Milestones on the Quantum Utility Highway

Catherine C. McGeoch and Pau Farré

D-Wave Systems Inc.

{*cmcgeoch, pfarre*}*@dwavesys.com*

May 2, 2023

Abstract

We introduce *quantum utility*, a new approach to evaluating quantum performance that aims to capture the user experience by including overhead costs associated with the quantum computation. A demonstration of quantum utility by a quantum processing unit (QPU) shows that the QPU can outperform classical solvers at some tasks of interest to practitioners, when considering computational overheads. We consider overhead costs that arise in standalone use of the QPU (as opposed to a hybrid computation context). We define three early milestones on the path to broad-scale quantum utility that focus on restricted subsets of overheads: Milestone 0 considers pure anneal time (no overheads) and has been demonstrated in previous work; Milestone 1 includes overhead times to access the QPU (that is, programming and readout); and Milestone 2 incorporates an indirect cost associated with minor embedding.

We evaluate the performance of a D-Wave Advantage QPU with respect to Milestones 1 and 2, using a testbed of 13 input classes and seven classical solvers implemented on CPUs and GPUs. For Milestone 1, the QPU outperformed all classical solvers in 99% of our tests. For Milestone 2, the QPU outperformed all classical solvers in 19% of our tests, and the scenarios in which the QPU found success correspond to cases where classical solvers most frequently failed.

Analysis of test results on specific inputs reveals fundamentally distinct underlying mechanisms that explain the observed differences in quantum and classical performance profiles. We present evidence-based arguments that these distinctions bode well for future annealing quantum processors to support demonstrations of quantum utility on ever-expanding classes of inputs and for more challenging milestones.

Figure 1: Subtasks associated with using a quantum annealing system. Input and output formulations are marked in blue; subtasks in teal are associated with problem translation. Tasks in orange correspond to access time, which is considered in Milestone 1. The orange arrows identify network transmission overheads in a cloud-based service model.

1 Introduction

Annealing-based QPUs manufactured by D-Wave[1] are designed to heuristically solve problems in combinatorial optimization. These systems operate within a declarative problem-solving paradigm: rather than implementing a program to solve a given problem, the user reformulates the problem to match a solver already implemented in quantum hardware and firmware. Reformulation creates subtasks, including translating the original input X into a *logical* graph input G that matches the problem solved natively, and minor-embedding[2] G to create a *physical* input P that matches the qubit connectivity structure in the QPU.

Figure 1 illustrates these subtasks, which create overhead costs that increase total computation time; the purely quantum component of this workflow is called an anneal. Note that this diagram applies only to a sequential workflow that arises in standalone use of the QPU. We do not consider hybrid computations, in which quantum and classical components operate concurrently on distributed platforms, creating different workflows and cost breakdowns.

We introduce the concept of **quantum utility**, which aims to capture quantum performance as experienced by the user. A demonstration of quantum utility is an affirmative answer to this question: *Considering overheads, can the quantum system outperform classical alternatives at some task of application interest?* Quantum utility is demonstrated when the quantum computation is fast enough to compensate for classical overhead costs of using the quantum system.

---

[1]D-Wave trademarks and registered trademarks used herein include D-Wave, Leap, Ocean, Chimera, Pegasus, Zephyr, Advantage, D-Wave Two, D-Wave 2X, D-Wave 2000Q, and Advantage2. Intel and Xeon are trademarks of Intel Corporation. NVIDIA and NVIDIA GeForce GRX are trademarks of NVIDIA Corporation.

[2]In graph theory, an embedding maps a graph G to another graph P, and a minor embedding maps G onto a minor of P (including P). Here we use both terms interchangeably to refer to the latter task.

A demonstration of quantum utility over a broad range of overheads and inputs must wait upon some future date (see Section 5). In the meantime, we identify milestones along the way: a milestone may be thought of as a test of quantum utility in a restricted context associated with specific overheads. We define the first three milestones as follows.

Milestone 0, no overheads: A demonstration that the QPU outperforms classical solvers that read the same (physical) inputs, measuring quantum annealing time. Section 1.1 reviews previous publications containing demonstrations of Milestone 0. We do not explicitly address it here, except to point out that a demonstration of Milestone 1 or 2 implies a demonstration of Milestone 0.

Milestone 1, access time: A demonstration that the QPU outperforms classical solvers reading the same (physical) inputs, when measuring QPU access time. Access time includes a *programming* step that occurs at least once per input, an *anneal* step that occurs once per output, and a *read* step that occurs once per output. On the Advantage system tested here, programming imposes a lower bound access cost of approximately 16 ms, before the first anneal can begin.

Milestone 2, indirect cost of embedding: A demonstration that the QPU, which reads a larger physical input, can outperform a classical solver that reads a smaller logical input, when measuring access time. That is, the minor embedding step maps a logical input defined on an arbitrary graph G (with n nodes and m edges), onto a physical input defined on a fixed hardware graph P (with q qubits and c couplers), such that $n+m \leq q+c$. This incurs both a direct cost to compute the mapping (not considered here), and an indirect cost in the sense that classical solvers may exploit a relative speedup due to this size differential.

Milestones 1 and 2 are arguably most important to achieving the above-mentioned consensus on quantum utility, because the relevant overheads are unique to quantum annealing and are common to most users' experience. That is, depending on the envisioned use case, the other overhead times in Figure 1 may be considered negligible or otherwise irrelevant to a performance comparison. Section 4.3 describes some application scenarios for which Milestones 1 and 2 might be considered sufficient to demonstrate real-world quantum utility.

Section 2 describes our test setup. We evaluate performance of an Advantage QPU with respect to Milestones 1 and 2, in comparison to seven classical solvers: two read physical inputs P and five read general inputs G; three run on GPUs and four run on CPUs. Our tests use 25 inputs each from 13 different classes. For each instance, solvers must return a sample of $s \in \{1, 10, 100, 1000\}$ independent solutions within a time limit of $t \in \{.02, .05, .1, .2, .5, 1\}$ seconds (that is, allowing up to t/s seconds per solution, depending on initialization overheads). A test *scenario* corresponds to a specific combination of s and t.

We perform 247 tests combining 19 scenarios and 13 input classes, and rank solvers according to the quality of solution samples returned. For each scenario and input class,

a *win* is awarded to a solver that finds solutions strictly better than the others. A *fail* is awarded if the solver is unable to return a complete set of s solutions within time limit t. Note that a few tests had multiple winners, and it is possible for a solver to neither win nor fail a given test. Results are summarized below.

- Section 3.1 presents Milestone 1 results. The Advantage QPU wins in 99% of tests (that is, all but 3 of 247). Some classical solvers fail in scenarios where t/s is smallest.

- Section 3.2 presents Milestone 2 results for eight input classes that have both pre-embedded (G) and post-embedded (P) versions. We compare the QPU, reading P, to five classical solvers, reading G, for a total of 152 tests in 19 scenarios. The QPU wins in 19% of the tests. The tests where Advantage wins correspond to largest n and smallest t/s; these are the same tests for which classical solvers most frequently fail.

- Section 4 identifies some fundamental differences in the computational mechanisms that drive quantum and classical performance, which explain the observed contrary trends in quantum versus classical patterns of wins and losses. We present evidence-based arguments that these fundamental differences bode well for demonstrations of these two milestones on more varied types of inputs, as well as demonstrations of more challenging milestones, using future annealing-based QPUs.

We remark that these results represent a watershed moment in the development of quantum annealing processor technologies over the past decade. With the few exceptions mentioned in Section 1.1, previous-generation QPUs were too small, and access times too high, to support positive outcomes on milestone tests such as these. As discussed in Section 2.1, Advantage QPUs have reached a threshold of size and connectivity sufficient to hold inputs that require more than 16 ms to be solved classically; as a result, Milestones 1 and 2 (and beyond) have become feasible to test.

Nevertheless, these are just the first steps of a longer journey toward routine demonstrations of quantum utility over a broad range of overheads and inputs. As discussed in Section 2.1, our tests using a current-generation Advantage QPU are incomplete because other input classes can be identified for which logical inputs remain too small to serve as viable test candidates. See Section 5 for more about this point.

