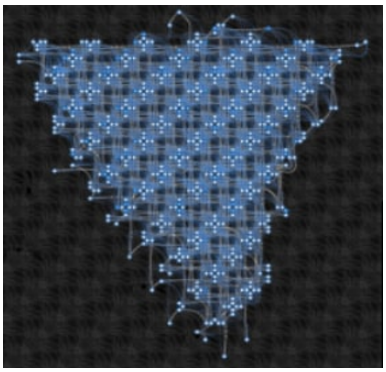

Figure 25: N=70 subsets
max chain length = 11
max chain strength=1839.2
physical qubits = 553

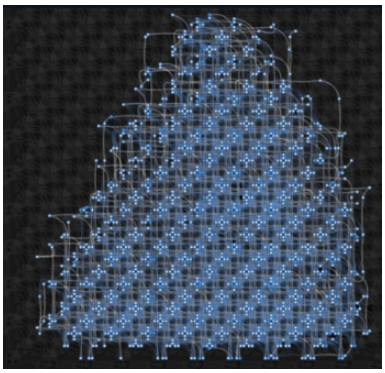

Figure 26: N=100 subsets
max chain length = 19
max chain strength=2722.5
physical qubits = 1324

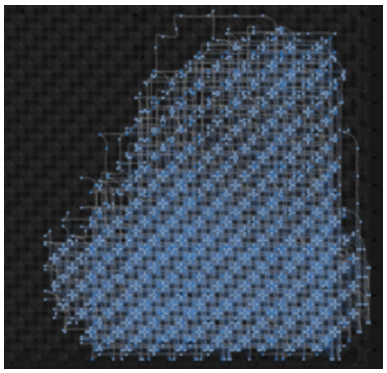

Figure 27: N=130 subsets
max chain length = 28
max chain strength=3011.2
physical qubits = 2356

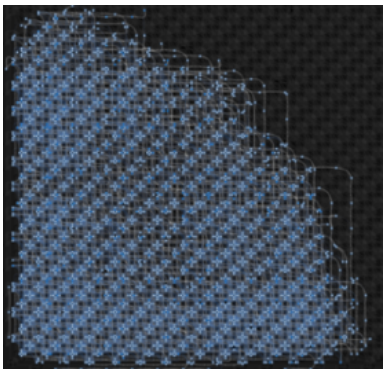

Figure 28: N=150 subsets
max chain length=34
max chain strength=2350.4
physical qubits=3309

In order to evaluate the time complexity, with the same aforementioned settings, we chose to repeat the experiments ten times. Then we made the average of the results achieved in order to collect a more reliable result. Thanks to Leap we could print on a file several data about timing and energy and using python scripts we obtained the following results:

