Benchmarking Quantum Computing for Combinatorial Optimization

1st Nathan Kittichaikoonkij
Faculty of Engineering
Chulalongkorn University
Bangkok, Thailand
6538047621@student.chula.ac.th

2nd Nutthapat Pongtanyavichai
Faculty of Engineering
Chulalongkorn University
Bangkok, Thailand
6532068721@student.chula.ac.th

3rd Poopha Suwananek
Faculty of Engineering
Chulalongkorn University
Bangkok, Thailand
6532141821@student.chula.ac.th

4th Prabhas Chongstitvatana
Faculty of Engineering
Chulalongkorn University
Bangkok, Thailand
Prabhas.C@chula.ac.th

5th Kamonluk Suksen*
Faculty of Engineering
Chulalongkorn University
Bangkok, Thailand
kamonluk@cp.eng.chula.ac.th

***Abstract*—Quantum computers can be much faster than any classical computers in solving a certain class of problems. One of the interesting problems is the combinatorial optimization problem, which is a challenge for classical computer systems. We encode these problems into the Quadratic Unconstrained Binary Optimization (QUBO) format and then solve them with quantum solvers. In this study, we compare the performance of D-Wave's quantum annealing system with classical solvers, namely, Gurobi and Fixstars. We demonstrate the current capabilities of the D-Wave system.**

***Index Terms*—quantum optimization, quantum annealing, combinatorial optimization.**

I. Introduction

Quantum computing has emerged as a transformative approach for addressing the challenges of combinatorial optimization, leveraging quantum mechanical phenomena to potentially outperform classical methods in specific cases. Quantum annealing and gate-based quantum algorithms provide novel paradigms for solving optimization problems, specifically combinatorial optimization problems.

Combinatorial optimization lies at the heart of real-world applications, including logistics, scheduling, and network design, with their complexity growing exponentially as the problem size increases. While classical solvers such as Gurobi and Fixstars excel at problems like the 3-Satisfiability (3SAT), the Quadratic Assignment Problem (QAP), and the Traveling Salesman Problem (TSP), using techniques like linear programming and heuristics, their scalability is limited by computational resources and runtime. These limitations highlight the potential of quantum computing as a complementary approach.

In the current Noisy Intermediate-Scale Quantum (NISQ) era, where quantum systems are characterized by limited coherence time, noise, and susceptibility to errors, quantum annealing (QA) has stood out as a currently practical approach for solving combinatorial optimization problems. Unlike the gate-based quantum computing model, QA demonstrates resilience by leveraging the quantum tunneling effect to traverse energy barriers, thus avoiding local extrema. This feature makes QA particularly suitable for the NISQ era [1].

QA is rooted in simulated annealing but employs quantum phenomena for optimization. It is closely linked to the Quadratic Unconstrained Binary Optimization (QUBO) formalism, which has been a standard representation in combinatorial optimization for decades. QUBO serves as the input language for quantum machines such as D-Wave Systems and other quantum-inspired technologies such as Fujitsu's Digital Annealer and NTT's Coherent Ising Machine [1].

This study focuses on benchmarking quantum annealing's capabilities for combinatorial optimization problems, examining its strengths and limitations. To ensure fairness, we kept all default hyperparameters of the quantum annealer unchanged—such as `annealing_time`, and `chain_strength`—modifying only the `num_reads` parameter, which controls the number of annealing samples returned by the hardware. Since quantum annealing is inherently probabilistic, increasing the number of reads improves the likelihood of obtaining the optimal solutions. By comparing the performance of D-Wave Advantage System 6.4 with classical solvers, we aim to provide insights into the practical potential of quantum computing in optimization.

II. Preliminaries

A. Quadratic Unconstrained Binary Optimization (QUBO)

QUBO is a mathematical formulation used to represent optimization problems. It encodes the objective function as a quadratic polynomial in binary variables. The goal is to find a binary vector that minimizes the quadratic cost function [2]. QUBO problems can be expressed as:

$$E(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x}, \tag{1}$$

*Corresponding author

where Q is a symmetric matrix of weights, and $\mathbf{x}$ is a binary vector $\{0,1\}$. QUBO is widely used due to its flexibility in representing problems such as 3SAT, QAP, and TSP.