1.1 Previous Work

Figure 2 lists published papers that meet our criteria for demonstrating Milestone 0 (and in some cases Milestone 1): they measure both solution quality and computation time, and report superior QPU performance on a significant proportion of inputs tested. These results span a variety of application domains (including quantum simulation, combinatorial optimization, and diverse sampling), input types, performance metrics,

Year	Source	QPU	Milestone	Metric	Solvers
2013	[44]	D-Wave Two	0*,1	A	EX(1) HS (1) PS(1)
2015	[59]	D-Wave 2X	0	B	EX(2) HS(3)
2015	[35]	D-Wave 2X	0, 1	A	HS(2)
2016	[12]	D-Wave 2X	0, 1*	C	HS(1) MC(1)
2017	[41]	D-Wave 2000Q	0, 1	B	HS(1)
2019	[36]	D-Wave 2000Q	0, 1	C & D	HS(2) MC (2)
2019	[37]	D-Wave 2000Q	0,1	D	HS(1)
2019	[48]	D-Wave 2000Q	0*, 1	B	EX(3) HS(2) MC(1)
2020	[27]	D-Wave 2X	0	B	EX(1) HS(1)
2020	[24]	D-Wave 2000Q	0	B	PS(2)
2021	[33]	D-Wave 2000Q	0, 1*	E	MC(1)
2021	[3]	Advantage	0	C	HS(3)
2022	[51]	D-Wave 2000Q	0	C	HS(1)
2022	[58]	(several)	0, 1	C	EX(1) HS(2) MC(1)
2022	[38]	D-Wave 2000Q	0	B	PS(1)

Figure 2: Previous work demonstrating Milestones 0 and 1 using past- and current-generation D-Wave QPUs. An asterisk marks a milestone that was not explicitly addressed but can be inferred from the published results. The Metric column refers to the figure of merit used in the evaluation, and the Solvers column lists categories of classical solvers used for comparisons. Column entries are decoded in the text.

and classical competition: we consider such variety to be necessary to reaching consensus about whether quantum utility has been demonstrated.

The QPU column refers to previous and current generation D-Wave QPUs, with nominal qubit counts as follows: D-Wave Two (512), D-Wave 2X (1000+), D-Wave 2000Q (2000+), and Advantage (5000+). The first three generations were based on the Chimera connection topology, with 6 couplers per qubit; the Advantage generation of QPUs is based on the Pegasus topology with 15 couplers per qubit.

In the Milestone column, an asterisk marks a milestone that was not explicitly measured in the paper but can be inferred from the results. For example, Milestone 1 implies Milestone 0 because anneal time is always less than access time. Milestone 1 can be inferred from Milestone 0 when a reported computational speedup is large enough (that is, 10,000-fold or higher) to easily account for access overhead times. Tasseff et al. [58] test on several available D-Wave 2000Q and Advantage systems; they consider Milestones 0 and 1, as well as overhead costs of network transmission and queuing time, which constitutes a milestone somewhere beyond Milestone 2. (Our Milestone 2 is not relevant to their work because inputs are not minor embedded.)

The Metric column identifies the type of performance metric(s) used in each study, as follows.

A Best solutions returned within a fixed time limit
B Better solutions faster, considering both solution quality and runtime
C Fastest time to an optimal solution
D Fastest time to sample all near-optimal valleys in the solution space
E Fastest convergence of solution samples to a target distribution

The Solvers column identifies categories of classical solvers tested in each work; numbers in parentheses refer to multiple versions of solvers within the category, as follows.

- Exact (EX) solvers guarantee to return optimal solutions if given enough computation time. When used in tests of heuristic performance that accept non-optimal solutions (as with metrics A, B, D, E), these solvers report working solutions found at earlier stopping times. Algorithms of this type include Branch-and-Bound, Integer Linear Programing, and Message Passing.

- Heuristic Search (HS) solvers typically start with a random candidate solution and then iterate, making incremental changes to the candidate while seeking a downhill path towards better-quality solutions. HS solvers vary according to, e.g., mechanisms for choosing next-moves, strategies for escaping local minima, or the number of multiple solution paths being pursued. HS solvers in the published works include Greedy, the Hamze-de Frietas-Selby (HFS) method [17, 55], Simulated Annealing, TABU search, Parallel Tempering, and population-based (genetic) methods.

- Monte Carlo (MC) search represents an important subclass of HS methods. An MC solver performs a guided random walk to explore the solution landscape using

a probabilistic strategy that simulates a natural process. Variations of MC in the published works of Figure 2 include Markov Chain MC, path integral MC, quantum MC, spin vector MC, and Wolff cluster MC. (The Simulated Annealing and Parallel Tempering heuristics arguably belong to this paradigm, although implemented solvers can vary widely in their fidelity to a natural model; our classification follows authors' nomenclatures.)

- Problem Specific (**PS**) solvers read original inputs X for a specific problem domain, rather than translated logical G or physical P graphs. The differential in problem size between X and P, together with measurement of access time, would correspond to a milestone somewhere beyond our Milestone 2.

Other empirical work. A number of papers that compare performance of D-Wave QPUs against classical solvers are not included in Figure 2, for reasons briefly summarized below.

- Several papers consider a performance metric known as *quantum speedup* [5, 20, 29, 30, 39, 54, 66]. This body of work aims to evaluate how computation time *scales* with problem size over a finite range. Scaling analyses look at the shape of a curve showing computation time versus problem size, but reported times are not intended to accurately reflect total computation times. Instead, these papers typically report times for a core operation (the part that scales with n), and some evaluate classical performance on a hypothetical parallel platform (dividing core times by n). Furthermore, these papers look at QPU performance against the "best available" classical solver, which involves extensive precomputation to optimize solver parameters, contrary to the Fair Test policy described in Section 2.2.1. Lacking information about true computation times under realistic use scenarios, we cannot determine whether Milestone 0 has been demonstrated in these papers.

- Some papers compare quantum and classical solution quality — reporting both positive and negative results — but omit computation times; others report computation times but apply unequal levels of computational work [21, 23, 28, 31, 64]. Some negative results in these papers can be attributed to small QPU size, as discussed in the introduction. In any case, without a way to calibrate computational work, we cannot determine which solvers would give superior results under comparable use conditions; see Section 4 for discussion of this issue.

1.2 Overhead Costs of QA

This section describes two categories of overhead costs that are considered in our benchmark tests.

cubic graph with 180 nodes and 270 edges

embedded in P4 using 248 qubits and 338 couplers

(a)

(b)

Figure 3: Minor embedding. (a) General graph G. (b) A minor embedding of G onto a Pegasus graph P. The highlighted black node in G is split into a chain of two nodes in P; both the original node and the chain are connected to three neighbors (pink, green, red). Each qubit in Pegasus is connected by couplers to 15 neighbors (fewer on boundaries). Qubits and couplers that are unused in the embedding are shown in pale gray.

1.2.1 Minor Embedding

Milestone 2 considers an indirect cost associated with minor embedding as follows. The Pegasus graph defines the qubit and coupler connectivity structure inside all Advantage QPUs. The specific hardware graph inside any given QPU is a subgraph of Pegasus, due to a small proportion of disabled qubits and couplers that do not meet specifications; see Section 2.2 for details about the system used in our tests.

Given an arbitrary logical graph G, minor embedding maps each node of G to a *chain* of one or more connected qubits in the hardware graph. (Despite the name, they need not be connected in a strictly linear sequence, and tree-like chain structures are common.) Minor embedding is necessary for any G that is not a subgraph of the hardware graph; in particular, minor embedding is useful for mapping high-degree nodes of G onto limited-degree nodes of P. For example, a logical node x of degree 20 can map to a two-node chain $(q1, q2)$, each of degree 10, plus an additional chain edge $(q1, q2)$ tying them together. The *chain strength J_{chain}* is a weight applied to all new chain edges that are introduced by minor embedding. Our tests use a default chain strength that is recommended by an Ocean system utility (more discussion below).

In Figure 3, panel (a) shows a graph G with $n = 180$ nodes and $m = 270$ edges, totaling 450 components. Panel (b) shows G after being minor-embedded onto a Pegasus graph containing $q = 248$ qubits and $c = 338$ couplers, totaling 586 components. Each node in G is mapped to an equal-colored chain of one or more nodes in P; for example, the oversized black node with three neighbors (green, orange, red) is represented by a two-qubit chain in the embedding. In terms of total *input size* (nodes plus edges), the embedded graph has $586 = 248 + 338$ qubits and couplers, about 1.4 times larger than G.

Minor embedding incurs two types of overhead costs: (a) the computation time to find a mapping from G to P and apply it; and postprocessing time to unmap solutions for P back to solutions for G (a direct cost that is not considered in this paper), and (b) the indirect cost considered in Milestone 2, whereby classical solvers reading G can work on a smaller input of size (n, m), while the QPU works on a larger input of size (q, c). Under the reasonable assumption that smaller inputs are easier to solve, this size difference creates a potential runtime advantage for classical solvers relative to the QPU. Our tests measure this difference by running all solvers for equal time limits, and comparing solution quality with respect to the logical problem G, after QPU solutions to P been untranslated back to solutions for G.

1.2.2 Access Time

Milestone 1 incorporates *access time*, which, in addition to purely quantum annealing time, includes the time needed to transfer information on and off the quantum chip, as follows.

