

Solving Currency Arbitrage Problems using D-Wave Advantage2 Quantum Annealer

Lorenzo Mazzei
DII, Univ. of Pisa
L.go L. Lazzarino 1
56122 Pisa, Italy
l.mazzei7@studenti.unipi.it

Giada Beccari
DII, Univ. of Pisa
L.go L. Lazzarino 1
56122 Pisa, Italy
g.beccari3@studenti.unipi.it

Mirko Laruina
DII, Univ. of Pisa
L.go L. Lazzarino 1
56122 Pisa, Italy
m.laruina@studenti.unipi.it

Marco Cococcioni
DII, Univ. of Pisa
L.go L. Lazzarino 1
56122 Pisa, Italy
marco.cococcioni@unipi.it

Abstract—Quantum annealing has emerged as a powerful tool for solving combinatorial optimization problems efficiently, making use of the principles of quantum mechanics. Companies are increasingly investing in the market of quantum computers, providing the users with the possibility to solve these optimization problems by resorting to quantum computers. This paper explores how Quantum Annealing can be applied to the Currency Arbitrage (CA) optimization problem and its comparative performance against classical methods. A key contribution of the work is an original formulation of the CA problem as a QUBO (Quadratic Unconstrained Boolean Optimization) problem. We test the speed of D-wave quantum annealer, using the recently released latest version (Advantage 2).

I. INTRODUCTION

Tackling and solving an optimization problem is a task that can be carried out using multiple algorithms, each of them having their unique strengths and weaknesses. One of them, Simulated Annealing, aims at approximating the global optimum in a large search space. Simulated Annealing is still of interest, because it is one of the most promising concepts to give quantum superiority in Quantum Computing with its corresponding algorithm: Quantum Annealing [1], [2], [3], [4]. Quantum Annealing is a quantum computing technique designed to solve complex optimization problems [5] by exploiting quantum mechanical phenomena. Unlike classical algorithms, Quantum Annealing leverages on quantum superposition and quantum tunneling to explore the vast search space, thereby speeding up the search for optimal solutions. One compelling application of Quantum Annealing is in financial optimization, particularly in solving Currency Arbitrage problems [6], [7]. Currency Arbitrage involves exploiting discrepancies in exchange rates across different currencies to generate profits. Given the dynamic nature of global financial markets, the ability of Quantum Annealing to process vast amounts of interconnected data and detect optimal and near-optimal solutions in small time windows makes it a promising tool for financial institutions. This work explores the theoretical concepts behind Quantum Annealing as well

as the formalization of the proposed Currency Arbitrage problem into the Quadratic Unconstrained Binary Optimization (QUBO) model [8], [9], which is the mathematical model to follow when the problem has to be approached with Quantum Annealing. The algorithm based on Quantum Annealing used in this paper is the one implemented by company D-Wave Systems. The performances of D-Wave’s algorithms based on Quantum Annealing are compared to the ones of classical methods, both from the point of view of being able to reach the optimum as well as execution times. The paper is organized as follows. Section II provides an introduction to the formulation of the Currency Arbitrage problem and how it can be expressed to be solved with Quantum Annealing. Section III includes the comparisons between the classical algorithm and Quantum Annealing. The analysis of the Advantage2 prototype2.6 QPU developed by D-Wave is carried out in Section IV. Conclusions are drawn in Section V.

II. PROBLEM FORMULATION

The optimization problem we are trying to solve is the Currency Arbitrage problem. We want to capture trading loops of currencies that would allow us to make a profit. The goal is to start from a given currency, convert the money into other currencies, then close the loop by returning to the same currency we started with: if at the end of the loop we have more money than the quantity we started with, we made a profit and we can denote the loop as a “profitable loop” (see Fig. 1).

A. QUBO Formulation

A Quadratic Unconstrained Binary Optimization (QUBO) problem is defined using an upper-diagonal matrix Q , which is an $N \times N$ upper-triangular matrix of real weights, and x , a vector of binary variables of size N . The QUBO problem can be expressed concisely as follows:

$$\min_{x \in \{0,1\}^N} x^T Q x \quad (1)$$

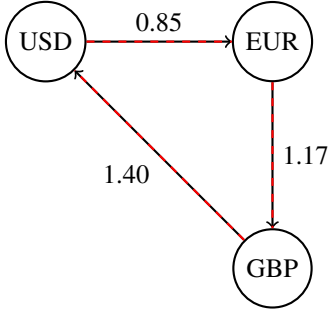


Fig. 1: Currency Arbitrage Loop: The exchange rates allow for a profitable arbitrage opportunity by cycling through USD \rightarrow EUR \rightarrow GBP \rightarrow USD.