B. The Ising Model

The Ising model is a widely used representation of optimization problems in quantum annealing, formulated to find the ground state of the Hamiltonian:

$$H = \sum_i h_i \sigma_i^z + \sum_{i<j} J_{ij} \sigma_i^z \sigma_j^z, \tag{2}$$

where h_i represents local fields, J_{ij} denotes spin coupling, and σ_i^z are Pauli-Z operators. The Ising model and QUBO are equivalent and can be interconverted using the linear transformation [3]:

$$x_i = \frac{1+s_i}{2}, \tag{3}$$

where x_i is a binary variable and s_i is a spin variable $\{-1,1\}$.

III. Methods for Solving QUBO Problems

A. Classical Methods

Classical optimization techniques often use Mixed Integer Programming (MIP) solvers to address QUBO problems. These solvers handle quadratic problems involving integer and real variables, including QUBO formulations. For example, the Gurobi Optimizer is a widely used MIP solver that can process QUBO problems by defining the objective function in quadratic form and applying standard optimization algorithms to find the optimal solution [4].

B. Quantum Annealing with D-Wave

The D-Wave quantum processing unit employs quantum annealing to find the minimum of an energy landscape defined by the biases and couplings applied to its qubits in the form of a problem Hamiltonian. This process involves evolving qubits, initially in a superposition of states, toward the ground state of a problem-specific Hamiltonian [5]. To solve a problem through sampling, the objective function is formulated such that finding its minimum corresponds to solving the problem. The objective is defined as the problem Hamiltonian by specifying the linear and quadratic coefficients of a binary quadratic model (BQM), which maps these values to the qubits and couplers of the quantum processing unit (QPU). For the QPU, objective functions can be represented using either the Ising Model or QUBO, both of which are binary quadratic models and converting from one formulation to the other is trivial.

C. Gate-Based Quantum Computing

Gate-based quantum computing uses quantum circuits for computation, with the Quantum Approximate Optimization Algorithm (QAOA) commonly applied to QUBO problems. QAOA uses parameterized quantum gates to prepare a quantum state that encodes the solution, alternating between problem-specific and mixing Hamiltonians to find the optimal solution through quantum interference. This approach is implemented with quantum programming frameworks such as Qiskit, enabling execution on various quantum processors. QAOA offers a flexible framework for approximating solutions in the NISQ era [6].

IV. Methodology

A. Solvers

Brute Force: Our approach of Brute Force is implemented with Python. It tries all possible answers until it hits the time limit and stops, returning the best result so far. It runs single-threaded on a MacBook Pro 2023 (M2 Max).

Gurobi Optimizer: A classical solver employing advanced mathematical programming techniques, Gurobi primarily uses a Branch-and-Bound framework, enhanced with Cutting Planes, Heuristics, and Presolve Reductions. It dynamically selects between the Simplex and Barrier (Interior Point) methods for solving continuous relaxations based on the problem structure. Optimized for multi-threaded performance, we used Gurobi 11.0.3 under an Academic License on a MacBook Pro 2023 (M2 Max, 12-core CPU).

Fixstars Amplify QUBO Solver: A quantum-inspired GPU-accelerated solver optimized for QUBO problems. As Fixstars Amplify offers only its GPU-accelerated Annealing Engine for QUBO, our experiments employed its Simulated Annealing (SA) algorithm—emulating thermal annealing to escape local minima and approach near-optimal solutions [7]. The experiments were conducted on the cloud platform via their API under the Basic Plan (Free Plan for Evaluation and Testing). The experiment was done on the first half of January 2025.

D-Wave Quantum Annealing: A quantum annealing system for optimization tasks. Experiments were conducted on the D-Wave Leap platform with a Developer Plan (Free Trial Access) during the first half of January 2025. We used a solver named Advantage System 6.4 with default parameters, except for `num_reads` set to `1000`.

B. Mathematical Formulation of QUBO Models

The QUBO formulations for these three benchmark problems are denoted by $H_{\text{Problem Name}}$. For QAP and TSP, the QUBO model is composed of two parts: The objective function H_O and the penalty function g which enforces constraints along with the penalty weight λ [8]. The model formulation for three problems is as follows:

- **3SAT**

$$\begin{aligned} H_{\text{3SAT}} = -\sum_{i=1}^{m} \Big(&(1+w_i)(y_{i1}+y_{i2}+y_{i3}) \\ &- y_{i1}y_{i2} - y_{i1}y_{i3} - y_{i2}y_{i3} - 2w_i \Big) + K \end{aligned} \tag{4}$$

  where y_{ij} are binary variables representing literals (x_i), w_i are the binary variables associated with each clause, and K is an offset for normalization and is the minimum number of clauses which are satisfied no matter the binary literals. [9]