First, when given a physical input graph P defined by weight vectors (h, J, J_{chain}), the QPU performs at least one *programming*[3] step to map (h, J, J_{chain}) onto the analog control system that drives qubit states during the anneal. On the Advantage_system4.1 QPU used in our tests, this operation takes $t_{program} = 16$ ms. Multiple programmings can be used to make small adjustments to the physical problem representation. In our tests, reprogramming was applied for two reasons:

- For native inputs, each programming step applies a random spin reversal transform (SRT). An SRT assigns the physical output states (up,down) of a subset of qubits to be either $(-1, +1)$ or $(+1, -1)$. This random assignment counteracts certain types of physical biases in the quantum hardware.

- For embedded inputs, each programming step increments or decrements chain strength within a small range of values $J_{chain} \in [0.5x, \ldots, 2x]$, where x is the default chain strength suggested by an Ocean system utility.

In both cases, the modification can shift the distribution of sampled solution energies higher or lower by a small amount. Multiple programmings have the effect of widening the range of energies in the full set of sampled solutions, which may potentially move sample minimums closer to ground states.

Second, after one anneal produces a solution to input P, a *readout* operation is performed to move the solution off the chip. On the Advantage system used in our work, $t_{read} \leq 0.241$ms. Smaller readout times can be observed when only a portion of the chip is occupied by input weights, but our tests focus on large inputs and readout times were fairly consistent. (Both programming and readout times include other low-level operations not discussed here.)

Three annealing control parameters were varied in our tests: the number of programmings p, anneal time per solution t_{anneal}, and the number of anneal-plus-readout operations r. This works out to $\lfloor r/p \rfloor$ solutions per programming, with the last batch rounded up if necessary to meet a given time limit t. Access time to return r solution to an input instance P is therefore measured as

$$t_{access} \quad = \quad p \cdot t_{prog} + r \cdot (t_{anneal} + t_{read}). \tag{1}$$

2 Experimental Setup

This section presents an overview of inputs, solvers, and performance metrics used in our tests.

---

[3]Here, *programming* and *readout* are used as umbrella terms that include several subtasks. When applicable, subtask times are set to constant default values throughout, such as `qpu_delay_time_per_sample` $= 20 \, \mu$s, which is performed during readouts.

The Problem   The Ising Model (IM) problem is defined as follows: given a graph $G = (V, E)$ on n nodes and m edges, together with real-valued weights $h = \{h_i\}$ (called fields) on nodes and $J = \{J_{ij}\}$ (called couplings) on edges, assign spin values $x_i \in \{-1, +1\}$ to nodes so as to minimize the energy function

$$F(x) \;\; = \;\; \sum_{i \in V} h_i x_i + \sum_{(i,j) \in E} J_{ij} x_i x_j. \tag{2}$$

This problem is NP-hard when G is nonplanar [25]. In physics applications that model natural phenomena, an optimal solution is called a ground state and non-optimal solutions are called excited states. The Quadratic Unconstrained Binary Optimization (QUBO) problem, more familiar to researchers in computer science and operations research, is to optimize the same objective function (2), defined on binary variables $b_i \in \{0, 1\}$ instead of spins. Under the trivial transformation $x_i = 2b_i - 1$, the objective functions for QUBO and IM have equivalent cost spectra, except for a constant offset that varies by instance.

Either formulation can be used as input to the QPU, and we generically refer to both as binary quadratic models (BQMs). Our input generators construct BQMs of either type, according to their motivating application domains.

2.1   Inputs

Figure 4 summarizes the 13 input classes that were selected for our study. Columns labeled n and m show the number of nodes and edges in logical (pre-embedded) graphs; column q shows the number of nodes in the physical (post-embedded) graphs. The last column shows the mean ratio of nodes in embedded versus unembedded versions of each graph category, equivalent to mean chain length. Inputs are listed in decreasing order by problem size n, and are grouped by the strategy used for embedding, as follows.

native Five input classes are generated directly on the Pegasus hardware graph in the Advantage QPU; their logical and physical versions are identical. These are the largest inputs in our tests, using nearly all available qubits and couplers.[4]

These inputs use weighting schemes (h, J) that have been identified as challenging for classical and quantum solvers to cope with. The labels refer to corrupted biased ferromagnets (CBFM) from [58]; native spin glasses (NAT1, NAT7) discussed in [29]; frustrated tiles (TILE) [50]; and frustrated cluster loops (FCL) from [36].

---

[4]The Advantage_system4.1 QPU in our tests contains a small proportion of unused qubits and couplers that did not meet quality standards and were disabled when the system came online. Active qubits and couplers in the hardware graph number 5627 and 40279, representing yields of 99.77% and 99.49% respectively, of the full Pegasus graph. Our tests use a subgraph of that hardware graph because of a design limitation of our GPU-based native solvers, which ignore some irregularly-connected qubits on the periphery.

Name	Embedding	n	m	q	$L = q/n$
CBFM	native	5387	38751	5387	1.0
NAT1	native	5387	38751	5387	1.0
NAT7	native	5387	38751	5387	1.0
TILE	native	5387	38751	5387	1.0
FCL	native	$\overline{5029}$	$\overline{22030}$	$\overline{5029}$	1.0
3DLAT	custom	2688	7444	5376	2.0
BPSP	heuristic	$\overline{867}$	$\overline{1211}$	$\overline{3037}$	$\overline{3.4}$
DREG03	heuristic	754	1131	$\overline{2856}$	$\overline{3.8}$
SOCs	heuristic	355	1053	$\overline{1263}$	$\overline{3.4}$
SOCu	heuristic	355	1053	$\overline{1263}$	$\overline{3.4}$
SK	clique	175	15225	2625	15
CDMA	clique	175	15225	2625	15
DAIG	clique	$\overline{174}$	$\overline{15051}$	2625	15

Figure 4: Our testbed of input classes. Input sizes (nodes n and edges m) correspond to the largest graph of each class that can be reasonably embedded onto the QPU. Column q shows the number of qubits used in physical (embedded) versions of these graphs. In the last column, L denotes mean chain length. This is the mean ratio of physical to logical problem sizes: for example, physical SK inputs have 15 times more variables than logical SK inputs. Numbers with overlines are averages taken over the 15 randomly generated graphs or embeddings in a given class.

custom Graphs with regular lattice-like structures can exploit efficient custom embedding techniques. Three-dimensional lattices (3DLAT), consisting of unit cubes that share corner nodes, arise in scientific applications and simulations [19].

heuristic Four input classes with random and irregular connection structures are embedded using the heuristic embedding tool `minor_miner` available in D-Wave's Ocean SDK [2]. Labels refer to inputs for the Binary Paint Shop Problem (BPSP), similar to those in [56]; random 3-regular graphs (DREG03); and signed and unsigned social network graphs (SOCs, SOCu) [16].

clique Three input classes are embedded using the `busclique` clique-embedding tool available in Ocean SDK. They are: Sherrington-Kirkpatric (SK) graphs studied in statistical physics [62]; inputs for the Code Division Multiple Access (CDMA) signal decoding problem; and dense AI graphs (DAIG), generated by a tool that "learns" hard substructures of multiple application problems. This tool constructs inputs such that n is always even.

Screening Inputs for Hardness Given an input graph G with n nodes and m edges, the *problem size n* is the number of variables that are assigned solution values, and *input size $k = n + m$* is the number of graph components (node and edge weights) required to fully specify the problem via formula (2). Problem size is a key indicator of input hardness, in the sense that the space of solutions to be explored grows as 2^n. Input size gives an upper bound on embeddability, since it must hold that $n + m \leq q + c$.

We performed a pilot study to identify interesting input classes for our benchmark tests, prioritizing hardness, structural diversity, and application relevance. Our tests require inputs that are both small enough to embed onto the Pegasus hardware, and large enough that Milestones 1 and 2 are at least feasible to demonstrate. That is, if the problem is so small or easy that classical heuristics can regularly find ground states in less than the 16 ms lower bound on programming time — before the quantum computation has a chance to begin — then there is no point in performing the test. We argue that too-small or too-easy inputs are not interesting candidates for quantum solution methods because they can be quickly solved classically, leaving little room for improvement.

The native, custom-embedded, and clique inputs passed our screening criteria. We included three clique-embedded input classes to represent this important boundary case; among embedded graphs, these are the smallest in terms of problem size n but the largest in terms of input size k.

For the remaining heuristic-embedded inputs, we applied a two-step screening procedure as follows.

1. We used the heuristic embedder to find a largest *reasonable* input size, selecting K such that the median value (over 3 random trials) of maximum chain length

Name	Label	Inputs	Platform
D-Wave `advantage_4.1`	QPU	physical	QPU
Simulated Annealing-Optimize	SAo_native	physical	GPU
Simulated Annealing-Sample	SAs_native	physical	GPU
Random	Random	general	CPU
Steepest Greedy Descent	SGD	general	CPU
Simulated Annealing (Logical)	SA	general	CPU
Parallel Tempering (sequential)	PTc	general	CPU
Parallel Tempering (parallel)	PTg	general	GPU

Figure 5: Solvers in our benchmark study. The third column indicates whether the solver reads physical or general inputs. The fourth column shows the type of platform on which the solver runs.