More in detail, we are trying to minimize the following function:

$$f(x) = \sum_i Q_{i,i}x_i + \sum_{i < j} Q_{i,j}x_i x_j \quad (2)$$

where the diagonal terms $Q_{i,i}$ are the linear coefficients and the nonzero off-diagonal terms $Q_{i,j}$ are the quadratic coefficients. To align our problem to the QUBO model formulation we formulate the problem with a set of variables $x_{curr,pos}$, where $curr \in \{1, \dots, N\}$ and $pos \in \{1, \dots, K\}$. K corresponds to the **loop length**, i.e. the number of currencies we want to have in the loop, while N is the **total number of currencies**, i.e. how many currencies we consider when trying to find profitable loops of length K . To make the optimization problem non-trivial, we have to consider $K < N$.

$$x_{curr,pos} = \begin{cases} 1 & \text{if currency } curr \text{ is present at position } pos \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

A loop is profitable if it brings a profit. This is true if its profitability factor is larger than 1. The profitability factor is computed as the product of all the conversion rates of the currencies in the loop. We can denote the profitability factor P of a loop L as $P(L)$ and compute it as follows:

$$P(L) = \prod_{k=1}^{K-1} ExchangeRates(x_{ik}, x_{i(k+1)}) \quad (4)$$

where k is an index that corresponds to the position in the loop (e.g., in the Loop $[USD, EUR, GBP, USD]$ the USD currency is present in position $k = 0$ and $k = 3$, EUR in $k = 1$, GBP in $k = 2$). Using $P(L)$ expressed as a multiplication of several terms would raise the complexity of the problem, making it more than quadratic in complexity: this issue is solved by using the logarithm operator, which allows us to express the product as a sum by considering the logarithmic value of the exchange rate between any two currencies. We define it as:

$$\logRate_{ij} = -\log(ExchangeRate_{k,k+1}) \quad (5)$$

Therefore, $P(L)$ is substituted with computing the sum of the \logRate_{ij} terms for each consecutive couple of currencies

k and $k + 1$ in the loop L . This procedure reduces the complexity, allowing us to use a QUBO model to build the Hamiltonian that expresses the objectives and constraints of the problem. We also consider the inverse of the logarithm, i.e. the negative value of each \logRate_{ij} term, to convert this optimization problem into a minimization one.

B. Hamiltonian

The Hamiltonian is the operator that represents the total energy of a quantum system. The system's evolution is governed by the Hamiltonian, which is specifically designed to guide the system toward the optimal solution of the optimization problem. The ground state (i.e., the lowest energy state) of the final Hamiltonian corresponds to the optimal solution. To build the Hamiltonian for the Currency Arbitrage problem we want to solve, the objective(s) and the constraints have to be outlined: we can map each objective and constraint into their own Hamiltonian.

Objectives:

- 1) \mathcal{H}_A : the sum of the \logRate_{ij} terms should be minimized in order to obtain the most profitable loop.
- 2) \mathcal{H}_B : a currency appearing in consecutive positions in the loop should be rewarded, to facilitate shorter loops.

Constraints:

- \mathcal{H}_C : a position in the loop cannot contain more than one currency.
- \mathcal{H}_D : the loop must begin and end with the same currency.
- \mathcal{H}_E : no position of the loop must be left empty.

