D:Wave
The Quantum Computing Company™

Optimization with Clause Problems

TECHNICAL REPORT

C. C. McGeoch, J. King, M. Mohammadi Nevisi, S. Yarkoni, and J. Hilton

2017-01-27

Overview

We introduce a new input class called *clause problems*, that can be used to study local constraint structures, which occur in inputs translated from general NP-hard problems to the D-Wave native topology. We describe a small family of clause problems that are contrived to create significant challenges for two classical competition solvers, simulated annealing (SA) and the Hamze–de Frietas–Selby solver (HFS). We identify key properties of these inputs that lead to poor performance by the classical solvers, and consider whether these properties might naturally arise in problems from real-world applications.

CONTACT

Corporate Headquarters
3033 Beta Ave
Burnaby, BC V5G 4M9
Canada
Tel. 604-630-1428

US Office
2650 E Bayshore Rd
Palo Alto, CA 94303

Email: info@dwavesys.com

www.dwavesys.com

14-1001A-A
D-Wave Technical Report Series

Notice and Disclaimer

Summary

- We describe a new input class called *clause problems* that capture *local constraint structures*. These structures are commonly found in native (Chimera-structured Ising model) inputs that are mapped from general NP-hard optimization problems. While clauses do not capture the full complexity of these logical problems, they can be used to isolate certain component structures for in-depth study.
- We show how to construct simple clause problems that create significant performance challenges for two well-studied classical competitors, simulated annealing (SA) and the Hamze–de Frietas–Selby solver (HFS). SA is implemented on a 2048-core graphics processing unit (GPU) platform, and runs about 500 times faster than a comparable implementation running on a standard CPU.
- Experimental results show that a recent-model D-Wave quantum processing unit (QPU) can solve these problems up to 16,000 times faster (total time) and 625,000 times faster (anneal time) than *both* SA and HFS. (Total time includes I/O time to transfer information to and from the QPU chip; anneal time refers to pure computation time.)
- Multiplying SA GPU times by 500 gives an estimated anneal time speedup of $3 \cdot 10^8$ over a comparable CPU version, which is the same order of magnitude as that observed by Denchev et al. in ''What is the computational value of finite range tunneling?'', *Phys. Rev. X*, 2015.
- These clause inputs are specially designed to have certain properties that expose poor performance of SA and HFS. We describe these properties and argue that similar outcomes might be observed when real-world problems (of sufficiently large size) are transformed for solution on D-Wave QPUs.

Contents

1 Introduction

A D-Wave computing system implements a family of quantum annealing algorithms in hardware on a quantum processing unit (QPU) to heuristically solve instances of the NP-hard Ising model (IM) optimization problem. The QPU exploits quantum properties such as superposition, entanglement, and tunneling, to carry out its task (for details see [1–7]).

Qubit connectivity in the QPU is described by G, a subgraph of a Chimera graph, which is a $k \times k$ grid of 8-qubit cells (see Section 3 for more). A Chimera graph of size k containing $8k^2$ qubits is denoted C_k. Several classical optimization heuristics have been implemented that are specialized to the Chimera topology in the sense that they only read inputs defined on G. These so-called *native solvers* exploit certain properties of the Chimera graph such as bipartiteness and low degree, and are generally accepted as the fastest classical approaches known for solving native inputs defined directly on G.

Our goals in this report are threefold: (1) to introduce a new family of inputs called *clause problems* that can be used to study local constraint structures, which appear in general problems that are mapped to the Chimera topology; (2) to present a simple demonstration of the No Free Lunch principle by showing that simple input structures can create significant performance challenges for state-of-the-art native classical solvers; and (3) to identify the key properties that make these inputs hard for these classical solvers to solve.

Clause problems, described in Section 3, are motivated by several recent papers describing use of previous generation D-Wave systems to solve general NP-hard problems, including circuit fault diagnosis [8], database query optimization [9], and job shop scheduling [10]. In this work a *logical input* formulated for the original problem is transformed to IM and then mapped to G to create a native input for direct solution on the QPU, a process that may expand input size.[1]

The logical problems described in these papers contain *local constraints* that define problem restrictions involving small subsets of variables. A constraint is analogous to a clause in a general Boolean formula, which describes a logical relationship among a small number of variables, and which must be satisfied as part of the global solution. In these papers the constraints are translated and mapped as distinct input components, creating a pattern of small regular subgraphs in G; various strategies (such as chain embeddings or place-and-route schemes [8]) may be used to connect the representations of single variables appearing in multiple subgraphs.

Clause problems can be used to investigate properties of inputs built from small regular structures representing such constraints. The problems studied here are built from grids of C_2 and C_3 clauses, with limited interactions between grid neighbors: note that unlike native inputs derived from logical problems, these problems do not contain any global structures to tie one clause to another. Since they do not contain global interactions (as discussed in King et al. [11]), no claim is made that these inputs have combinatorial complexity that scales with problem size.