(per trial) is in the range $15 \leq L_{max} \leq 20$, slightly higher than found in clique embeddings.

2. The Greedy and SA heuristics were run on these largest input; if they could find putative ground states — as indicated by agreement on the same minimum energy over many random trials — within the 16 ms lower bound on programming time, the candidate were omitted from our main tests.

As it turned out, the inputs rejected by this screening procedure were of sizes $175 < n < 355$, both smaller and denser than those that passed.[5] Benchmark tests on moderately dense graphs must await availability of larger and more-connected hardware graphs.

We consider the fact that eight embedded graph classes *were* identified as test candidates to be an encouraging sign of progress, since (with a few exceptions mentioned in Section 1.1) previous-generation QPUs have been too small for logical inputs of embeddable size to be considered viable candiates for the benchmarking tests in this paper.

2.2 Solvers

The table in Figure 5 lists the heuristic solvers in our study. The first three read only physical inputs defined on the Pegasus hardware graph, which may be either native or post-embedded versions of logical graphs. The remaining five solvers read general graphs, which can be either pre-embedded logical inputs or native inputs. Both physical solvers and one general solver (PTg) are implemented on GPUs; the remaining four general solvers are implemented on CPUs.

---

[5]These included four variations on random Satisfiability inputs, and random k-regular graphs with $k \geq 10$.

Classical solvers   The classical solvers all implement variations on the heuristic search paradigm; these are sometimes called landscape-traversal methods, based on a popular metaphor that envisions the solution space as a landscape defined over all possible solutions x, where the surface elevation at point x corresponds to the objective function value defined by formula (2). A landscape-traversal solver starts at a random solution point x and then moves step-by-step through the landscape, making incremental changes to x by flipping one (or more) binary variables, while seeking lower ground. A *local minimum* is a valley surrounded by uphill moves; a *global minimum* is a valley of minimum elevation.

Total work w corresponds to the number of iterations performed while moving about the space; this is an upper bound on the total number of states visited, since some solvers can repeat visits to the same states. More work is associated with longer computation time (the exact relationship depends on the implementation), but tends to produce better-quality solutions. Different solvers manage this tradeoff differently, as follows.

- The **Random** solver generates an initial solution and makes no effort to improve it, so $w = 0$. This solver provides baseline measurements of the fastest runtimes and worst-quality solutions among landscape-traversal heuristics.

- A greedy solver moves strictly downhill. At each iteration, the steepest gradient descent (**SGD**) solver studied here considers all single-variable flips and selects the one giving the largest improvement in (2), stopping and reporting the result when no improvements can be found. Work w is determined by the number of visited states between a random start and a nearby local minima, and solution quality reflects the distribution of local minima reachable from random starts.

- Our simulated annealing (**SA**) solver has total work controlled by the user via a `num_sweeps` parameter. The sweep loop iterates over n nodes when deciding where next to step, so that $w = n \cdot$ `num_sweeps`. A temperature parameter τ is used to probabilistically decide which node is used for the next-step, which may be uphill; τ decreases with sweep count so that uphill moves become less likely over time. This approach allows SA to escape local minima earlier in the computation, becoming more strictly greedy in later stages.

- The **SA-native** solver is a GPU-based version of SA that parallelizes the sweep loop by assigning n nodes to n CUDA cores, so that the computation of next-step probabilities can exploit n-way parallelism. Note that the step itself is not parallelized because the GPU can only be in one state at a time and computation of next-state probabilities depends on the previous state; that is, parallelization speeds up time-per-sweep, but the work/iteration loop is inherently sequential since the computation can only be in one state at a time. This solver reads only Pegasus-structured inputs. We implemented two versions: for **SAo_native**, parameters are set to prioritize optimization (more rapid descent to a low-energy solution), and for **SAs_native**, parameters are set to prioritize sample diversity (less likely to be stuck in local minima).

15

- Parallel tempering (**PT**) is based on SA but contains an extra loop to represent R *replicas* of SA, each working on its own solution x_r and operating at a constant temperature τ_r. At intervals, solutions are swapped among adjacent replicas (i.e., the replica at τ_r swaps with τ_{r+1} and τ_{r-1}) to move the best candidate solutions toward lower τ. The user parameter `num_sweeps` controls the total number of sweeps for all replicas. Over time, the number of replicas can grow or shrink according to swapping frequencies; for a mean number of replicas $\overline{R}$, total work is equal to $w = \overline{R} \cdot n \cdot$ `num_sweeps`. We tested two implementations of PT: **PTc** runs sequentially on a CPU; **PTg** parallelizes computations in the sweep and the replica loops, and runs on a GPU.

Quantum annealing solver See [42, 61] for introductions to computation by quantum annealing. Briefly, the QPU control system follows a time-varying Hamiltonian $\mathcal{H}(t)$ over a time interval $t : 0 \to t_{anneal}$, to create a smooth transition from an initial transverse field Hamiltonian $\mathcal{H}_{initial}$ to a problem Hamiltonian $\mathcal{H}_{problem}$ that corresponds to the objective function (2). The qubits behave like a quantum particle process driven by $\mathcal{H}(t)$, naturally seeking their collective ground state, which corresponds to an optimal solution to (2) at the end of the anneal.

In landscape-traversal terms, the computation can be visualized as taking place on an initially flat landscape (the transverse field), from which the problem landscape defined by (2) gradually emerges. Rather than stepping point-by-point through this moving landscape, the qubits exploit quantum superposition and entanglement to probabilistically represent "all states at once," in such a way that highest probabilities track lowest-lying areas of the landscape as it evolves. At the end of the anneal, the classical problem Hamiltonian $\mathcal{H}_{problem}$ dominates $\mathcal{H}(p)$: classical state is read according to those final probabilities and returned as the solution.

Note that our classical conception of work does not translate well to the quantum computation because qubits do not visit states one-by-one; indeed there are no iterations to be counted. Instead, we modify our definition of work to match the natural quantum unit of computational effort, and set $w = t_{anneal}$.

In an ideal noise-free environment, if t_{anneal} is above a certain threshold time determined by the input, the qubits will finish in a ground state of the objective function (2) with high probability [14, 46, 57]. In a real-world quantum system, noise and imprecision of the analog control system can cause the qubits to move away from their ground state and instead finish in a low-energy state of (2) [4, 65].

2.2.1 Fair Test Policy

The QPU, SA, and PT solvers offer several runtime parameters to the user, which can be tuned to elicit best performance on specific input types. The question of how to design a so-called *fair test*, which ensures equal parameter-tuning effort across all solvers, has

received much attention in the optimization methodology literature [6, 13, 22, 26, 45]. The goal of fair testing is to ensure that outcomes can be replicated by practitioners; Johnson [26] recommends that all tuning procedures be well-defined and algorithmic, with tuning time included in reported runtimes.

Following this advice, we implemented a wrapper code for each solver that accepts exactly three inputs — the instance (G or P), the sample size s, and the time limit t — and sets solver parameters to either fixed defaults or auto-tuned values according to code-based policies, with auto-tuning included in measured runtimes. The most important parameters, controlling work, were auto-tuned as follows.

- For Random and SGD, which offer no user parameters, the wrapper code simply takes a maximum number of samples within the time limit t. A sample of s lowest-energy results is subsampled from the full set during data analysis.

- For the SA and PT solvers, best results come from maximizing work to meet (s, t). The wrapper code sets the `num_sweeps` parameter to the maximum possible within the per-solution time limit t/s, according to quick time-per-sweep estimate that is performed once per solver per instance and re-used for all combinations of (s, t). A separate procedure occasionally checks progress against the full time limit t and makes adjustments to `num_sweeps` if necessary.

- For the QPU, maximizing anneal times to meet test limits of $t \in [20 \ldots 200]$ ms is not possible under current (public access) policies, which support anneal times in range $t_{anneal} \in [.0005, \ldots, 2]$ ms. Furthermore, we do not necessarily expect best results from maximizing this parameter: experience suggests a law of diminishing returns whereby setting t_{anneal} to higher values within its range yields no perceptible improvements in solution quality.

  Therefore our wrapper code aims to maximize quantum utilization with a default anneal time set to $t_{anneal} = t_{readout}$, and default r set such that $t_{program} = r(t_{anneal} + t_{readout})$. This time block is multiplied to meet a given test scenario (s, t), with reprogramming steps as described in Section 1.2.2. If this default setting does not allow enough time for the required number of samples, then reads are increased and programming steps are reduced; if this is still not, anneal time is reduced. As with Random and SGD, if the QPU returns more samples than needed in a given scenario, the s-best solutions are subsampled during analysis.