- **QAP**

$$H_O = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} f_{ik} d_{jl} x_{ij} x_{kl} \tag{5}$$

$$g = \sum_{i=1}^{n}\left(\sum_{j=1}^{n} x_{ij} - 1\right)^2 + \sum_{j=1}^{n}\left(\sum_{i=1}^{n} x_{ij} - 1\right)^2 \tag{6}$$

$$H_{\text{QAP}} = H_O + \lambda g \tag{7}$$

where f_{ij} represents the flow (interaction) between facilities i and j, d_{ij} is the distance between locations i and j, x_{ij} is a binary variable indicating if facility i is assigned to location j, and n is the number of facilities and number of locations.

- **TSP**

$$\begin{aligned} H_O = &\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{p=1}^{n-1} d_{ij} x_{i,p} x_{j,(p+1)} \\ &+ \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij} x_{i,n} x_{j,1} \end{aligned} \tag{8}$$

$$g = \sum_{i=1}^{n}\left(\sum_{p=1}^{n} x_{i,p} - 1\right)^2 + \sum_{p=1}^{n}\left(\sum_{i=1}^{n} x_{i,p} - 1\right)^2 \tag{9}$$

$$H_{\text{TSP}} = H_O + \lambda g \tag{10}$$

where $x_{i,p}$ is Binary Variable indicating whether city i is being visited at time p, d_{ij} is distance between cities i and j, and n is the total number of cities.

C. Datasets

The guidelines for selecting benchmark datasets are as follows:

- **3SAT:** The benchmark test sets are generated and verified to be satisfiable with PySAT and are from SATLIB All Satisfiable Uniform Random-3-SAT [10], where one problem is randomly selected from each set of 100 problems. The number of variables and clauses for each set are as follows: 5 variables with 15 clauses, 10 variables with 30 clauses, 15 variables with 55 clauses (which are generated), 20 variables with 91 clauses, 50 variables with 218 clauses, 75 variables with 325 clauses, 100 variables with 430 clauses, and 125 variables with 538 clauses (which are from SATLIB). These correspond to the sizes of the binary vectors of the problems: 20, 40, 70, 111, 268, 400, 530, and 663, respectively.
- **QAP and TSP:** The benchmarks used fixed seeds to generate random weight matrices with sizes ranging from 4 to 12 $nodes$, where $nodes$ represent the number of factories and locations in QAP or the number of cities in TSP. The matrix values were uniformly distributed between 1 and 9 inclusive. Both problems required constraints, implemented using a penalty value of 10^6, which was found to be effective without affecting the computation time. Both of these problems are encoded such that the size of the binary vectors (problem size) is $nodes^2$.

D. Evaluation Metrics

We used three metrics to evaluate the solvers' performance:

1) Success Rate: The success rate is calculated as the number of times the solver returns a feasible solution over the total number of runs. If all samples in a run fail to meet the constraints, that run is considered to have no feasible solution. For problems like 3SAT, which do not have additional constraints, the success rate is either 1.0 or 0.0, with 0.0 occurring when the problem size becomes too large for the solver.

2) Accuracy: Percentage of the optimal solution given.

- **3SAT:** Accuracy is verified by checking the correctness through boolean algebra.
- **QAP and TSP:** Accuracy is determined by matching the solution to Brute Force for the problem instance. Our experiment shows that for $nodes \leq 10$ for QAP and $nodes \leq 11$ for TSP. The Brute Force method, having not hit its time limit, explored all possible solutions, ensuring the optimal result. For $nodes = 11, 12$ we will assume Fixstars provided an optimal solution.

3) Computation Time: Measure the time it takes for the solver to first find the best solution. This does not include the time it takes to formulate the problem, upload, and queue. Some solvers like Fixstars will continue running despite finding an optimal solution, therefore, we will only use the time it first found its best solution.