[1] To date, performance analyses on these problems have returned mixed results, sometimes showing quantum runtime speedups by factors of 1000 [9] or 25 [8] against classical solvers working on the logical problem, and sometimes showing little difference. In some cases the latter result may be due to limited problem sizes required by early generation QPUs: significant performance differences cannot be observed when inputs are small and all solution methods take time on the scale of a few milliseconds.

Nevertheless, these seemingly trivial native problems can be extremely challenging for some native solvers to solve. We consider two well-studied solvers—simulated annealing (SA), and Selby's implementation of the Hamze-de Freitas-Selby algorithm (HFS)—in comparison to a recent-model 2000-qubit QPU. The SA solver runs on an NVIDIA 980 platform with 2048 GPU cores, and is roughly 500 times faster than a comparable CPU-based implementation. HFS runs single-threaded on a standard Intel CPU core. Section 2 contains details about these solvers and platforms.

We look at performance on two different clause types that are adaptations and generalizations of the weak-strong cell clusters of Boixo et al. [12] and Denchev et al. [13]. Weak-strong clusters were designed to demonstrate that D-Wave QPUs are capable of finite-range multiqubit tunneling, a property of quantum computation that is not available to classical algorithms. Denchev et al. observe that the "computational value" of quantum tunneling can be substantial, yielding runtime speedups by factors up to 10^8 against two native solvers, SA and quantum Monte Carlo (QMC). However, as they point out and Mandrá et al. [14] later verify, other classical approaches solve these inputs much more efficiently—in particular the HFS solver studied here.

Our experimental results in Section 4 show that these clause problems are challenging for *both* SA and HFS to solve. We do not have adequate computing resources to demonstrate runtime speedups on the scale of 10^8 as in [13], which would require several CPU-months of classical computation time.[2] Instead we report results of smaller-scale tests with 200-second timeout limits placed on the classical solvers. Our tests at largest problem sizes revealed quantum runtime speedups by factors as high as 16,000 (total time) and 625,000 (anneal time) over HFS and SA. (Total time includes I/O overhead to move information on and off the QPU; anneal time refers to pure computation time.) Multiplying SA GPU times by 500, this translates to an anneal-time speedup of about 3×10^8 over a CPU implementation of SA, within the same order of magnitude as the runtime speedup observed in [13]. These speedups correspond to cases where the classical solvers ran to timeout without finding solutions of comparable quality to those found by the D-Wave QPU within a few milliseconds: larger experiments with longer timeouts would have revealed even greater performance gaps.

The performance metric used here is similar to the TTT metric described by King et al. [15], which measures time to find good-quality solutions in scenarios where optimality is not guaranteed; see Section 4 for details. As in [15], we focus on runtime comparisons and do not closely analyze scaling performance (as defined in Rønnow [16]), for several reasons discussed in Section 2, including the fact that scaling of the classical solvers cannot be observed at higher problem sizes due to timeouts.

This work provides a simple empirical demonstration of the No Free Lunch (NFL) principle, based on a theorem by Wolpert and Macready ([17–19]). For technical reasons the theorem does not apply to the three solvers studied here,[3] but a consequence of the theorem is relevant: apart from its intrinsic complexity, the perceived hardness of any given input instance depends on how well its structural properties "align with" [19] the partic-

[2] Overall our tests required a few weeks of computation time, including pilot tests to tune experimental parameters.

[3] The theorem applies to classical algorithms that are complete, and either deterministic or derandomized (it would not be difficult to adjust these assumptions to cover SA and HFS). It also assumes that algorithms examine solutions one-by-one in sequential order, whereas quantum annealing processors are capable of examining multiple solutions simultaneously through the use of superposition states. It is not known whether an analogous NFL theorem holds for quantum computing.

ular classical algorithm(s) used to solve it. Section 4 describes how the structures of SA and HFS are misaligned with certain key properties shared by weak-strong clusters and the clause problems studied here.

A primary mission of empirical research in heuristics is to understand those relationships between input properties and algorithm structures, both in order to build better algorithms and to make recommendations to the practitioner as to which solution method is best for which types of input. We do not expect that generated inputs such as the clause problems studied here (or generated native inputs studied elsewhere) are likely to appear in practical scenarios. However it is useful to consider whether the underlying properties that make these problems hard to solve might arise naturally in application inputs. Properties that are easy to identify provide a basis for making recommendations about solution methods; properties that are hard to identify motivate interest in *robust* solvers that perform well (best worst-case rather than best best-case) over broad classes of inputs. Section 5 briefly considers whether the results described here might extend to other native solvers and more general input classes.

2 Solvers and platforms

This section presents an overview of the solvers tested here and discusses tuning and timing considerations.

The D-Wave system. A D-Wave QPU heuristically solves instances of the Ising model (IM) problem: Given input weights (h, J) on the nodes and edges of a working graph $G = (V, E)$, assign *spins* $s_i = \pm 1$ (sometimes denoted $s_i \in \{\uparrow, \downarrow\}$ for convenience) to nodes so as to minimize the *energy function*

$$\mathcal{E}(s) = \sum_{i \in V} h_i s_i + \sum_{(i,j) \in E} J_{ij} s_i s_j. \tag{1}$$

The J values are called *couplings*, which may be *ferromagnetic* (negative) or *antiferromagnetic* (positive). The h values are called *fields*. A spin assignment that minimizes $\mathcal{E}(s)$ is a *ground state*, and non-optimal assignments are *excited states*. If an individual term $h_i s_i$ or $J_{ij} s_i s_j$ has positive sign (i.e., is not minimized) in a ground state solution, we say that the corresponding field or coupling is *frustrated*. The problem is NP-hard, and trivial transformations exist for quadratic unconstrained binary optimization (QUBO), and weighted maximum 2-satisfiability.