2000Q

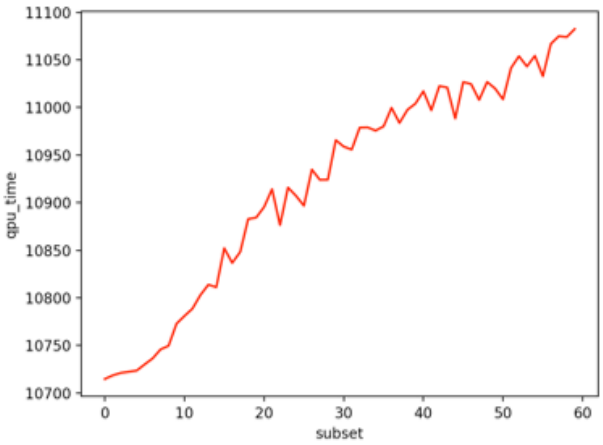

Figure 29: 2000Q Time complexity: mean

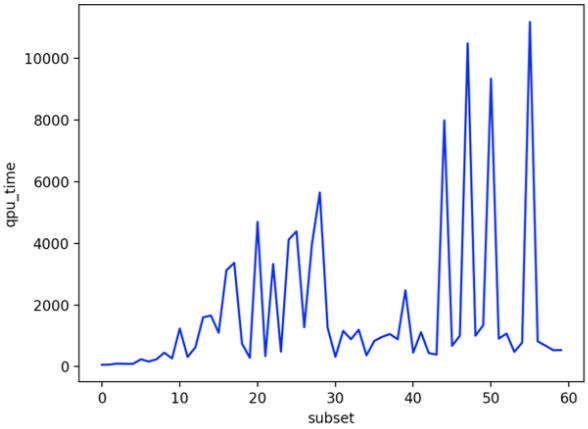

Figure 30: 2000Q Time complexity: variance

As it can be noticed in figures 29 and 30, the mean and variance of the QPU time has been computed for each value of N. The resulting mean increases as a function of the number of subsets defined in the problem. The variance increases as well, even though more irregularly.

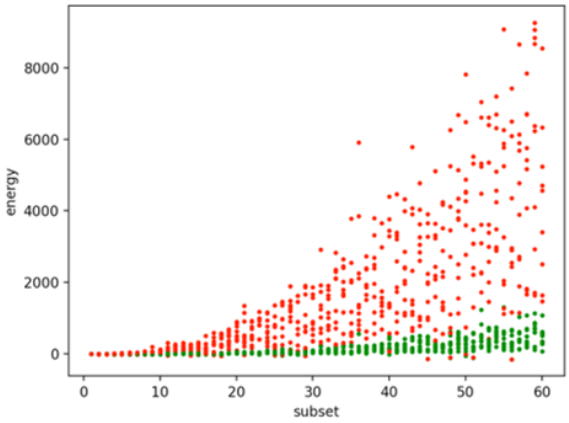

Figure 31: 2000Q Max and min energy:values

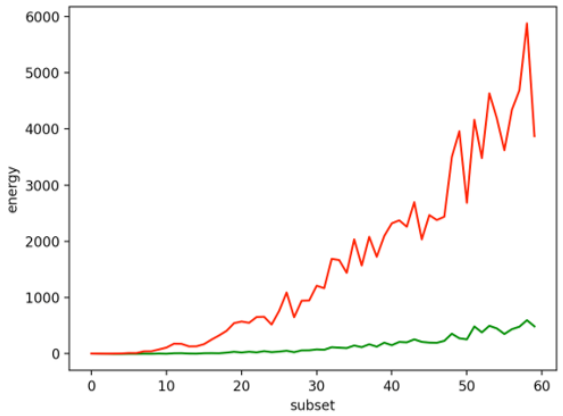

Figure 32: Max and min energy: mead

For what concerns the maximum and minimum value of the energy, for each value of N, it can be seen that the mean value for the minimum energy remains approximately constant, while the maximum value increases as a function of N.