These Hamiltonians are formulated in the following way:

$$\mathcal{H}_A = A \sum_{i=1}^N \sum_{j=1}^{i-1} \logRate_{ij} \sum_{k=1}^{K-1} x_{ik} x_{j(k+1)} \quad (6)$$

$$\mathcal{H}_B = B \sum_{k=1}^K \sum_{i=2}^N x_{ik} \sum_{j=1}^{i-1} x_{jk} \quad (7)$$

$$\mathcal{H}_C = C \sum_{i=1}^N (1 - (x_{i1} - x_{iK})^2) \quad (8)$$

$$\mathcal{H}_D = D \sum_{i=1}^N \sum_{k=1}^{K-1} x_{ik} x_{i(k+1)} \quad (9)$$

$$\mathcal{H}_E = E \sum_{k=1}^K \sum_{i=1}^N x_{ik} \quad (10)$$

Where A, B, C, D, E are the weights of the Hamiltonians. Assigning an appropriate value to each of the weights is necessary to obtain satisfying results: e.g., the larger the value given to weight A of \mathcal{H}_A , the more we are pushing the algorithms to solve the problem in search of more profitable loops, but this could lead to solutions that are not feasible by violating one or more of the constraints. This means that there is a trade-off between solving the problems for more profitable loops and obtaining feasible solutions that respect the constraints imposed.

	x_{11}	x_{12}	x_{13}	x_{21}	x_{22}	x_{23}	x_{31}	x_{32}	x_{33}
x_{11}	E	A+D	C	B	A		B	A	
x_{12}		E	A+D	A	B	A	A	B	A
x_{13}			E	A	B		A		B
x_{21}				E	A+D	C	B	A	
x_{22}					E	A+D	A	B	A
x_{23}						E		A	B
x_{31}							E	A+D	C
x_{32}								E	A+D
x_{33}									E

Component A
 Component B
 Component C
 Component A and D
 Component E

Fig. 2: QUBO Matrix.

Therefore the total Hamiltonian that includes the objectives and constraints of our problem is obtained as follows:

$$\mathcal{H} = \mathcal{H}_A + \mathcal{H}_B + \mathcal{H}_C + \mathcal{H}_D + \mathcal{H}_E \quad (11)$$

We can now define the QUBO matrix, which is a square matrix that represents a QUBO problem according to the formulation reported in Eq. (2). In our case, the coefficients correspond to the weights of the Hamiltonian and \logRate_{ij} ; the binary variables correspond to the ones defined in Eq. (3). For the Hamiltonian \mathcal{H}_E , the interactions are only between the same variables, which means that the weights will represent the diagonal elements of the QUBO matrix. In all the other Hamiltonians the interactions are between different variables, so each of their coefficients are off-diagonal elements. Since the problem regards exchange rates, the matrix is a triangular matrix, therefore we can analyze only the upper or lower triangular part of it. Graphically, we consider the upper part. Let us take as an example a Currency Arbitrage problem with three currencies, the QUBO matrix with all the coefficients included is reported in Fig. 2:

- \mathcal{H}_A : the coefficient given by $A \cdot \logRate_{ij}$ contributes to the Hamiltonian only if currency i is in position k in the loop and currency j is in position $k+1$, therefore it fills the cells where the currency j is one position ahead of the currency i .
- \mathcal{H}_B : the coefficient given by B contributes to the Hamiltonian only if currency j is not the same currency as i for the same position k .
- \mathcal{H}_C : the coefficient given by C contributes to the Hamiltonian only if currency i is the same as currency j in position 1 and in the last position of the loop.
- \mathcal{H}_D : the coefficient given by D contributes to the Hamiltonian only if currency i is the same as currency j in position k and in position $k+1$.
- \mathcal{H}_E : the coefficient given by E represents the diagonal elements as mentioned before.

III. COMPARISON BETWEEN CLASSICAL SOLVERS AND QUANTUM ANNEALING

An interesting behavior examined in this work was how quickly the algorithms used were able to find the optimal solution: in Fig. 3 we reported the total execution times of the algorithms, more specifically how much time it took each algorithm to compute all the reads they were set up for (through the *num_reads* parameter). Therefore, these data highlight only

the speed of each algorithm to execute a given number of reads. The goal of this work though is to find optimal solutions for the Currency Arbitrage problems as quickly as possible, so it would be more useful to gather insights on how many reads each algorithm actually need to find the optimal value: for instance, if the DWaveSampler algorithm was set up to execute a total of 1,000 reads, but it was observed that it finds the optimum value with fewer than 50 reads for the majority of the trials, then the speed performance of the DwaveSampler algorithm should be measured according to that 50 reads, i.e. the actual reads it takes the algorithm to get to the optimal value. These line of process involves both the classical algorithms and the DwaveSampler algorithm based on Quantum Annealing. The classical algorithms taken into account for this comparison are *SimulatedAnnealingSampler* and *TabuSampler*, with the latter being based on Tabu Search, a metaheuristic algorithm that solves optimization problems by exploring the solution space while avoiding cycles and revisiting previously explored solutions by making use of a memory structure called *tabu list*. For Quantum Annealing we used *D-WaveSampler*, a sampler that uses a D-Wave quantum computer to sample from the QUBO formulation of our problem (which is more specifically a Binary Quadratic Model), leveraging on the Quantum Annealing algorithm to find the optimal or near-optimal solutions [10].