The QPU contains a physical representation of the nodes and edges of G using qubits and couplers. Solutions are obtained by the following process: (1) Program the QPU by assigning weights to qubits and couplers according to input (h, J); (2) Repeat an anneal-readout process R times to return R solutions, also called *samples*. Specifications for the particular QPU studied here appear in the table below. See Bunyk et al. [3] and Johnson et al. [5] for more about the quantum annealing process.

Size	Qubits in G	$t_{program}$	t_{anneal}	t_{read}
C_{16}	2033	9.4 ms	≥ 5 μs	123 μs

Tuning for better QA performance. Time performance on a D-Wave QPU, as well as on classical comparators, is typically evaluated in terms of the expected number of samples needed to observe a solution of some given target quality; better quality samples yield runtime speedups by reducing R. Several techniques are known for improving raw sample quality; some have been implemented as utilities in recent D-Wave systems, to allow the system to be tuned for better performance on some types of inputs. These include:

- Preprocessing techniques modify the inputs presented to the QPU. Our tests incorporate a preprocessing utility called *spin-reversal transforms*, which applies single-spin reversals (flipping the signs of associated fields and couplings) to a random subset of qubits in G. The resulting perturbed problem may avoid small systematic biases in the physical qubits and couplings, in effect averaging-out certain types of errors.
- Anneal time t_a can be specified by the user (see below). In principle, longer anneal times return better solutions, but in practice the correlation between t_a and solution quality can be quite small. Furthermore, the D-Wave systems available in 2017 provide a utility that allows users to modify the QPU annealing path by specifying perturbations of the quantum annealing algorithm used to solve the problem. We did not explore this interesting new capability.
- Two postprocessing utilities, for *optimization* (reducing individual solution costs) and for *sampling* (increasing variety in the solution set) are available. Pilot tests revealed that postprocessing was not needed to improve raw QPU performance, so these options were not included in our main set of experiments.

Based on a pilot study evaluating parameter combinations with $t_{anneal} \in \{5, 10, 20, 50, 100\}$ and $g \in \{1, 2, 5, 10\}$ spin-reversal transforms per 1000 samples, the pair $(t_{anneal} = 20, g = 1)$ showed best performance and was used in the main experiments. Overall, QPU performance on these inputs appears to be fairly robust to such parameter changes.

A computation taking R samples with $g = \lceil R/1000 \rceil$ spin-reversal transforms has total running time

$$T = g \cdot t_{program} + R \cdot (t_{anneal} + t_{read}).$$

We define sampling time as $t_{sampling} = t_{anneal} + t_{read}$.

Simulated annealing (SA). Our GPU implementation of simulated annealing follows the well-tuned native (CPU) solver of Isakov et al. [20], which is structured as follows.

1. An outer *sampling* loop returns R independent solutions.
2. A middle *anneal* loop carries out the simulated annealing process.
3. An inner loop performs a *sweep* of the n nodes of G. (This loop is not standard to simulated annealing, but is common in native solvers.)
4. A code block in the inner loop probabilistically modifies the state of each node.

Our GPU implementation parallelizes the first and third loops of this algorithm: note that the second anneal loop is inherently sequential (not parallelizable) because solution states are modified incrementally according to the loop parameters.

The outer sampling loop is easy to parallelize because it has low communication overhead, requiring negligible time to ship the problem to the other cores and receive results.[4] Parallelization yields M-fold relative speedup,[5] where M is the *minimum* of R and the number of available cores.

Successful parallelization of the inner sweep loop requires careful assignment of sections of G to individual cores, to balance local work against the relatively high cost of global memory access. Our SA code runs on an NVIDIA GeForce GTX 980 GPU platform with 2048 cores; our implementation finds good balance by assigning each core to a single Chimera cell (eight nodes of G), giving a relative speedup of $256 = 2048/8$ on C_{16} problems. The code achieves up to 60 code block executions per nanosecond, compared to about 0.15 per nanosecond with single-threaded code running on an Intel CPU (described below). This yields an absolute parallel speedup of approximately 500 on C_{16} problems, assuming $R \geq 8 = 2048/256$.

This approach spreads a problem of size C_k across k^2 cores, and uses the remaining core capacity to parallelize the outer-loop collection of samples. Note that on a fixed-size GPU, parallel speedup from sweeping *increases* and parallel speedup from sampling *decreases* with increasing problem size k. For example, a C_8 problem uses just $64 = 8 \times 8$ cores per anneal, so that samples are collected at a rate of $32 = 2048/64$ anneals per fixed time unit. But a C_{16} problem uses 265 cores per anneal, so that the sampling rate is just $8 = 2048/256$ anneals per time unit. As a result, depending on how R grows with k, GPU times *scale differently* (typically worse) than single-core CPU times.[6] CPU time is a more faithful representation of total work performed by the algorithm and would be the preferred measurement in a study of how algorithm performance scales with problem size; however, that option is unavailable to us due to prohibitively high computation times (on the scale of months) for the CPU implementation.