2.3 Performance measurement and analysis

We use one common test procedure to gather data for both milestones as follows. For each of 13 input classes we generate 25 random instances. A native input has one version that is sent to both logical and physical solvers; a logical input has a pre-embedding version sent to logical solvers (Random, SGD, SA, PTc, PTg) and a post-embedding version sent to physical solvers (QPU, SAo-native, SAs-native).

For each input instance and each solver, we construct a test scenario (s, t), requesting $s = [1, 10, 100, 1000]$ solutions to be returned within time limit $t = [.02, .05, .1, .2, .5, 1]$ seconds. Of 24 possible scenarios, we omit five because t/s is below the (amortized) lower bound on QPU access time; thus, t/s is in range $[.0002 \ldots 1]$ seconds in our tests.

In Milestone 1 tests using 13 input classes, all solvers read identical inputs: we compare QPU performance to logical solvers on the five native classes, and to physical solvers on all classes, and solution energies are calculated in the physical problem space. In Milestone 2 tests using eight logical inputs, the five logical solvers read pre-embedding inputs while the QPU reads their post-embedding physical versions. The physical solutions returned by the QPU are mapped back to the logical solution space for evaluation; broken chains are resolved using the default parameter setting `chain_voting = True`.

Statistics of sample solution quality are calculated as follows.

- Let $\mathcal{S}(a, x, s, t)$ be a sample of s energies returned by solver a for instance x when run for time limit t. Sample solution quality is measured by **median sample energy**, denoted $\mathcal{M}(a, x, s, t)$.

- For each instance x, the **target energy** $\mathcal{T}(x)$ is the minimum energy observed in all tests over all solvers and scenarios. This is not necessarily the optimal energy for x.

- The **relative error** for solver a on instance x is the absolute scaled difference between $\mathcal{M}(a, x, s, t)$ and $\mathcal{T}(x)$,

$$\mathcal{R}(a, x, s, t) \quad = \quad \frac{|\mathcal{T}(x) - \mathcal{M}(a, x, s, t)|}{|\mathcal{T}(x)|}. \tag{3}$$

Relative error is equal to zero when at least half of the sample energies found by a on input x are equivalent to the target energy. When relative error is nonzero, $\mathcal{R}(a, x, s, t)$ indicates how close the solver came to finding target energies.

3 Results

Figure 6 shows example results for three input sets from our tests: NAT1, 3DLAT, and SK. These inputs all have spin-glass weight with fields $h_i = 0$ and couplings chosen u.a.r. from $J_{ij} \in \{-1, +1\}$. They represent three extremes of graph structures and sizes: the native Pegasus graph (NAT1, $n = 5387$), among the largest tested; three-dimensional lattices (3DLAT, $n = 2688$), the largest embedded graphs; and Sherrington-Kirkpatric graphs build in cliques (SK, $n = 175$), among the smallest of embedded graphs.

Each row shows results for four test scenarios, corresponding to sample sizes $s = [1, 10, 100, 100]$ at the time limit $t = .5$ seconds. Each panel shows curves for median relative error $\mathcal{R}(a, x, s, t)$ over 25 instances, sorted in increasing order and color-coded

Figure 6: Performance results for the NAT1 (top), 3DLAT (middle) and SK (bottom) input sets, for time limit $t = .5$ seconds and sample sizes $s = [1, 10, 100, 100]$. Each panel shows a plot of median relative error for 25 inputs from the set, sorted in increasing order. These are symlog [53] plots with y on a logarithmic scale that switches to linear to expose the critical region near $y = 0$: the apparent step functions are due to the switch of scale and not indicative of discontinuities in the data.

by solver: the rightmost point is the worst $\mathcal{R}$ returned by the solver over all 25 instances, and the midpoint $x = 13$ is the median result over all instances.

These are *symlog* plots [53] with a logarithmic y-scale that becomes linear near $y = 0$, which give the best view of outcomes in our rank-based analysis; the steps at low y are due to the scale change and are not indicative of significant discontinuities in the data.

For each input class and scenario, we evaluate solver performance as follows.

- A *win* is awarded to a solver if its ECD curve is strictly below those of all other solvers on at least half (13) of the instances. For example, the QPU wins against all solvers on all four NAT1 scenarios. These wins are recorded for Milestone 1 because all solvers read the same physical inputs.

  A win is also awarded to a solver that ties with other winning solvers on at least half the instances. For example, in the first SK scenario ($s = 1$), SA, PTc, and PTg share the win, but SGD does not because it ties in fewer than 13 instances.

  A win for Milestone 1 is awarded to a physical solver that outperforms other physical solvers reading embedded problems. For example, the QPU outperforms both SAs_native and SAo_native in all SK scenarios.

- A *fail* is awarded if a solver cannot return all requested samples within the time limit, on at least half the inputs tested; if it fails on all inputs the ECD line is absent. For example, in the fourth NAT1 panel with $s = 1000$, all logical solvers fail on all 25, and the three physical solvers succeed.

A solver that neither wins nor fails in a given scenario is said to compete.

3.1 Milestone 1: Physical Solvers and Inputs

Figure 7 summarizes Milestone 1 results, which consider cases where all solvers read the same physical inputs: QPU versus logical and physical solvers on eight native inputs, and versus physical solvers on five embedded inputs. Panel (a) shows tallies of QPU wins over all 13 input classes in 19 scenarios, comprising 475 tests. The QPU clearly dominates classical approaches, winning in all but 3 of 475 tests covering 13 input classes and 19 scenarios (98.5 percent).

Panel (b) shows a tally of number of classical fails per input class in each scenario, considering the five native inputs on which all seven solvers were tested. The number of possible failures in any test scenario is between 0 and 7; each box shows the mean number of fails per scenario, averaged over all five native inputs.

Here are some details.

- Nearly all of the QPU wins are solo wins: a few ties with SAo_native and SAs_native were found in scenarios with large t/s (lower left corner), on the smallest (clique) inputs in our tests.

20

Figure 7: Panel (a): Number of wins or ties earned by QA in Milestone 1 tests, over all scenarios and (physical) inputs. The maximum number of wins per scenario is 13, the number of input classes tested. Panel (b): number of classical failures in tests using physical inputs, averaged over all input classes. The maximum number of failures per input class is seven or two, equal to the total number of solvers tested.

- The three cases where the QA does not win are near-ties versus SAo_native on CDMA inputs. These inputs are distinguished from others in our testbed by containing unusually high-precision Gaussian weights J, h. Higher precision poses a challenge to quantum annealing due to finer gradations of elevation in the solution landscape, which are harder for the QPU to distinguish.

- Considering only classical solvers, the GPU-based SAo_native and SAs_native solvers performed best overall, confirming the intuition that parallelism can boost performance of the SA heuristic.[6] It is interesting to note that on these 5000-node Pegasus inputs, GPU parallelism yields only about a 60-fold speedup over comparable CPU solvers; on previous-generation Chimera inputs, GPU solvers saw 133-fold speedups. This degradation in speedup relative to sequential solvers can be attributed to the fact that parallel speedup is reduced by increased graph connectivity: the Pegasus graph is nonbipartite with degree 15, whereas Chimera is bipartite with node degree 6.

- The upper right boundary where classical solvers are most likely to fail corresponds to small t/s, below about 0.5 milliseconds. The physical solvers (QPU and SA_native) never failed in any scenario. Among the logical solvers, PTc and PTs failed most frequently.

---

[6]GPU solvers implemented in C++ and CUDA, and run on an NVIDIA GeForce GTX 1080-Ti processor with 3584 CUDA cores and 1582MHz processor clock speed. All CPU tests ran on a 3.30GHz Intel Core i9-7000x CPU with 20 cores.

Figure 8: Milestone 2. Panel (a) Number of QA wins (out of eight maximum) over all logical inputs and scenarios. Panel (b) is the mean number of classical fails (out of five maximum), averaged over eight input classes.

3.2 Milestone 2: Logical Solvers and Inputs

In Milestone 2 tests we consider performance on eight general input classes, for which five logical solvers read smaller pre-embedding versions and the QPU reads larger post-embedding versions of the same problems. (Performance of physical solvers versus the QPU are considered in M1 test). Thus, at most eight wins and at most five fails can be awarded per scenario.

Results appear in Figure 8. Panel (a) shows that the QPU wins in 28 of 152 cases, or about 18% of problem scenarios. In other cases, the QPU competes (neither wins nor fails). Panel (b) shows the mean number of classical fails per scenario per input class. Although classical solvers do a better job overall in this milestone, it is interesting to note that the scenarios where classical solvers can fail are all on the upper right boundary, just as in Figure 7. Panel (a) shows that these are the same scenarios where the QPU solver is most likely to win. Here are some details.