For this purpose, tests were run to check the number of reads that were necessary to each algorithm to reach the optimal solution for the first time. We used D-Wave's *ExactSolver*, a sampler that performs brute-force enumeration over all possible states of a Binary Quadratic Model, to retrieve the optimal solution for the configuration of currencies considered so that it could be used as the ground truth for the other algorithms; while running the algorithms it is not possible to stop their execution when the optimal solution is reached, because there is no way to control the flow of executions of the algorithms until they have finished their number of reads, so the workaround method used in this work was to save the samples obtained by the algorithms in the exact order they were found and then iterate all these results. For each algorithm, once the sample representing the optimal solution was found, the index associated with that sample was retrieved. At the end of this process we were able to obtain the first read of each algorithm when the optimal value was found.

The results are reported in Fig. 4, where 'Run Number' on the x-axis refers to the corresponding batch of 500 reads. From that, it can be inferred that Simulated Annealing is on average the algorithm that requires the higher number of reads to reach the optimum for the first time; D-WaveSampler takes fewer reads to get the optimum solution; TabuSampler is the best in this sense since it requires just one read; this can be due to the fact that if the problem size is small or the structure of the solution space is such that TabuSampler's heuristic rules can efficiently guide it to the optimal solution, a single read can identify the optimal answer.

From Fig. 3 it can be observed that Simulated Annealing is also the slowest between the algorithms, which confirms

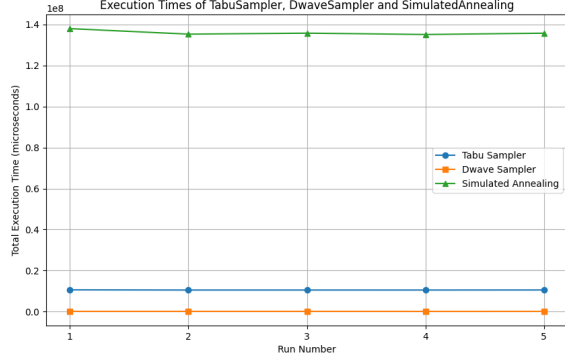


Fig. 3: Total execution times

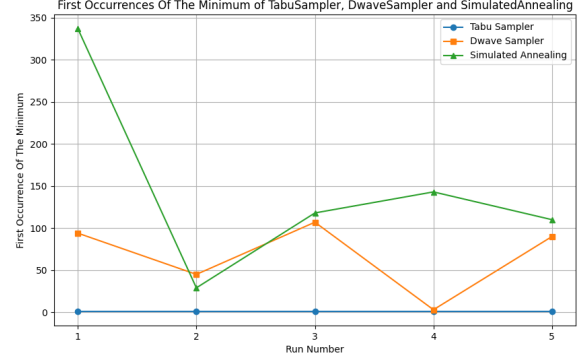


Fig. 4: First reads needed to find the optimum

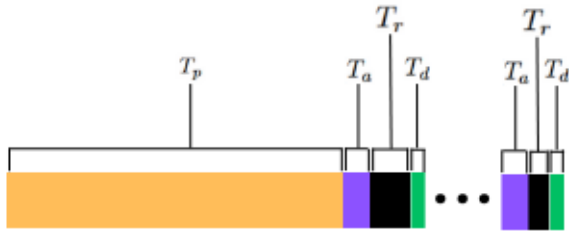


Fig. 5: Components of the QPU access time.