This solver has four parameters, *betaStart*, *betaEnd*, *annealPath*, and *numSweeps*. Pilot tests were run on $CR_{2,12}$ problems (see Section 3 for description) using a full factorial design on $betaStart \in \{1, .01, .001\}$, $betaEnd \in \{1, 3, 5, 10, 15, 20\}$, $annealPath \in \{linear, geometric\}$ and $numSweeps \in \{10^4, 5 \times 10^5, 10^5, 5 \times 10^5, 10^6\}$, and the six best combinations were tested again on $CR_{2,16}$. The best combination ($betaStart = .001$, $betaEnd = 1$, $annealPath = linear$) was used in the main tests, which took $numSweeps \in \{10^4, 10^6\}$.

Hamze-de Freitas-Selby (HFS). We use Selby's implementation of the Hamze-de Freitas-Selby algorithm (HFS) ([21], [22], [23]), modified to report each stopping state that it finds, to simplify comparison to other sampling-based approaches.

Given an input defined on G and an initial random solution s, this algorithm iterates to improve s by optimizing low-treewidth spanning subgraphs of G. At each iteration a new

[4] We note that, compared to anneal times in fractions of seconds, communication overhead moves from negligible to dominant if this code is deployed in massively parallel cluster machines or large networks. We also note that, in principle, speedups from parallel sampling are available to D-Wave platforms as well, by mapping multiple small problems onto G and/or using networked QPUs. Of course, D-Wave QPUs are currently less ubiquitous than Intel cores.

[5] *Relative* parallel speedup compares time on P parallel cores to time on a single core of the same type. *Absolute* parallel speedup compares time on P parallel (e.g., GPU) cores to the fastest known CPU implementation, which may solve the problem an entirely different way.

[6] GPU times also scale differently from those of the hypothetical classical annealer defined by Rønnow et al. [16], which (assuming one qubit equates to one core) corresponds to perfect absolute parallel speedup of the inner loop but no parallel speedup of the outer loop.

subgraph is selected at random from a fixed pool, an optimal solution is found for the subgraph, and if the new solution improves s it is adopted. The process stops when an internal stopping criteria is met. A *strategy* parameter p determines the size and complexity of subgraphs in the pool. Higher p corresponds to bigger and more complex subgraphs, which take more time per iteration to optimize, but may require fewer iterations or fewer random samples overall. The two strategies considered here—denoted HFS3 and HFS4 for $p \in [3, 4]$—are described further in the next section.

HFS cannot be implemented efficiently on a GPU, due to prohibitively high memory access costs in the subgraph optimization step. Our tests run this solver on an Intel Xeon E5-2643v3 CPU with 3.4 GHz clock, measuring wall clock time for a single computation thread running on a single unloaded core. As with the other solvers, parallelization of the outer sampling loop is straightforward, but limited by R and the number of parallel cores available.

A note about runtimes and scaling analysis. The computation time for each solver depends on the combination of R—the number of solutions in a given sample—with measured times of algorithm components, calculated as follows for QPU total time (T_{DWT}), QPU anneal time (T_{DWA}), and the computation time for the two classical solvers $S \in \{\text{SA}, \text{HFS}\}$:

$$\begin{aligned} T_{DWT} &= t_{program} + R \cdot t_{sample}, \\ T_{DWA} &= R \cdot t_{anneal}, \\ T_S &= t_{initialize} + R \cdot t_{sample}. \end{aligned} \tag{2}$$

For T_{DWT}, we have $t_{sample} = t_{anneal} + t_{readout}$. For SA, t_{sample} is proportional to $(numSweeps \cdot n)$ for a problem on n variables. For HFS, t_{sample} can vary depending on problem size and problem type, because the number of subgraph-optimization iterations for each output depends on an internal stopping criteria that is not controlled by the user; the time per optimization step also depends on the strategy parameter p and problem size n.

Figure 1 shows computation times for these solvers, as a function of R, measured on a typical $CR_{2,16}$ clause problem (described in Section 4) containing 2033 variables. The inset shows a close-up of the lower left corner of the chart, corresponding to $R \leq 4$ and runtimes below 12,000 μs.

The solid and dotted blue lines (inset) labelled DW show QPU total times and anneal times, respectively; total time is dominated by programming time, equal to 9ms on the QPU tested here; the two lines have slightly different slopes corresponding to sampling time of 143 μs and anneal time of 20 μs, respectively. The red and green lines show that HFS3 and HFS4 times are dominated by initialization costs, in this case near 210 ms. The two strategies scale differently due to different costs of subgraph optimization. Finally, the yellow and purple lines show that SA has negligible initialization overhead, but significantly different scaling under the two parameterizations $numSweeps = 10^4$ (purple) and $numSweeps = 10^6$ (yellow).