- The QPU is more likely to win, and classical solvers more likely to fail, in tests involving small t/s. Furthermore, these outcomes are strongly correlated with problem size. In Milestone 1, QPU wins nearly always when $n \geq 5029$. In Milestone 2, QPU wins occur on the three largest embedded input graphs: 3DLAT ($n = 2688$) in 18 scenarios, BPSPh ($n = 867$) in 6 scenarios, and DREG03 ($n = 754$) in 4 scenarios.

- Among classical solvers, SA was the sole winner in most cases. The exception is scenarios with large t/s (lower left corner) and small n (e.g. clique problems), for which many solvers shared the win. PTc and PTg failed most frequently at small t/s, as in M1 tests.

- The GPU-based PTg solver never outperformed its CPU-based counterpart in these tests; sometimes they tied, and sometimes PTc outperformed PTg. We

Graph	Problem Size n	Edges m	Input Size $K = n + m$	Chain Length $L = q/n$
Native	5387	38751	44138	1.0
3DLAT	2688	7444	10132	2.0
Clique	175	15225	15400	15

Figure 9: Sizes of three graph structures used in our tests.

attribute this outcome to the fact that algorithms for irregularly structured graphs are notoriously challenging to parallelize effectively [9].

4   Discussion: What drives performance?

This section considers relationships between input properties and performance of classical and quantum solvers that explain some of the observations of Section 3. For simplicity we focus on three graph structures (NAT1, 3DLAT, and SK), and on the Advantage QPU versus three classical solvers (Random, SGD, and SA), all implemented on CPUs. (The PTc solver, not shown here, was dominated by SA on NAT1 and 3DLAT inputs and by SGD on SK inputs.)

Figure 9 shows some relevant size metrics for these graphs. Given a graph with n nodes and m edges,

- **Problem size** n is the number of variables, which determines the size 2^n of the solution space to be searched, a key indicator of problem complexity.

- **Input size** $K = n + m$ is the number of components (node and edge weights) needed to fully specify the problem. This correlates with the size of a key data structure (and a lower bound on computation time) for classical solvers, and with the size of the embedded input on the QPU.

- **Chain length** $L = q/n$ is the mean ratio of problem size in logical and physical versions of each instance (although technically speaking native graphs do not contain chains). Chain length serves to quantify the indirect overhead cost of embedding that is considered in our Milestone 2 tests.

In terms of problem size, NAT1 is among the largest graphs in our study, 3DLAT is the largest embedded graph, and SK is among the smallest of embedded graphs. 3DLAT is about 15 times larger than SK in problem size, while SK is about 50% larger in input size, with 7.5 times longer chains.

Figure 10 shows how solution quality converges towards optimality with increasing computation time, with work controlled by the auto-tuning policies described in Section 2.2. The three panels are ordered bottom-up by increasing problem size. In each panel,

the y-axis marks relative error $\mathcal{R}_m$ observed in 15 independent trials at geometrically-increasing time increments t. Note that cases where the solver collects multiple samples to fill the time limit, relative error here is the *minimum* energy found in the batch, not the *median energy* as measured in Section 3. The shaded area marks the range of time limits considered in that section.

These results showing sample minimums are consistent with results for sample medians in Figure 6. On NAT1 (Milestone 1), the QPU outperforms classical solvers in all scenarios, and SA fails in scenarios where time per sample is low (here when $t/s < .006$ seconds). On 3DLAT (Milestone 2), the QPU outperforms classical solvers when $t/s < .1$ sec; SA can win when $t/s > .2$ sec, but can fail when $t/s < .02$ sec. On SK inputs (Milestone 2), QPU results are dominated by multiple classical solvers; but SA can fail when $t/s < .02$ sec.

These three inputs have spin glass weights, with fields $h_i = 0$ and edge weights selected u.a.r. from $J_{ij} \in \{-1, +1\}$; all solution energies are separated by multiples of ± 2. Results for 3DLAT and SK are shown in the logical problem space, after physical solutions from the QPU have been mapped back to the original input.

4.1 Classical Performance Drivers

Recall from Section 2.2 that the classical solvers in our tests work to improve an initial random solution by flipping one node at a time, while seeking lower-energy regions of the solution space. The work parameter w equals the number of main-loop iterations performed as a solver explores the space, which is an upper bound on the total number of states visited. We consider the relationship between work, time, and solution quality for these solvers.

First, Figure 10 illustrates the normal expectation that computation time per unit of work increases with input size K. Moving from bottom to top, the classical start-times (leftmost data points) generally shift right, reflecting increasing costs to initialize data structures, autotune parameters (if applicable), and return the first solution. For SGD and SA, both start-time and time-per-iteration are proportional to K because these operations require full traversals of the input data structure.

Second, the three classical solvers form a Pareto frontier ranging from least-time/worst-solutions to most-time/best-solutions as follows.

- Random applies no work to improve the initial solution ($w = 0$). Here, $\mathcal{R}_m$ follows the extremal statistics (minima) of random solution samples that increase in size with t. (Note that the distribution of solution energies for spin-glass inputs is symmetric around zero: therefore, minima of small random samples are near zero, which by formula (3) works out to $\mathcal{R}_m \approx 1$ in all three panels.)

- SGD does more work than Random, resulting in later start times and better-quality solutions. For any specific input, mean work per solution is constant,

Figure 10: Convergence of minimum relative error with respect to computation time, for three inputs and four solvers. These panels show results for 15 independent trials for time limits t in geometric increments. The shaded regions mark the range of test times from Section 3.

25

reflecting the average traversal distance from a random initial solution to a nearby local minimum. As with Random, the data points follow the extremal statistics of increasingly large samples from the distribution.

- The SA work parameter `num_sweeps` is set to maximize work-per-solution while respecting time limit t; increasing work allows SA to climb out of local minima and continue its downhill progress.

Compared to the barely perceptible improvement of Random and SGD in the top two panels, the SA strategy of increasing work is clearly a more effective use of the alotted time. The difference is that increasing work allows SA to drive the center of *location* of its sample distribution toward lower energies, whereas Random and SGD are stuck sampling from fixed distributions created by constant work.

Also in the top two panels, we note that the gradual flattening out of the SA convergence curve — in the sense that doubling work is much less effective at large w than at small w — is not unexpected in the context of NP-hard problems. The existence of an approximation method that achieves polynomial improvements in solution quality with only polynomial extra work in the worst case (called an FPTAS), would imply that $P = NP$ [60].

The bottom SK panel runs contrary to these expectations, since both SGD and SA converge quickly to putative optimal solutions, as indicated by agreement on the same minimum energy (at $\mathcal{R}_m = 0$) over many independent trials. In this case, SGD has no problem finding ground states among the local minima that it samples, and SA needs only a few sweeps to routinely find ground states. (Figure 6 shows that PTc and PTg also performed well on SK inputs at higher t/s.)

This result is not surprising in the sense that any NP-hard problem class can contain easy inputs that are amenable to greedy and heuristic search approaches. Although SK inputs are hard enough to pass our screening tests of Section 2.1 (needing more than 16 ms to find ground states in their sample medians), strong performance from several classical solvers suggests that the problem landscape is smooth, with relatively few obstacles to impede progress toward ground states, or that ground states are unusually common, or both. Determining whether these "easy" conditions are common to clique-like graph structures or are simply artifacts of small problem size, is an interesting topic for future study.

4.2 Quantum Performance Drivers

We now consider the relationship between work $w = t_a$ and solution quality $\mathcal{R}_m$ on the Advantage QPU, and how that relationship is affected by input properties.

Note that nothing definitive can be concluded based on the *shapes* of the three convergence curves in Figure 10, for two reasons. First, by our auto-tuning policy (Section 2.2), anneal time is $t_a = 240$ μs throughout most of the test range: these

convergence patterns are due to increasing sample size, not increasing work. Second, the data points are plotted according to access times rather than anneal times, which distorts the true energy-versus-work relationship: an anneal-time plot would show QPU start times at $t_a = 240$ μs, about 750 times lower than minimum access time (16 ms), and beyond the left edge of the plot field, thus demonstrating Milestone 0 on all three inputs.

On the other hand, the *vertical locations* of QPU data points relative to classical results are informative: QPU solution energies are dominated by both SGD and SA in the bottom panel, intersect with SA in the middle panel, and lie strictly below the Pareto frontier in the top panel. This reverse trend, by which QPU solution energies *improve* relative to classical energies on increasingly large inputs and under fixed time limits, outright contradicts any assumption that quantum computation time must necessarily increase with problem size to ensure competitive results.

This divergence illustrates a fundamental distinction between quantum and classical models of computation. As a popular quantum computing metaphor explains it: a classical solver can represent one of 2^n solution states with a register R of n bits that are updated incrementally as the solver moves from state to state. In contrast, the quantum solver can operate on a register Q of n qubits, exploiting superposition and entanglement to probabilistically represent all 2^n states simultaneously, and manipulating Q so that the most desirable states are most likely to be observed at the end of the computation. Since the quantum computation does not visit states one-by-one, the quality of solutions it finds does not depend on how many states might be visited in a given time limit.