V. Benchmarking QUBO Solvers

A. Experimental Results

Results from solving 3SAT, QAP, and TSP instances revealed:

1) 3SAT:

a) Success Rate: Across all test sets, the classical solvers (Gurobi and Fixstars) successfully find all the solutions. In contrast, D-Wave's Advantage System 6.4 fails to solve problems with a vector size of 663 as the problem size became too large for D-Wave's minor embedding.

b) Accuracy Results: As shown in Fig. 1, the results' accuracy indicates that the D-Wave solver can only solve problems up to a size of 663, primarily due to minor embedding errors caused by the problem's large size. Comparatively, D-Wave's accuracy is the lowest among the three solvers, with Advantage System 6.4, but still remains above 0.95. For the Gurobi solver, its accuracy depends heavily on the timeout parameter as it gradually declines as the problem size gets larger. Finally, Fixstars demonstrates the highest accuracy, remaining completely accurate except for problem sizes of 40 and 663, where it maintains the highest overall accuracy.

c) Computation Time: As shown in Fig. 2, for problem sizes ranging from 111 to 530, the execution time of D-Wave remains relatively low compared to that of Fixstars and Gurobi. Both solvers exhibit increasing execution times as the problem size grows, but D-Wave's execution time remains significantly lower compared to the Gurobi solver. Gurobi's execution time grows much faster than the other two solvers, quickly reaching its timeout limit, even for relatively small problem sizes. And in Fig. 3, Fixstars' and D-Wave's computation time grows linearly, and D-Wave's with the smaller slope demonstrates a slower increase in execution time.

2) QAP:

a) Success Rate: Classical Solvers which are Brute Force, Gurobi, and Fixstars, are able to find a feasible solution for all problems. However, for D-Wave, it struggles to find a feasible solution at $nodes = 6$ and no solution after $nodes = 7$, before reaching the hardware's size limit.

b) Accuracy Results: The accuracy results for solving QAP instances are shown in Fig. 4. It is shown that all classical solvers achieve optimal solutions for $nodes \leq 10$. D-Wave achieved 100% accuracy for $nodes = 4$ but the accuracy drops significantly as the problem size increases. At $nodes = 5$, accuracy drops to 70% and fails to find optimal solutions for $nodes \geq 6$.

c) Computation Time: In Fig. 5, we can see that Brute Force and Gurobi's computation time grows exponentially (or linearly on a log scale) as the number of $nodes$ increases.

Fig. 6 provides a clearer view of Fixstars' and D-Wave's computation time. They grow almost linearly as the number of $nodes$ increases. The graph also shows that Fixstars' growth is more than D-Wave, but this cannot be concluded as we lack data on D-Wave.

3) TSP:

a) Success Rate: Classical Solvers which are Brute Force, Gurobi, and Fixstars, are able to find a feasible solution for all problems. However, for D-Wave, it struggles to find a feasible solution at $nodes = 7$ and no solution after $nodes = 8$, before reaching the hardware's size limit.

b) Accuracy Results: The accuracy results for solving TSP instances are shown in Fig. 7. It is shown that all classical solvers achieve optimal solutions for $nodes \leq 12$. D-Wave achieved 100% accuracy for $nodes = 4$ and $nodes = 5$ but the accuracy drops significantly as the problem size increases. At $nodes = 6$, accuracy drops to 20% and fails to find optimal solutions for $nodes \geq 7$.

c) Computation Time: In Fig. 8, we can see that Brute Force and Gurobi's computation time grows exponentially (or linearly on a log scale) as the number of $nodes$ increases.

Fig. 9 provides a clearer view of Fixstars' and D-Wave's computation time. They grow almost linearly as the number of $nodes$ increases. As with QAP's case, we cannot compare the growth rate as we lack data on D-Wave.

B. Discussion

The experimental results highlight distinct behaviors and limitations of quantum and classical solvers when applied to all 3 problems. Below, we delve into the key observations and underlying reasons behind these phenomena.

1) Quantum Solver Scalability and Limitations: For dense combinatorial problems like QAP and TSP, the embedding process becomes increasingly resource-intensive as the number of nodes grows. Specifically, for QAP $nodes = 6$ and TSP $nodes = 7$. This inability stems from hardware constraints, specifically the quantum annealer's topology and the process of minor embedding [11].

In contrast, sparse combinatorial problems like 3SAT are more amenable to quantum annealing. The sparse nature of 3SAT instances allows for more efficient embedding into the quantum annealer's topology, enabling the system to handle larger problem sizes. For example, D-Wave's quantum annealer can effectively solve 3SAT instances with up to 530 variables, demonstrating its capability to manage larger sparse problems compared to dense ones.