The striking differences in magnitudes and slopes in these cost profiles create some subtleties when reasoning about runtimes and problem complexity. As a general rule, runtimes grow with R, so increasing runtimes represent increasing levels of algorithm "work" in response to increasing problem difficulty. However, the figure shows that individual solver runtimes track R in very different ways; this is especially true for HFS times and

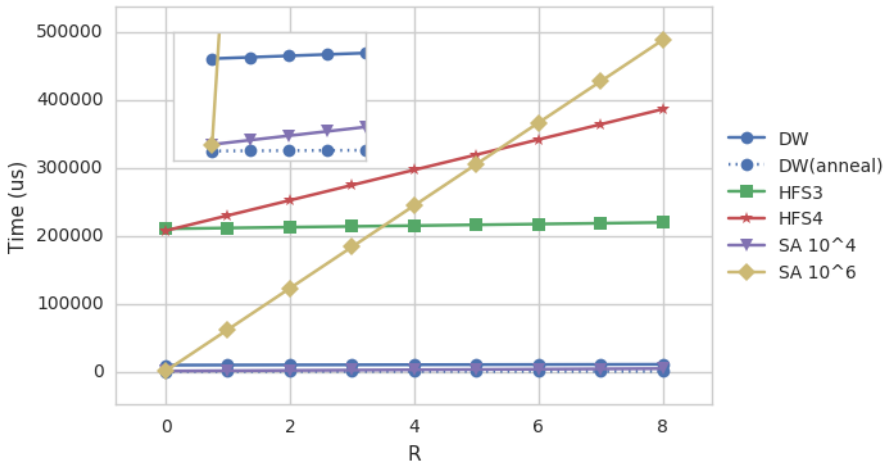

Figure 1: Typical computation times (microseconds) per number of solutions R returned, on a $C_{2,16}$ problem with 2033 variables. The inset shows a closeup of the lower left corner of the chart, with $R \leq 4$ and $Time \leq 12{,}000$ μs. DW denotes the D-Wave QPU.

DW total times, which are dominated by relatively high initialization and programming costs, respectively. As well, differences in time-per-sample create different perceptions of scaling in R. For example, doubling the number of samples from $R = 2$ to $R = 4$ approximately doubles the runtime of SA, but changes HFS4 runtime by only 20 percent. For another example, increasing R by two orders of magnitude, from $R = 1$ to $R = 100$, increases QPU total times by less than 25 percent. Solver-to-solver differences in runtime scaling with R do not necessarily represent corresponding differences in algorithm effort.

These subtleties are compounded when evaluating how algorithm performance scales with problem size, since software component times increase, while QPU component times (t_a and t_r) are constant in problem size. Also, as discussed previously, SA runtimes scale differently on CPU and GPU platforms, giving different performance profiles for the same amount of total work. Furthermore, the observed performance of any given D-Wave platform can be dominated by properties of its classical control system, which are in a state of rapid development, so that scaling curves cannot be reliably extrapolated to larger systems. We also note that the lack of global structure in clause problems means that we cannot claim that problem complexity scales exponentially with problem size.

For these reasons, in what follows we do not attempt to draw strong conclusions about problem complexity based on how runtimes scale, except to note that performance gaps between the quantum and classical solvers tend to grow with problem size.

3 Clause problems

Referring to input weights (h, J) used in the IM energy function (1), Figure 2 shows a CR_2 *ring clause* built on a 2×2 grid of four Chimera cells. All edges are ferromagnetic with $J = -1$, forming a 4-ply ring around the cells. Two nodes in the northwest corner have $h_{strong} = -0.9$ and two nodes in the southeast corner have $h_{weak} = +0.4$; all other nodes

have $h_0 = 0$.

In terms of logical problem constraints, this ferromagnetic loop structure expresses an *equality constraint* on some number $v = 1 \ldots 32$ of variables (represented by the nodes), by requiring that they all have equivalent values, either $s = \uparrow$ or $s = \downarrow$, in the optimal (clause) solution. More complex relationships among the variables can be expressed, for example, by flipping the signs of all Js associated with some variables. Clause size may grow or shrink, according to how many variables are involved and how much "surface area" is needed to connect variables among different clauses and structures.

The addition of nonzero fields h_{strong} and h_{weak} turns this into a *majority constraint* (sometimes called a voting circuit) so that in the ground state, all variables have the spin assignment preferred by the majority. In constraints deriving from application problems the fields would likely be replaced by connections to variables or structures that are external to the clause, but here for convenience we use local fields to create votes.

Generalizing this basic idea, the next section (Figure 7) describes a ($K_{8,8}$) *bipartite* clause that divides nodes into two groups A and B, with the constraint that all nodes $a \in A$ must have different values from all nodes $b \in B$. Other types of local constraints can be expressed in this way; for example a C_2 can represent a complete (K_4) graph. Bian et al. [8] give several examples mapping boolean expressions (i.e., clauses) defined on two and three variables to small groups of Chimera cells; Trummer and Koch [9] describe L-shaped TRIAD clauses of varying sizes; and Venturelli et al. [10] describe mappings of constraints in a job-shop scheduling problem. Full exploration of the myriad possibilities for representing local constraints by clauses, and evaluating the efficacy of various representations, is an interesting problem for future research.