that D-WaveSampler is the fastest of the three in the total execution time considering all the reads, but as seen in Fig. 4 it needs more reads than TabuSampler. Therefore, it proved interesting to compute an estimate on the actual time needed by TabuSampler and D-WaveSampler to reach the read corresponding to the optimum solution without considering Simulated Annealing in the process since it would come out as the slower of the three apriori. Since, as previously mentioned, there is no way of retrieving information during the execution of the algorithms, estimations had to be made. TabuSampler reached the optimum in the first read all the times in this configuration and it took around 24,000 microseconds with one read. For D-WaveSampler, the way its execution time is computed required a more detailed study of the time components: the execution time reported in the figures refers to the *qpu_access_time* parameter. The time provided by this parameter is actually composed of multiple entities, all of them being reported in the D-Wave Documentation which also provides the following image (more details can be found in [11]) to display them visually: Overall, the *qpu_access_time* can be computed as follows:

$$T_{qpu} = T_p + \Delta + T_s \quad (12)$$

T_p (programming time) is a one-time initialization step to program the QPU. T_s represents the time for all the samples. Δ is denoted as *qpu_access_overhead_time* and it is an initialization time spent in low-level operations, roughly 10-

TABLE I: Comparison of QPU metrics for different *num_reads* values.

Metric Name	num_reads			
	1	10	100	500
qpu_access_time	15900	17133	34145	91141
qpu_sampling_time	117	1370	18384	75380
qpu_programming_time	15782	15762	15761	15761
qpu_readout_time_per_sample	47	66	113	80
qpu_delay_time_per_sample	20	20	20	20

20 ms for Advantage systems. In the D-Wave documentation it is stated that Δ is not included in the *qpu_access_time* field. Since Δ is still a time component that has a weight in the total execution time, we analyzed the results with it included: if we consider Δ to be always 20ms (the worst case scenario according to the documentation) and add it to the total execution times of D-WaveSampler, the relative results would actually be the same. Indeed, even if we consider the worst time of D-WaveSampler between all the ones obtained during the trials, which was 107,482 μ s, and add to it the 20ms given by Δ , we obtain 127,482 μ s which is still almost two orders of magnitude smaller than the total execution times of TabuSampler (see Fig. 3). T_s can be estimated in the following way: the time for a single sample is given by the sum of the *anneal time* T_a (the parameter set by the user when executing D-WaveSampler), *readout time* T_r (time required to read the sample from the QPU) and *thermalization time* T_d (time required by the QPU to regain its initial temperature). Therefore, if we multiply the time for a single sample with the *total number of reads* R , we obtain T_s :

$$T_s = R \cdot (T_a + T_r + T_d) \quad (13)$$

From how T_{qpu} is computed it can be deduced that the only component on which the developer has control is T_a . It also can be inferred that T_p can account for most of the total time of D-WaveSampler if we use very few reads.

Overall, the behavior of the time components of D-WaveSampler can be observed directly by accessing the *info['timing']* field of the SampleSet object returned by the

execution of D-WaveSampler.

The execution of D-WaveSampler considering *anneal_time* = 50 for different *num_reads* values brought the results reported in Table I for the mentioned time components. From that, it can be observed that *qpu_delay_time_per_sample* has a constant value independently from the *num_read* value; the same can be said for *qpu_programming_time* with negligible variations; moreover, even running D-WaveSampler on different days, i.e., with different rates between the currencies, does not affect this parameter values, so it could be considered constant from what it was observed in this work; *qpu_readout_time_per_sample* instead does not follow a pattern in its values, it has a non-deterministic behavior.

Overall, it can be inferred that *Simulated Annealing* is the slowest both in total time and in the time needed to reach the optimum for the first time; *TabuSampler* reaches the optimum in the first read in the tests made and takes around 24,000 μ s if it does only one read; D-WaveSampler is the one that requires the shortest amount of time to perform a single read per se, but it requires few reads to reach the optimal value while also having to deal with overheads like *qpu_programming_time*. *TabuSampler* is therefore, at the moment this work was carried out, the fastest in reaching the optimum, but D-WaveSampler is promising and could surpass it if the future hardware will bring lower overhead times. For this purpose, a comparison with a new QPU prototype that was being developed by D-Wave is explored in Sec. IV of this paper.

