Supplementary Material

In this appendix, we provide additional information on our small proof-of-concept benchmark.

1 SUPPLEMENTARY DATA BENCHMARKING INSTANCES

For the maximum weighted independent set (MWIS) problem, we consider randomly generated graphs with different numbers of vertices (size k) and random vertex weights. We generate graphs with 5, 10, 15, 25, and 50 vertices and perform three tests for each size (”instances”). The adjacency matrix of the graph is built by generating a binary upper triangular matrix with discrete uniform random number generator. All entries on the main diagonal are set to 0 to delete loops. Vertex weights a_i are generated with a discrete uniform random generator with values ranging between 1 and $2k + 1$.

For the QUBO formulation of the MWIS problem given in eq. (3), we set a penalty factor for each instance. This is given by $2 \max_{v \in V} weight(v)$, two times the maximum vertex weight of the instance: if we add to the set two vertices v_1 and v_2 connected with an edge, the penalty $2 \max_{v \in V} weight(v)$ is greater than or equal to the contribution $weight(v_1) + weight(v_2)$ to the objective function. Thus, infeasible solutions are unlikely to occur. The complete set of instances can be found here: `https://gitlab.itwm.fraunhofer.de/halffmann/maximum-weighted-independent-set-instances`.

2 SUPPLEMENTARY TABLE BENCHMARKING RESULTS

size	Gurobi		D-Wave QPU			
(no. graph vertices)	Sol.	Time	Embed Time	Time	Sol.	Sol. Freq.
5	20	0.0000	0.0000	0.1198	20	757
	22	0.0000	0.0000	0.1161	22	10
	17	0.0004	0.0000	0.1268	17	89
10	42	0.0244	0.0000	0.1432	42	384
	50	0.0000	0.0000	0.1344	50	12
	69	0.0000	0.0000	0.1262	69	625
15	78	0.0000	0.0161	0.1292	78	43
	110	0.0191	0.0669	0.1517	110	351
	75	0.0092	0.0000	0.1532	75	58
25	162	0.0116	0.4769	0.1259	162	34
	184	0.0128	0.4035	0.1353	184	18
	139	0.0031	0.4349	0.1360	139	14
50	481	0.0246	6.2639	0.1149	469	1
	402	0.0351	6.8284	0.1167	387	1
	512	0.0201	7.4317	0.1148	420	1

Table S1. Benchmarking results for MWIS problem.

Table S1 shows the benchmarking results using Gurobi and D-Wave.

We use Gurobi 10.0 on classical hardware with Intel(R) Core(TM) i7-8665U CPU 1.90GHz, 16 GB RAM, 250GB SSD with Windows 10 and Python 3.9. The MWIS problem is implemented via the classical integer programming formulation given in eq. (1) of the manuscript. Only the objective function has been adjusted such that our problem becomes a minimization problem. We set a computing time limit of 300 seconds and MIP gap of 0.0001. No other Gurobi solver settings are altered. For the Gurobi results, we report the objective function value of the optimal solution and total running time for every instance of every size. For every instance, the solver status is 'optimal'. That is, the MIP gap is lower than the threshold.

For comparison, we use D-Wave Quantum Annealer's Version Advantage System 5.2 with over 5,000 qubits. We use 1,000 shots and annealing time of 40μs. We apply the formulation in eq. (3) of the manuscript, also transformed to a minimization problem. The resulting QUBO matrix is encoded in a dictionary and given to (embedded in) the D-Wave annealer. We apply standard embedding function of D-Wave Ocean, no further settings altered. For the D-Wave results we report the time for embedding the QUBO problem on the D-Wave architecture, time spent on the QPU (annealing time plus overhead), smallest objective function value of the solutions found in 1,000 shots and the number of shots that produce the smallest objective function value.

size	IBM	
(no. graph vertices)	Sol.	Time
	20	2.603
5	22	2.6855
	17	2.9454
	42	6.4216
10	50	4.0994
	69	8.9249

Table S2. Benchmarking time for MWIS problem using IBM QASM with statevector simulator.

For IBM devices and Qiskit, we first consider instances of size 5 and 10 using QASM (Version 0.39.4 QASM) with statevector simulator and the built-in QAOA algorithm with $reps = 3$ and COBYLA optimizer (Maximum number of function evaluations: maxiter=250). The hardware is the same as for the Gurobi experiments. We find the optimal solution for every instance. Although running time is not of interest here as the experiments are performed on classical hardware, we report the times in Table S2. For the real quantum hardware, we use the Falcon r6 QPU located in Ehningen, Germany with 27 qubits. We use Qiskit Runtime's built-in QAOAClient with alpha=0.75, the fraction of top measurement samples to be used for the expectation value, and standard settings such as $reps = 1$ and SPSA optimizer[1]. However, only the first instance of size 5 could be tested due to long queuing times and system offline times. Occasionally, the optimal solution occurred. Further data could not be retrieved.

We briefly discuss these results in Sections 5 and 6 of the manuscript. However, we emphasize that this is not a full benchmarking, thus the results (especially for the IBM device) offer only a quick look at the current capabilities of quantum devices for optimization problems. Better algorithms and more sophisticated benchmarking methods may exist for the comparison between classical and quantum methods. We briefly address this also in the manuscript.

[1] `https://qiskit.org/documentation/stubs/qiskit.algorithms.optimizers.SPSA.html`