Describing Figure 2 in IM terms, by the energy function (1), the ground state $S_0 = \uparrow\uparrow \ldots \uparrow\uparrow$ has all 32 spins aligned; all couplings are unfrustrated, and weak fields are frustrated. The first excited state $S_1 = \downarrow\downarrow \ldots \downarrow\downarrow$ also has all spins aligned, and all couplings unfrustrated, but strong fields are frustrated. Other excited states have frustrated couplings, but local minima may be found whenever all the spins in individual cells are aligned.

Note that a weak-strong cluster as in [13] comprises a pair of cells (e.g., the top half of a CR_2), with weak and strong fields on each. The cells in that paper have $h_{strong} = -1$ and $0 \leq h_{weak} \leq 0.44$, assigned to all eight nodes per cell. Couplings are $J = -1$ throughout.

The right side of Figure 2 shows one possible path in the energy landscape of the CR_2, walking from S_0 on the left to S_1 on the right, moving clockwise from the northwest corner of the clause and flipping one spin per step; this landscape is discussed in the next section.

To form a full sized $CR_{2,c}$ problem, CR_2 clauses are placed in a grid on a larger $c \times c$ graph, with the positions (opposite diagonals) and signs of the strong and weak fields chosen at random in each clause. Neighboring clauses are connected with weak couplings ($J_{connect} = \pm.2$), with alternating signs across and down. This connection strategy, chosen for simplicity, has a minimal effect on the energy landscapes of individual clauses, since adjacent pairs of connectors cancel each other out when spins are aligned within cells (some alternative connection strategies are discussed in Section 5). Note that the working graph G has a small number of nonworking qubits (15 of 2048); missing qubits and couplings alter the energy spectrum and possibly the ground states in some clauses.

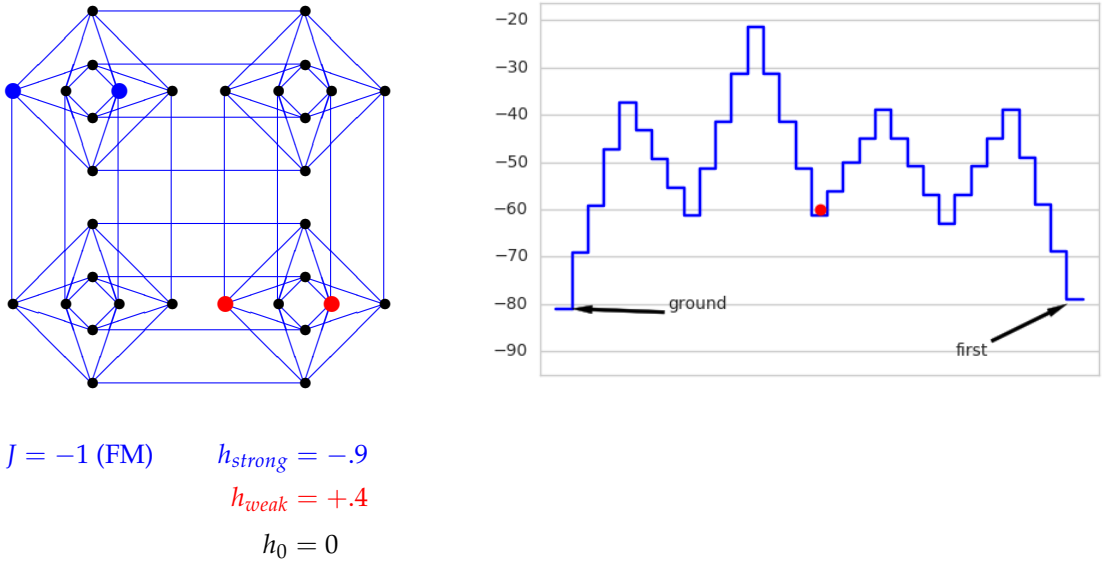

Figure 2: (left) A CR_2 ring clause. All couplings are ferromagnetic; there are two strong fields in the northwest corner and two weak fields in the southeast corner. (right) A path in the energy landscape, walking from the first excited state $S_1 = \downarrow\downarrow \cdots \downarrow\downarrow$, to ground state $S_0 = \uparrow\uparrow \cdots \uparrow\uparrow$, flipping one spin per step. The red dot is in a local minima where each cell has all eight spins aligned and some couplings between cells are frustrated.

4 Experimental results

This section describes performance comparisons of SA, HFS, and the QPU on different clause problems, starting with the $CR_{2,c}$ problems described in the previous section.