IV. ADVANTAGE2 PROTOTYPES

Analyses on the different D-Wave QPUs performances have already being carried out in other studies [12]. But currently D-Wave is developing a new QPU called Advantage2. They are creating prototypes of this new version and they update these prototypes as time passes. For instance, during August and September of 2024, the *Advantage2_prototype2.4* was being developed and made available to users. During the tests made in this work, *Advantage2_prototype2.4* was used to see if it allowed to have efficient results as with *Advantage_system4.1*, but with better times. The results obtained were promising and D-Wave was going to put out better prototypes as time passed on, so we considered this an aspect that we had to follow up with other tests. In October and November 2024 *Advantage2_prototype2.5* was created, then in November 2024 and December 2024 *Advantage2_prototype2.6* was created. To check if the results kept being promising, we performed tests on *Advantage2_prototype2.6*: in particular, D-WaveSampler was tried both with *Advantage_system4.1* and with *Advantage2_prototype2.6*, the solutions were then compared as well as their execution times. The results of these tests are reported in Fig. 6,7,8,9,10,11,12,13.

The parameter used to measure the execution times is the *qpu_access_time* parameter, which represents the actual time spent on the QPU for solving the problem, as already mentioned in Sec. III of this paper. Time is noted in microseconds as confirmed in the D-Wave documentation at the voice 'SAPI

Timing Fields'. These tests were made considering $N = 5$ and $K = 4$. The *qpu_access_time* was measured considering four different values for the *num_reads* parameter: 50; 500; 2000; 4000. For each of these values, two batches of ten executions each were carried out: i.e. if we take the *num_reads* = 50 case, D-WaveSampler was ran ten different times for the first batch and then another ten times for the second batch; this process was repeated also for *num_reads* = 500, *num_reads* = 2000 and *num_reads* = 4000. The goal was to examine if the difference in time executions between the two QPUs remained constant or if the gap actually grew when considering a growing amount of *num_reads*. The arithmetic means of the executions times of all the trials in Fig. 6,7,8,9,10,11,12,13 are reported in Table II.

From that it can be observed that for *num_reads* = 50 *Advantage*

TABLE II: Comparison of Advantage System Metrics Across Two Batches

<i>num_reads</i>	Batch 1		Batch 2	
	System4.1	Prototype2.6	System4.1	Prototype2.6
50	23917	25856	23971	25653
500	93725	85737	92374	85617
2000	345590	277681	348638	266169
4000	723505	530786	647858	498289

system4.1 performs better in execution time, for both the batches. This does not hold true when *num_reads* = 500, where *Advantage2_prototype2.6* already starts performing faster. It then can be seen that the difference in times grows wider when *num_reads* = 2000 and even more when *num_reads* = 4000. Overall, *Advantage2_prototype2.6* already seems to perform faster than the system used for this work, therefore when the actual *Advantage2* will be complete and released far better results are to be expected.

V. CONCLUSION

In this paper we first provide an original QUBO formulation for the currency arbitrage problem and then we explored the potential of Quantum Annealing in solving it, with an emphasis on evaluating its effectiveness with respect to classical counterparts (Simulated Annealing and Tabu Search). The APIs from D-Wave were used to access the Quantum Annealing based algorithm D-WaveSampler; we compared the obtained performance with the mentioned classical algorithms, both from the point of view of total execution times and from point of view of reaching the optimum for the first time. The results of tests made with a more recent prototype of D-Wave's QPU (Advantage 2) were also reported, highlighting how D-Wave's Quantum Annealing is becoming faster and faster, with the promise of outperforming the classical algorithms. Overall, D-wave's Quantum Annealer proved to be a valid alternative at solving our optimization problem, and it has the potential to become better than classical algorithms in its future, improved, versions (more qubits, better topology, increased coherence time, etc.).