2) Execution Time Trends: In contrast, classical solvers such as Gurobi and the Brute Force method exhibit exponential time growth due to the combinatorial nature of optimization problems. D-Wave demonstrates near-linear time growth, indicating more stable execution times as the problem size increases. This highlights a key advantage of quantum solvers: their ability to maintain consistent performance within the limitations of available hardware resources.

While this does not yet represent **quantum speed-up** in the strictest sense, it emphasizes the potential of quantum annealing for solving optimization problems efficiently.

VI. Conclusion

TABLE I
Summary of Results for 3SAT, QAP, and TSP

3SAT			QAP			TSP		
size	acc(%)	time (ms)	size	acc(%)	time(ms)	size	acc(%)	time(ms)
20	100.000	95.803	4	100	94.444	4	100	94.443
40	99.000	106.503	5	70	99.603	5	100	99.603
70	97.091	126.205	6	0	106.165	6	20	106.164
111	97.033	190.226				7	0	124.323
268	97.523	193.467						
400	95.661	211.567						
530	95.721	227.981						

Quantum computing holds promise for solving combinatorial optimization problems, with quantum annealing showing potential in this area. In our experiments, D-Wave demonstrated slower computation times compared to classical solvers such as Gurobi and Fixstars in most cases. However, the growth in computation time for D-Wave was minimal as problem size increased, while Gurobi exhibited noticeable growth rates.

This paper presents benchmarks that evaluate the performance of D-Wave's quantum annealer within a pure quantum computing framework, without the use of hybrid approaches. While extensive parameter tuning was not performed, we modified the `num_reads` parameter to ensure consistent

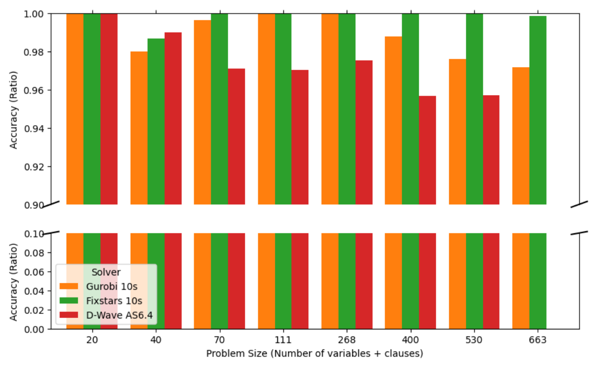

Fig. 1. 3SAT: Accuracy of all solvers with varying problem sizes

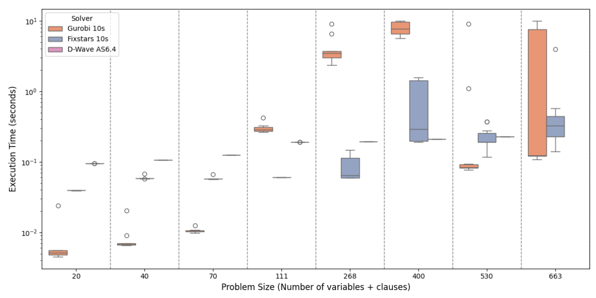

Fig. 2. 3SAT: Box and Whisker Plot of Execution Time for all Solvers

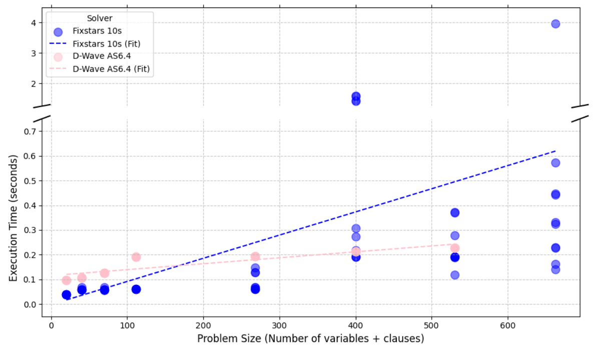

Fig. 3. 3SAT: Execution Time for Fixstars and D-Wave with Linear Regression Lines