The performance metric used here is *time-to-best* (TTB), calculated as follows. Let B denote the *target energy*—the minimum energy found among pooled solution samples from all solvers, running within a fixed time limit T^*. For solver S with sample $\mathcal{S}^*$, let π_S denote the proportion of solutions in $\mathcal{S}^*$ that have target energy B. Then $R^*_S = 1/\pi$ is the expected number of samples needed for S to return a solution with energy B. TTB is calculated by inserting R^*_S into the time formula (2) for each solver. In these tests the software solvers ran to timeout limits of 200 seconds. The QPU was given a 20-second (total time) limit but typically needed less than 100 milliseconds to find target solutions. (Although we did not systematically check that the QPU reached ground state in every case, comparison to HFS4 results in the first experiment, and spot-checks on intact clauses, suggest that ground states were typically found.)

4.1 $CR_{2,c}$ ring problems

The first experiment considers performance on random $CR_{2,c}$ problems for $c \in \{8, 12, 16\}$, corresponding to $n = 507, 1144, 2033$ variables, approximately doubling at each increment.

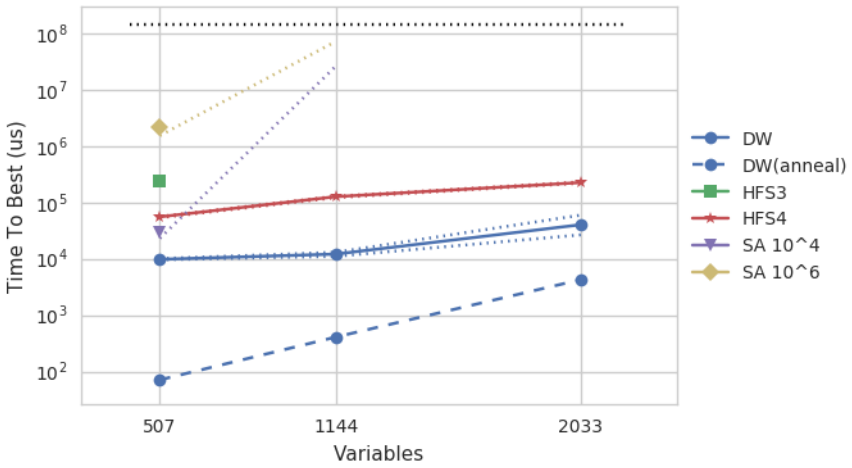

Figure 3: Median TTB in microseconds, on $CR_{2,c}$ problems for $c \in \{8, 12, 16\}$. Solid and dashed lines denote median times, dotted lines denote minimum and maximum observed times. The dotted line at top shows 200-second timeout limits for software solvers: missing data points correspond to solvers that reached timeout (on all inputs) without finding solutions of target quality B. DW denotes the D-Wave QPU.

This test includes the D-Wave QPU, the two parameterizations HFS3 and HFS4, and two versions of SA with $numSweeps \in \{10^4, 10^6\}$.

Figure 3 shows TTB for each solver at each problem size, measured over 15 random problems at each size. Solid and dashed lines show median TTB, and dotted lines show the minimum and maximum TTB values recorded at each point. Missing data points and lines (HFS3 and SA at $n \geq 1144$) correspond to cases where the solver timed out after 200 seconds without finding solutions of target quality B that were found by the other solvers. The 200-second timeout limit is indicated by the horizontal dotted line at the top.

The blue solid and dashed lines show median QPU total and anneal times, respectively; blue dotted lines around median total times show that the range of observed data was extremely small. Yellow and purple markers show results for SA with $numSweeps = 10^4$ and $numSweeps = 10^6$ respectively, at $n = 507$. On larger inputs these solvers timed-out in the median case, although some problems of size $n = 1144$ were solved within the 200-second limit, as indicated by the dotted minimum curves; the solvers timed out on all inputs at $n = 2033$.

The green and red markers show that the two parameterizations of HFS display qualitatively different performance on these problems. HFS3 fails on all $C_{2,c}$ problems for $c \geq 12$, but HFS4 does very well, with runtimes corresponding to $R^*_{HFS4} \leq 2$ throughout the range of problem sizes. This is very close to the lower bound on HFS4 runtime imposed by the initialization step, as shown in Figure 1.

Performance of SA and the QPU. The relative performance of SA and the QPU on these problems roughly corresponds to those observed on the weak-strong cluster problems of Denchev et al. [13], even though the tests involve different metrics and different platforms (GPU versus CPU) for SA.

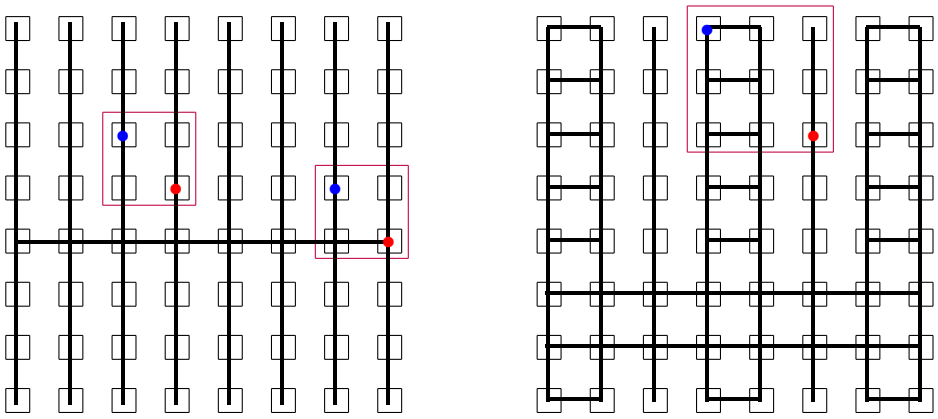

Figure 4: The boxes represent Chimera cells in a C_8 graph. The thick black lines show connectivity of spanning subtrees used by HFS. The left is an HFS3 subgraph with eight vertical branches and one horizontal crossbar, each one cell wide. The HFS4 subgraph on the right has branches and a crossbar that are one and two cells wide. The purple boxes show how these subgraphs intersect with C_2 (left) and C_3 clauses (right).