Indeed, since the QPU always operates on the full Pegasus graph of size $q + c$, irrespective of how many qubits and couplers are used to represent an input of size $n+m$, there is no direct structural mechanism (such as a data structure being traversed) that would require quantum computation time to vary with logical problem size.

Instead, models from statistical physics tell us that outputs from quantum annealing computations can be approximately characterized by a Boltzmann distribution with effective temperature τ; lower effective temperature is associated with distributions returning better-quality solutions. Effective temperature depends on three factors:

- Physical properties of the QPU (such as environmental noise and fidelity of analog controls), which do not vary with input [4, 7, 47, 52, 65];

- Annealing parameters (such as anneal time and location of pauses in the anneal path), which can have significant affect on solution quality [15, 40, 63], but (with two exceptions discussed below) were not varied in our tests.

- Properties of the physical inputs that determine landscape structure, such as the range of weights (h, J) [29, 32, 36, 50, 58].

Recall that by our auto-tuning policy, all annealing and input parameters were held fixed, with two exceptions: during each (re)programming step, a random spin-reversal

transform (an annealing parameter) was applied to native inputs; and a random increment/decrement of chain strength (an input property) was applied to embedded inputs. Each such randomization can shift the location τ of the Boltzmann distribution incrementally higher or lower: thus, randomizing over multiple programmings has the effect of spreading the aggregate distribution over a wider range. This explains the gradual downward trend in minimum relative error $\mathcal{R}_m$ with larger samples at increasing time limits, even though work is constant; however it does not explain the divergence in performance relative to classical solvers.

Strong performance in our Milestone 1 tests suggests that whether a physical input is native or embedded, τ is consistently low enough for the QPU to outperform classical solvers on all inputs tested.

We conclude that the observed pattern of quantum outcomes in Figure 10 is due to a mechanism exposed by the unembedding step, when physical solutions are mapped back to logical solutions and energies are calculated in the original problem space. Specifically, we conjecture that quantum solution quality is negatively correlated with chain length $L = q/n$: chain length grows inversely with n in our test design because embeddings range from large-and-sparse (with short chains) to small-and-dense (with long chains). We have observed, for example, that longer chains require stronger (larger magnitude) chain weights $-J_{chain}$, which could compress the logical problem scale in such a way that logical solution quality degrades relative to physical solution quality. However, this proposed mechanism does not correlate perfectly with observedperformance in general; more work is needed to fully characterize and understand this phenomenon.

Finally, we note that this unembedding penalty is not necessarily prohibitively damaging to logical solution quality. The bottom SK panel shows a clear horizontal point stratification that correspond to the spectrum of optimal and nearest-optimal solution energies for this input. Although the QPU did not find ground states in this particular test, it found several solutions within fifth- or sixth-best overall. Relative performance on SK inputs may be more due to classical "easiness" due to properties of small cliques, than of quantum "hardness" due to long chains.

4.3   Test Scope

The specific numerical results reported in Sections 3 and 4 should not be considered definitive, since they depend on the autotuning policies applied under our Fair Test procedure (Section 2.2). Our study, which focuses on robust performance over a broad variety of input types, considers parameter tuning to be type of computational overhead that our test design attempts to minimize. Development of benchmark tests that consider quantum and classical solver efficiency on specific problem domains, under comparable levels of tuning effort, would be an interesting direction for future research.

We believe, however, that the observed general patterns of relative performance and convergence, and our discussion of mechanisms driving those patterns, can be extended

to other general-purpose heuristic methods that similarly do not exploit input-specific assumptions. We believe this category of heuristics is the most appropriate comparison group for our envisioned use case, which requires robust performance from general-purpose BQM solvers deployed over the cloud.

As discussed in Section 2.1, some input classes were not included in our main tests, because instances small enough to be embeddable on the Advantage QPU were too small (and therefore too easy) for tests of Milestone 2 to be viable. Inputs from other application domains, such as problems with global constraints, which carry higher translation and embedding overhead costs than those studied here, could not be included for similar reasons. Although such "too easy" inputs will always exist, we believe they are not likely to arise in real-world applications, and *ipso facto* not interesting candidates for demonstrations of quantum utility.

As mentioned in the introduction, we consider Milestones 1 and 2 to be most important for demonstrating quantum utility because the other overhead times in Figure 1 may be considered negligible or irrelevant, depending on the intended use case. For example, the direct cost of minor embedding can sometimes be amortized over many runs on same-structured graphs, such as: cliques, used in wireless network applications [31]; regular lattice-like structures, used in materials simulation [34]; circuits, used in fault detection [8, 49]; and maps, used in routing problems [11, 18].

For another example, the overhead costs of network transmission may apply equally to both quantum and classical solution methods, and use cases may prioritize either network latency or throughput. For yet another, commercial users may prioritize performance metrics unrelated to computation time, such as the dollar cost of incorporating a given solution approach into an existing workflow.

In this context, application scenarios for which Milestones 1 and 2 might be considered sufficient to demonstrate quantum utility would have the following properties: (a) they use large lattice-like inputs that can be directly mapped to the hardware graph or else minor-embedded with short chains; (b) they require many independent solution samples in short time frames; and (c) additional overheads in Figure 1 are not important to the comparison. As it turns out, these criteria have been met by some research applications in quantum materials simulation, where D-Wave systems have been shown to outperform classical simulation solvers by several orders of magnitude [33].

5 Conclusions

We introduce quantum utility, an approach to quantum performance evaluation on optimization problems, that aims to capture the user experience by explicitly considering overhead costs attached to the quantum computation. We identify three milestones associated with specific categories of overheads, and present experimental results for Milestones 1 and 2, which measure performance of an Advantage QPU against seven classical solvers running on CPUs and GPUs, using a testbed of 13 input classes.

Using a rank-based performance metric that identifies solvers outcomes as "wins" or "fails", we observe that the Advantage QPU wins in 99% tests of Milestone 1 and in 19% of tests of Milestone 2. Cases where the QPU wins are generally the same cases where classical solvers most often fail, and are associated with largest problem sizes n and smallest limits on time per solution t/s. Section 4 describes some fundamental distinctions between classical and quantum performance mechanisms that explain these performance differences.

We believe that our understanding of classical and quantum performance drivers bodes well for continued progress towards the goal of demonstrating quantum utility on broad categories of problems using future-generation QPUs. This belief is based on early experience with a small prototype (500 qubits) of the Advantage2 system,[7] which suggests that full-sized QPUs (7000 qubits) will hold larger inputs with more compact embeddings (shorter chains), and will demonstrate technological improvements (better noise suppression and larger energy scale) yielding lower effective temperatures[1, 10, 43].

Furthermore, access time overheads, which have slightly decreased over the past decade as qubit counts have grown from 500+ on the D-Wave Two [44], to 5000+ on the Advantage processor, are not expected to increase significantly in the foreseeable future.

Thus, replicating our Milestone 2 tests on the same and larger inputs, under identical time constraints, should expose degraded classical performance (from traversing smaller proportions of larger search spaces), and improved QPU performance (from lower effective temperatures and shorter chains), thereby increasing the variety of inputs and test scenarios that see QPU wins and classical fails. As well, new types of input will pass our hardness test (Section 2.1) and qualify to enter the benchmarking arena. We look forward to developing new benchmark tests incorporating broader varieties of inputs and more challenging milestones using future generations of annealing-based quantum systems.

References

[1] Early Progress on Lower-Noise Fabrication Development for the Future, Full-Scale Advantage2 Quantum Computer. *D-Wave White Paper*, (09-1287A-A), June 2022.

[2] D-Wave Ocean SDK Documentation: Minorminer Embedding and QPU Topology. *docs.ocean.dwavesys.com*, accessed 2023.

[3] Abel et al. Quantum optimization of complex systems with a quantum annealer. *arXiv 2105.13945*, 2021.

[4] T. Albash et al. Temperature scaling law for quantum annealing optimizers. *Phys. Rev. Lett.*, 119(110502), 2017.

---

[7]The Advantage2 prototype is available to the public through D-Wave's Leap portal. Full sized Advantage2 systems are expected to launch in 2023 or 2024.

[5] T. Albash et al. Demonstration of a scaling advantage for a quantum annealer over simulated annealing. *Phys. Rev. X*, 8(031016), 2019.

[6] R. S. Barr et al. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1, 1995.

[7] M. Benedetti et al. Estimation of effective temperatures in quantum annealers for sampling applications: A case study with possible applications in deep learning. *Phys. Rev. A*, 94:022308, 2016.