Advantage

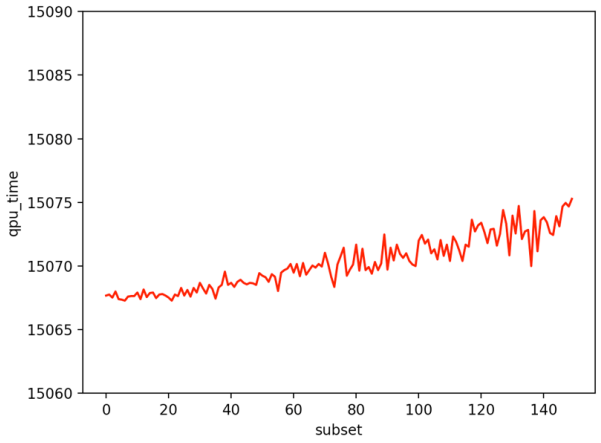

Figure 33: Advantage Time complexity: mean

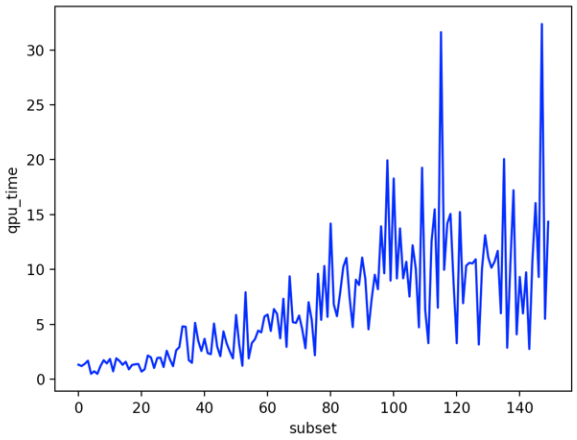

Figure 34: Advantage Time complexity: variance

As it can be noticed in figures 33 and 34, we calculated the mean of the ten executions for each N. The resulting mean increases as a function of the number of subsets, and the variance increases as well, even though more irregularly. In the following pictures, instead, the max and min energy is shown.

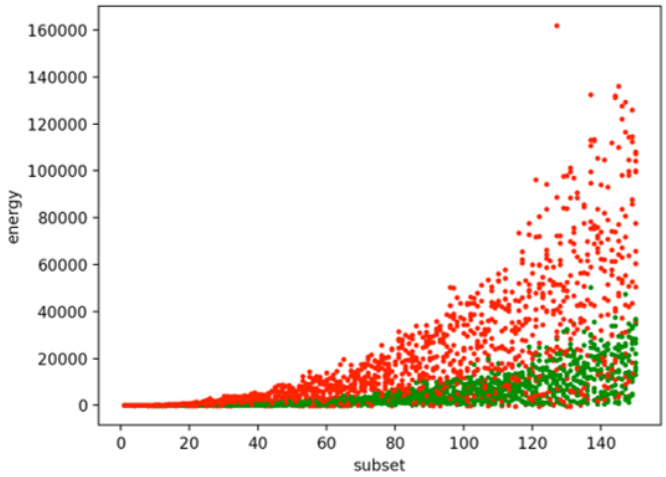

Figure 35: Advantage Max and min energy:values

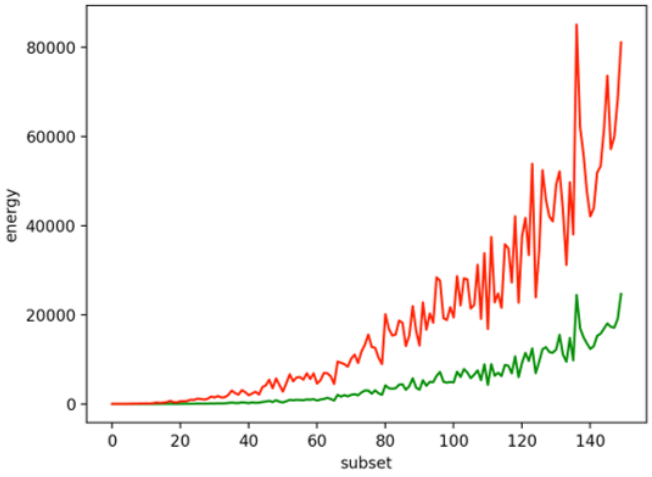

Figure 36: Max and min energy: mean

For what concerns the maximum and minimum values of the energy, as it can be noticed in figures 35 and 36, the minimum energy increases as a function of N, as the maximum energy does, but the former rises significantly slower than the latter.

4 Comparison

Time complexity and space complexity are the main focus of our analysis of the previously described experiments.

For what concerns time, the 2000Q system outscores Advantage. Figures 33 and 29 show the execution time of, respectively, Advantage and 2000Q. The former has, on average,a higher QPU (Quantum Processing Unit) execution time than the latter, as its values vary between 15065 and 15085 milliseconds, while for the 2000Q values range from 10700 to 11100 milliseconds. Nonetheless, we stress the fact that, although 2000Q might seem faster for each experiment, execution time rapidly increases and the values cover a wider range, while Advantage is capable of handling more complex instances of the problem, although it is a little bit slower. We also point out that Advantage is a noisier environment, thus these times also include some delays that are intrinsic to the system. In addition, in our analysis, we do not consider outliers, which indeed are quite common, especially in the Advantage system: during our measurements we faced some spikes in the graphs, such as the one in figure x, which we removed to better visualize the statistical trend.