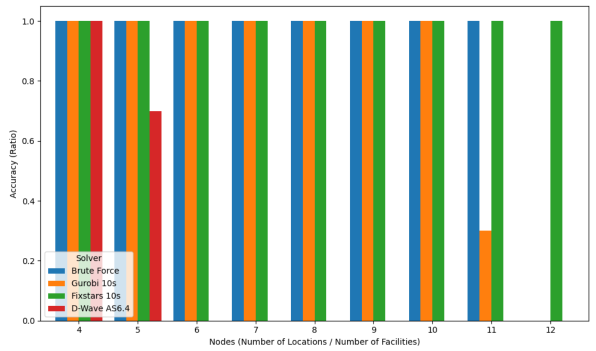

Fig. 4. QAP: Accuracy of all solvers with varying node counts

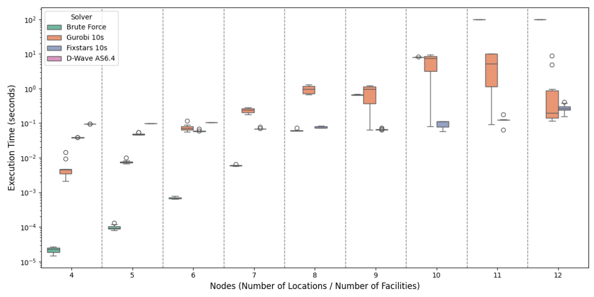

Fig. 5. QAP: Box and Whisker Plot of Execution Time for all Solvers

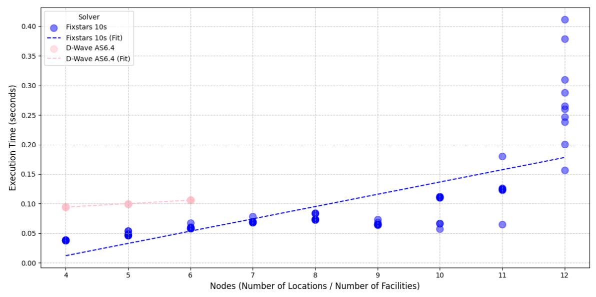

Fig. 6. QAP: Execution Time for Fixstars and D-Wave with Linear Regression Lines

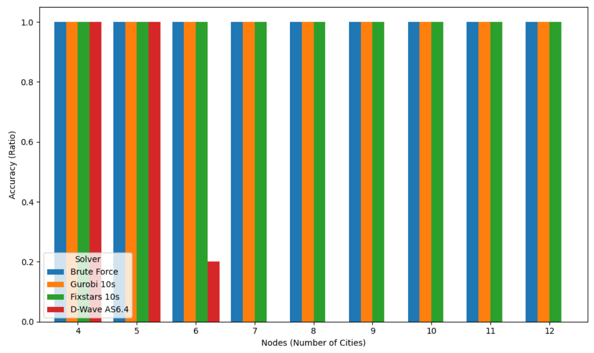

Fig. 7. TSP: Accuracy of all solvers with varying node counts

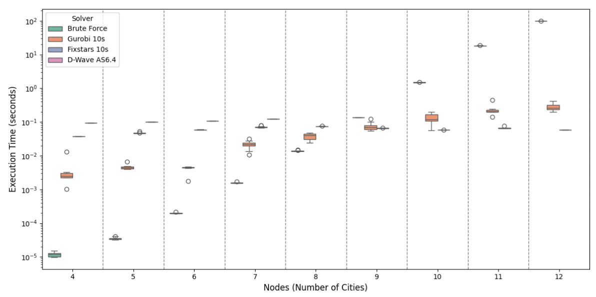

Fig. 8. TSP: Box and Whisker Plot of Execution Time for all Solvers

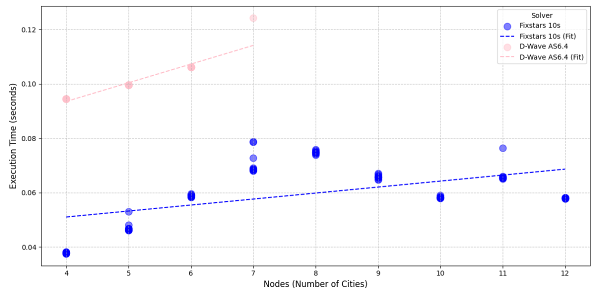

Fig. 9. TSP: Execution Time for Fixstars and D-Wave with Linear Regression Lines

sampling. Other parameters, such as `annealing_time` and `chain_strength`, were left at their default values. Further optimization is reserved for future work.