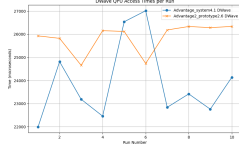


Fig. 6: First run with 50 reads

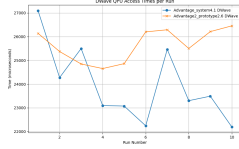


Fig. 7: Second run with 50 reads

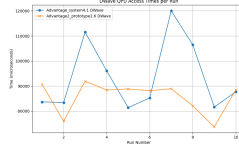


Fig. 8: First run with 500 reads

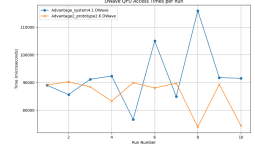


Fig. 9: Second run with 500 reads

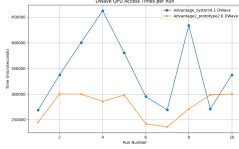


Fig. 10: First run with 2000 reads

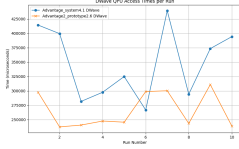


Fig. 11: Second run with 2000 reads

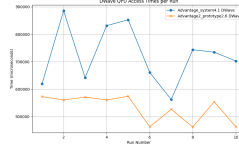


Fig. 12: First run with 4000 reads

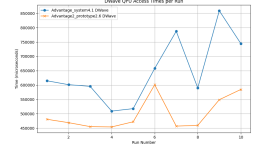


Fig. 13: Second run with 4000 reads

ACKNOWLEDGMENT

We deeply thank Sebastian Feld for his help in the creation of the QUBO formulation of this problem. This study was carried out within the ICSC National Centre on HPC, Big Data and Quantum Computing - SPOKE 10 (Quantum Computing) and received funding from the European Union Next-GenerationEU - National Recovery and Resilience Plan (NRRP) – MISSION 4 COMPONENT 2, INVESTMENT N. 1.4 – CUP N. I53C22000690001. This work reflects solely the opinions and views of the authors, and neither the European Union nor the European Commission can be held responsible for them.

REFERENCES

- [1] D-Wave Systems, “Dwavesampler,” 2024. [Online]. Available: https://docs.ocean.dwavesys.com/en/latest/quantum_research/annealing.html
- [2] F. A. Quinton, P. A. S. Myhr, M. Barani, P. Crespo del Granado, and H. Zhang, “Quantum annealing applications, challenges and limitations for optimisation problems compared to classical solvers,” *Scientific Reports*, vol. 15, no. 1, p. 12733, Apr 2025. [Online]. Available: <https://doi.org/10.1038/s41598-025-96220-2>
- [3] T. Kadowaki and H. Nishimori, “Quantum annealing in the transverse ising model,” *Phys. Rev. E*, vol. 58, pp. 5355–5363, Nov 1998. [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRevE.58.5355>
- [4] M. Vesely, “Finding the optimal currency composition of foreign exchange reserves with a quantum computer,” 2023. [Online]. Available: <https://arxiv.org/abs/2303.01909>
- [5] B. Tasseff, T. Albash, Z. Morrell, M. Vuffray, A. Y. Lokhov, S. Misra, and C. Coffrin, “On the emerging potential of quantum annealing hardware for combinatorial optimization,” *Journal of Heuristics*, vol. 30, no. 5, pp. 325–358, Dec 2024. [Online]. Available: <https://doi.org/10.1007/s10732-024-09530-5>
- [6] E. Aguilera, J. de Jong, F. Phillipson, S. Taamallah, and M. Vos, “Multi-objective portfolio optimization using a quantum annealer,” *Mathematics*, vol. 12, no. 9, 2024. [Online]. Available: <https://www.mdpi.com/2227-7390/12/9/1291>
- [7] J. Lang, S. Zielinski, and S. Feld, “Strategic portfolio optimization using simulated, digital, and quantum annealing,” *Applied Sciences*, vol. 12, no. 23, 2022. [Online]. Available: <https://www.mdpi.com/2076-3417/12/23/12288>
- [8] F. Glover, G. Kochenberger, and Y. Du, “A tutorial on formulating and using qubo models,” 2019. [Online]. Available: <https://arxiv.org/abs/1811.11538>
- [9] D-Wave Systems, “Dwavesampler,” 2024. [Online]. Available: https://docs.ocean.dwavesys.com/en/latest/quantum_research/qubo_ising.html

- [10] —, “Dwavesampler,” 2024. [Online]. Available: https://docs.ocean.dwavesys.com/en/latest/docs_system/reference/samplers.html
- [11] —, “Operation and timing,” 2024. [Online]. Available: https://docs.dwavesys.com/docs/latest/c_qpu_timing.html
- [12] D. Willsch, M. Willsch, C. D. Gonzalez Calaza, F. Jin, H. De Raedt, M. Svensson, and K. Michielsen, “Benchmarking advantage and d-wave 2000q quantum annealers with exact cover problems,” *Quantum Information Processing*, vol. 21, no. 4, p. 141, Apr 2022. [Online]. Available: <https://doi.org/10.1007/s11128-022-03476-y>