In general, QPU time to solve problems' instances increases as the square root of the number N of subsets for 2000Q, while it increases linearly for Advantage. To reach these considerations, in our experiments we fitted different models as figures 37 to 42 show, and the best ones we found are $\mathcal{O}(\sqrt{n})$ and $\mathcal{O}(n)$ for, respectively, 2000Q and Advantage, where N is the number of subsets in the instance of the problem.

About space complexity, the greater amount of qubits in the Advantage system allows for more complex problems to be solved: Advantage is more powerful than 2000Q, and this is because it has more qubits; however, the cost of a more powerful system is to be found in the noise. Noise affects the solution found by the machine, and to get more precise and stronger solutions, we tuned the chain strength parameter: by optimizing the value of such a parameter, it has been possible to nullify the number of chain breaks in the considered problems, while the maximum chain lengths result to be similar for both platforms. The maximum number of subsets for problems solved by the systems is 150 for Advantage and 60 for 2000Q (on average, it depends on the particular instance and the number of constraints of

the problem).

Our last considerations are addressed to energy levels of the solution found by the quantum annealers: Advantage has higher orders of magnitude than 2000Q. This is due to the fact that the Advantage system is an environment with higher noise, and, for this reason, it needs higher energy to provide a reliable result. The instability of Advantage is confirmed by the fact that, considering the time complexity but also the energy of the provided solutions, data have more variance than 2000Q for each size of the instances.

2000Q

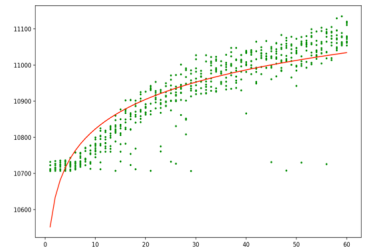

Figure 37:
QPU time=$\mathcal{O}(\log(N))$
R2 = 0.8830

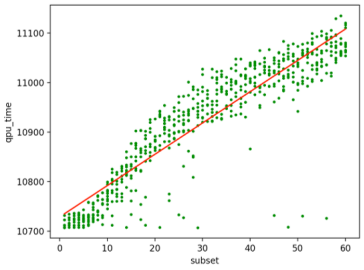

Figure 38:
QPU time=$\mathcal{O}(N)$
R2 = 0.9448

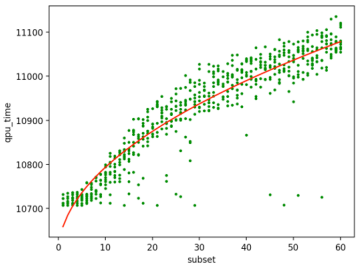

Figure 39:
QPU time=$\mathcal{O}(\sqrt{N})$
R2 = 0.9727

Advantage

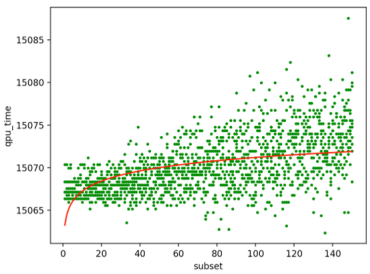

Figure 40:
QPU time=$\mathcal{O}(\log(N))$
R2 = 0.6113

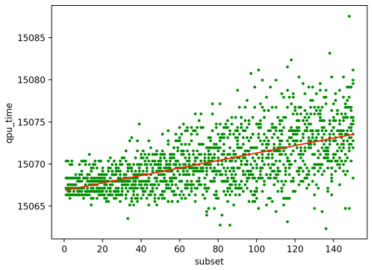

Figure 41:
QPU time=$\mathcal{O}((N))$
R2 = 0.8575

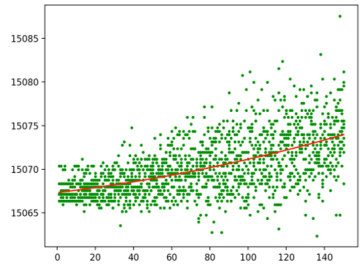

Figure 42:
QPU time=$\mathcal{O}(\sqrt{N})$
R2 = 0.7914

5 Conclusions

This research has been focused on solving Set Packing Problems by using quantum annealers, provided by D-Wave. The framework and the involved architectures have been thoroughly analyzed, in order to get the highest possible efficiency, by lowering the execution time and taking advantage of the available resources at the most.