In this study by Villarrodri et al. [12], a comprehensive analysis of the sensitivity of the Hamiltonian-related parameters for D-Wave's quantum annealer was conducted. By evaluating over 200 parameters configurations—including all the parameters mentioned, they showed that fine-tuning these parameters significantly influences solution quality and energy distribution. For instance, `chain_strength` controls how strongly the qubits in a logical chain are coupled together; if the strength is too weak, the chains may break which would result in the failure of problem embedding [13]. On the other hand, too strong and the constraints of the intended problem will no longer represent the original problem. `annealing_time` determines how long the system searches for solutions, with longer runs increasing the likelihood of remaining in the ground state in accordance with the adiabatic theorem.

These results highlight quantum annealing's potential and limitations. Key challenges—improving embedding techniques, mitigating errors, and scaling hardware—must be addressed to realize its full potential. While classical solvers dominate large instances, quantum annealers show promise for specific applications and smaller problem sizes, warranting further research.

Future work will expand to include hybrid quantum-classical methods, particularly gate-based approaches like the Quantum Approximate Optimization Algorithm (QAOA). We plan to explore QAOA implementations across multiple platforms—including Qiskit, Cirq, CUDA-Q, and Q#—to assess their practicality for solving the same combinatorial optimization problems benchmarked in this study. Future benchmarks will focus on evaluating scalability, accuracy, and execution time, as well as parameter tuning strategies. This includes tuning key parameters such as `annealing_time` and `chain_strength` (for annealing-based and quantum-inspired solvers), which play a critical role in solution quality and performance.

Acknowledgment

This work was made possible by a collaboration between Chulalongkorn University and the National University of Singapore in the project "Quantum Computing and Machine Learning for Combinatorial Optimization" supported by the Singapore Ministry of Education Academic Research Fund (AcRF) Tier 1, grant MOE-T1-251RES2302. The authors thank Lim Chee Keen for helpful discussions. The first three authors contributed equally to this work.

References

[1] P. Codognet, D. Diaz, and S. Abreu, "Quantum and digital annealing for the quadratic assignment problem," in *2022 IEEE International Conference on Quantum Software (QSW)*, 2022, pp. 1–8.

[2] F. Glover, G. Kochenberger, and Y. Du, "A tutorial on formulating and using qubo models," 2019. [Online]. Available: https://arxiv.org/abs/1811.11538

[3] A. Mandal, A. Roy, S. Upadhyay, and H. Ushijima-Mwesigwa, "Compressed quadratization of higher order binary optimization problems," 2020. [Online]. Available: https://arxiv.org/abs/2001.00658

[4] Fixstars. (2024) Gurobi optimizer - fixstars amplify sdk documentation. Accessed: 2025-01-05. [Online]. Available: https://amplify.fixstars.com/en/docs/amplify/v1/clients/gurobi.html

[5] A. Rajak, S. Suzuki, A. Dutta, and B. K. Chakrabarti, "Quantum annealing: an overview," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 381, no. 2241, dec 2022. [Online]. Available: http://dx.doi.org/10.1098/rsta.2021.0417

[6] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," 2014. [Online]. Available: https://arxiv.org/abs/1411.4028

[7] Fixstars. (2024) Fixstars amplify annealing engine. Accessed: 2025-04-17. [Online]. Available: https://amplify.fixstars.com/en/docs/amplify/v1/clients/fixstars.html

[8] ——. (2022) Constraint - amplify documentation. Accessed: 2025-01-05. [Online]. Available: https://amplify.fixstars.com/en/docs/amplify/v0/constraint.html

[9] M. J. Dinneen, "Maximum 3-sat as qubo," 2016, lecture notes, CompSci 750, Semester 2, University of Auckland. [Online]. Available: https://canvas.auckland.ac.nz/courses/14782/files/574983/download

[10] T. U. of British Columbia. (2024) Satlib - benchmark problems. Accessed: 2024-10-15. [Online]. Available: https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html

[11] S. Jain, "Solving the traveling salesman problem on the d-wave quantum computer," *Frontiers in Physics*, vol. 9, 2021. [Online]. Available: https://www.frontiersin.org/journals/physics/articles/10.3389/fphy.2021.760783

[12] E. Villar-Rodriguez, E. Osaba, and I. Oregi, "Analyzing the behaviour of d'wave quantum annealer: fine-tuning parameterization and tests with restrictive hamiltonian formulations," 2022. [Online]. Available: https://arxiv.org/abs/2207.00253

[13] D-Wave. (2020) Programming the d-wave qpu: Parameters for beginners. Accessedd: 2025-04-17. [Online]. Available: https://www.dwavequantum.com/media/qvbjrzgg/guide-2.pdf