[8] Z. Bian et al. Mapping constrained optimization problems to quantum annealing with application to fault diagnosis. *Front. ICT*, Jul. 2016.

[9] R.P. Biswas et al. *Chapter 1: Parallel computing strategies for irregular algorithms.* 2002. Available from NTRS ntrs.nasa.gov/citations/20020090950.

[10] K. Boothby et al. Zephyr Topology of D-Wave Quantum Processors. *D-Wave Technical Report*, (14-1656A-A), Sept. 2021.

[11] J. Clark et al. Towards real time multi-robot routing using quantum computing technologies. *HPC Asia 2019*, Jan. 2019.

[12] V. S. Denchev et al. What is the computational power of finite-range tunneling? *Phys. Rev. X*, (0321015), 2016.

[13] I. Dunning et al. What works best when? a systematic evaluation of heuristics for Max-Cut and QUBO. *INFORMS J. on Computation*, 30, 2018.

[14] E. Farhi et al. A quantum adiabatic evolution algorithm applied to random instances of an np-complete problem. *Science*, 292, Apr. 2001.

[15] J. Golden and D. O'Malley. Reverse annealing for nonnegative/binary matrix factorization. *PLOS ONE*, 16(e0244026), 2021.

[16] A.A. Hagberg et al. Exploring network structure, dynamics, and function using NetworkX. *Proc. 7th Python Conference (SciPy2008)*, Aug. 2008.

[17] F. Hamze and N. de Freitas. From fields to trees. *arXiv:1207.4149*, 2012.

[18] R. Harikrishnakumar et al. A quantum annealing approach for dynamic multi-depot capacitated vehicle routing problem. *arXiv*, (2005.12478), 2020.

[19] R. Harris et al. Phase transitions in a programmable quantum spin glass simulator. *Science*, 361(6398):162–165, July 2018.

[20] I. Hen et al. Probing for quantum speedup in spin glass problems with planted solutions. *Phys. Rev. A*, 92(042325), 2015.

[21] C. F. Higham et al. Testing a qubo formulation of core-periphery partitioning on a quantum annealer. *arXiv:2201.01543*, 1 2022.

[22] J. N. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1(1):33–42, 1995.

[23] D. Inoue et al. Traffic signal optimization on a square lattice with quantum annealing. *Nature Scientific Reports*, 11(2202), 2021.

[24] D. Inoue and H. Yoshida. Model predictive control for finite input systems using the D-Wave quantum annealer. *arXiv:2001.01400*, 1 2020.

[25] S Istrail. Statistical mechanics, three-dimensionality and np-completeness. *STOC*, 2002.

[26] D.S Johnson. A theoretician's guide to the experimental analysis of algorithms. In McGeoch Goldwasser, Johnson, editor, *Fifth and Sixth DIMACS Implementation Challenges*. AMS, Dec. 2001.

[27] E. B. Jones et al. On the computational viability of quantum optimization for PMU placement. *arXiv 2021.04489*, 2021.

[28] M. Jünger et al. Quantum annealing versus digital computing: an experimental comparison. *ACM Journal of Experimental Algorithmics*, 26(1.9), 12 2021.

[29] H. G. Katzgraber et al. Glassy Chimeras could be blind to quantum speedup: Designing better benchmarks for quantum annealing machines. *Phys. Rev. X*, 2014.

[30] H. G. Katzgraber et al. Seeking quantum speedup through spin glasses: The good, the bad, and the ugly. *Phys. Rev. X*, 2015.

[31] M. Kim et al. Levereging quantum annealing for large mimo processing in centralized radio access networks. *Proc. ACM SIGCOMM*, 8 2019.

[32] A. King et al. Performance of a quantum annealer on range-limited constraint satisfaction problems. Feb. 2015. arXiv 1502.02089.

[33] A. King et al. Scaling advantage over path-integral quantum monte carlo in quantum simulation of geometrically frustrated magnets. *Nature Communications*, 12(1113), 2021.

[34] A. D. King et al. Coherent quantum annealing in a programmable 2,000 qubit ising chain. *Nature Physics*, 18, 2022.

[35] J. King et al. Benchmarking a quantum annealing processor with the time to target metric. 2015. arXiv 1508.0508.

[36] J. King et al. Quantum annealing amidst local ruggedness and global frustration. *J. Phys Soc. JPN*, 88(061007), 2019.

[37] Y. Koshka and M. A. Novotny. Comparison of D-Wave quantum annealing and classical simulated annealing for local minima determination. *arXiv:1911.03338*, 11 2019.

[38] Y. Lui et al. Implementation of trained factorization machine recommendation system on quantum annealer. *arXiv:2210.12953*, Oct. 2022.

[39] S. Mandra and H. G. Katzgraber. A deceptive step toward quantum speedup detection. *Quantum Science and Technology*, 2018. iopscience.iop.org.

[40] J. Marshall et al. Power of Pausing: Advancing understanding of thermalization in experimental quantum annealers. *Phys. Rev. Applied*, (044083), Apr. 2019.

[41] C. McGeoch. *Optimization with clause problems.* D-Wave Technical Report 14-1001A-A, 2017.

[42] C. McGeoch et al. Practical annealing-based quantum computing. *IEEE Computer*, 52, June 2019.

[43] C. McGeoch et al. The D-Wave Advantage2 Prototype. *D-Wave Technical Report*, (14-1063A-A), June 2022.

[44] C. McGeoch and C. Wang. Experimental evaluation of an adiabatic quantum system for combinatorial optimization. *Proc. ACM Computing Frontiers*, (13), May 2013.

[45] C. C. McGeoch. *A Guide to Experimental Algorithmics.* Cambridge Press, 2012.

[46] S. Morita and H. Nishimori. Mathematical foundation of quantum annealing. *J. Math. Phys.*, (49), 2008.

[47] H. Nishimori. *Statistical Physics of Spin Glasses and Information Processing.* Clarendon Press, 2001.

[48] Y. Pang et al. The potential of quantum annealing for rapid structure identification. *Constraints (Special Edition on Constraint Programming, Artificial Intelligence, and Operations Research)*, 26:1–25, 2021.

[49] A. Perdomo-Ortiz et al. Fault detection and diagnosis of graph-based systems. *arXiv:1406.7601*, Oct 2014.

[50] D. Perera et al. Computational hardness of spin-glass problems with tile-planted solutions. *Phys. Rev. A*, 2020.

[51] O. M. Raissudin and S. De. Feqa: Finite element computations on quantum annealers. *arXiv 2201.09743*, 2022.

[52] J. Raymond et al. Global warming: temperature estimation in annealers. *Frontiers in ICT; Sec. Quantum Engineering and Technology*, 3, 2016.

[53] R. K. Rishi. What is the difference between log and symlog in matplotlib? *tutorialspoint.com*, 2023.

[54] T. F. Röennow et al. Defining and detecting quantum speedup. *Science*, 345(6195), June 2014.

[55] A. Selby and C. Coffrin. lanl-ansi/hfs-algorithm: Alex selby's chimera-qubo solver. *github.com/lanl-ansi/HFS-algorithm*, Accessed 2022.

[56] M. Streif et al. Beating classical heuristics for the binary paint shop problem with the quantum approximate optimization algorithm. *Phys. Rev. A*, 104(012403), July 2021.

[57] Albash T. and D. Lidar. Adiabatic quantum computation. *Rev. Mod. Phys.*, 90(015002), Jan. 2018.

[58] B. Tasseff et al. On the emerging potential of quantum annealing hardware for combinatorial optimization. *arXiv:2210.04291*, Oct. 2022.

[59] I. Trummer and C. Koch. Multiple query optimization on the d-wave 2x adiabatic quantum computer. *Proc. of the VLDB Endowment*, 19:648–659, May 2016.

[60] E. J. van Leeuwen and J. van Leeuwen. Structure of polynomial-time approximation. *Theory of Computing Systems*, 50:641–674, 2012.

[61] S. E. Venegas-Andraca et al. A cross-disciplinary introduction to quantum annealing-based algorithms. *Contemporary Physics*, 59:174–196, 2019.

[62] D. Venturelli et al. Quantum optimization of fully connected spin glasses. *Phys. Rev. X*, 5(031040), Sept. 2015.

[63] D. Venturelli and A. Kondratyev. Reverse quantum annealing approach to portfolio optimization problems. *Quant. Mach. Intell.*, 2019.

[64] S. Yarkoni et al. Multi-car pant shop optimization with quantum annealing. *IEEE IQCE*, 2021.

[65] K. C. Young et al. Adiabatic quantum optimization with the wrong Hamiltonian. *Phys. Rev. A*, 88(062314), 2013.

[66] Z. Zhu et al. Strengths and weaknesses of weak-strong cluster problems: A detailed overview of state-of-the-art classical heuristics versus quantum approaches. *Physical Review A*, 2016.