By using a specific format, based on JSON, and a suitable problem generator, an optimized algorithm has been designed, that takes advantage of the tools provided by the D-Wave System library and that exploits both the two main architecture topologies: 2000Q and Advantage.

After analyzing the time and space complexity of the algorithm, in the two phases of the development,

a comparison between the architectures has been performed, and it resulted that, for what concerns space complexity, Advantage has several pros with respect to the 2000Q, due to the greater amount of qubits belonging to the platform, whilst for the time complexity it resulted that, being equal the size N of the instance, 2000Q executes the algorithm in a faster way. It is important to highlight that Advantage can solve problems of higher size than 2000Q.
As a further observation, the empirical analysis of the experiments has underlined the difference between the time complexity of the 2000Q and Advantage, with the former having an $\mathcal{O}(\sqrt{n})$ complexity, with respect to the $\mathcal{O}(n)$ complexity of the latter. Yet, an even more important result of this study is that the usage of quantum annealers has allowed the solution of a classical NP-complete problem in reasonable times, thus proving, once again, the efficiency of this new frontier of research.

References

[1] Yu Du Fred Glover, Gary Kochenberger. Quantum bridge analytics i: A tutorial on formulating and using qubo models. *4OR*, 2019, May.

[2] Georg Gottlob and Gianluigi Greco. Decomposing combinatorial auctions and set packing problems. *journal of the ACM*, 60(4), August 2013.

[3] JSON. Json. `https://www.json.org/json-en.html`.

[4] Daniel Lemire. Parsing json really quickly: Lessons learned. `https://www.infoq.com/presentations/simdjson-parser/`.

[5] Vangelis T. Paschos. A survey of approximately optimal solutions to some covering and packing problems. *ACM computer survey*, 29(2), June 1997.

[6] D-Wave the quantum computing company. Binary quadratic models. `https://docs.ocean.dwavesys.com/en/stable/concepts/bqm.html`.

[7] D-Wave the quantum computing company. The d-wave 2000q™ quantum computer technology overview. `https://dwavejapan.com/app/uploads/2019/10/D-Wave-2000Q-Tech-Collateral_1029F.pdf`.

[8] D-Wave the quantum computing company. D-wave-hybrid framework documentation. `https://docs.ocean.dwavesys.com/projects/hybrid/en/latest/intro/overview.html`.

[9] D-Wave the quantum computing company. D-wave problem inspector. `https://docs.ocean.dwavesys.com/projects/inspector/en/latest/`.

[10] D-Wave the quantum computing company. D-wave system documentation. `https://docs.dwavesys.com/docs/latest/c_gs_2.html`.

[11] D-Wave the quantum computing company. D-wave system documentation. `https://docs.ocean.dwavesys.com/projects/system/en/stable/reference/generated/dwave.embedding.chain_strength.uniform_torque_compensation.html`.

[12] D-Wave the quantum computing company. Error sources for problem representation. `https://docs.dwavesys.com/docs/latest/c_qpu_ice.html`.

[13] D-Wave the quantum computing company. Getting started with d-wave solvers. `https://docs.dwavesys.com/docs/latest/doc_getting_started.html`.

[14] D-Wave the quantum computing company. Introduction to leap. `https://docs.dwavesys.com/docs/latest/leap.html`.

[15] D-Wave the quantum computing company. Operation and timing. https://docs.dwavesys.com/docs/latest/c_qpu_timing.html.

[16] D-Wave the quantum computing company. Programming the d-wave qpu: Setting the chain strength. https://www.dwavesys.com/media/vsufwv1d/14-1041a-a_setting_the_chain_strength.pdf.