This correspondence is explained by the fact that the ring clause and the weak-strong cluster share two key properties that drive performance of these solvers. The first property is the presence of weak and strong fields of opposite signs in different cells; the second is the use of blocks of ferromagnetic couplings within and between cells. Consider again the example landscape on the right of Figure 2. The weak fields (probabilistically) lure SA into some local minimum early in the anneal process, in which both the weak cell and the strong cell have internally-aligned spins but some couplings are frustrated, as indicated by the red dot. The couplings create a rugged landscape with high hills (in terms of the energy function) that are difficult for SA to traverse as it performs a probabilistically biased random walk to search for the ground state.

The QPU, when solving these problems, starts with a flat landscape and a quantum superposition of states. As the anneal progresses and hills grow, the QPU must use a combination of quantum tunneling and thermal excitations to move towards the evolving ground state. Compared to weak-strong clusters, a C_2 clause has greater Hamming distance between the first excited state and ground state (32 versus 8 bits), and more hills and valleys between the two (four hills versus one). Nevertheless the QPU is successful at negotiating this more complex landscape. See [13] for more about the mechanism of tunneling.

Additional tests not shown here indicate that this performance difference is fairly robust with respect to changes in number and magnitudes of (h, J) as long as basic inequalities (such as $|\sum h_{strong}| > |\sum h_{weak}|$ and $|\sum h_{strong}| < |8J|$) preserve the energy spectrum. SA finds these problems easier to solve if the magnitude of J is reduced, which lowers the height of hills in the landscape, making them easier to traverse. The QPU shows better performance if key inequalities are well-separated, so that (classical) analog control errors in the processor do not introduce spurious ground states.

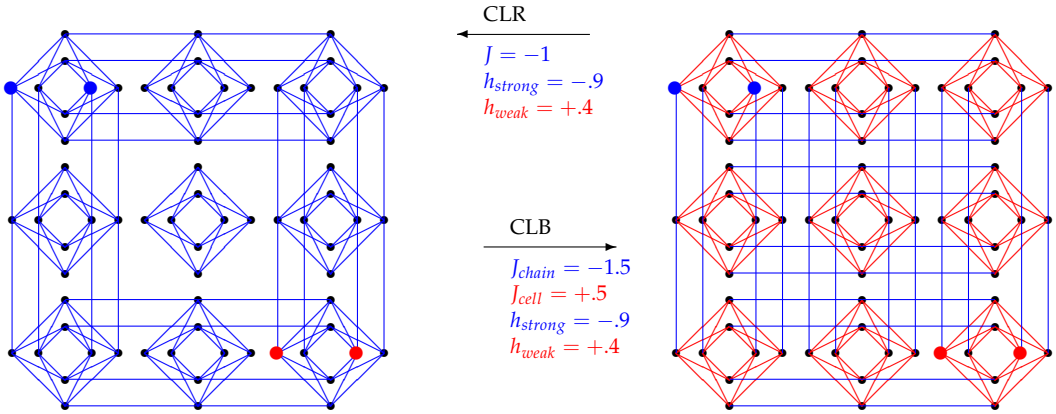

Figure 5: (left) A ring clause built on a C_3. (right) A bipartite clause built on a C_3.

Performance of HFS. The metaphor of solution landscapes and hills used for annealing-type heuristics does not apply to HFS, which belongs to a different heuristic paradigm based on subproblem optimization. Instead we may consider performance in light of the *optimal substructure property*, which holds when an optimal solution to a global problem contains optimal solutions to subproblems. This property is often invoked in proofs of correctness of polynomial-time algorithms, as well as in proofs of inapproximability and lower bounds for NP-hard problems (for examples see [24], [25] [26], [27]). While heuristics from this paradigm can give excellent performance on some types of inputs, they are vulnerable to scenarios where the optimal substructure property does not hold.

Recall that HFS works by iteratively selecting random spanning subgraphs of G and calculating the optimal solution for each selection. Two examples of spanning subgraphs on a C_8 graph are shown in Figure 4; the small boxes represent Chimera cells. The left side shows a spanning subgraph with dark lines connecting cells via eight vertical *branches* and one horizontal *crossbar*; the HFS3 strategy constructs subgraphs like this, with branches and crossbars that are one cell wide. The right side shows a spanning subgraph with five branches and a crossbar that are one and two cells wide, as constructed by HFS4. The purple boxes show how these subgraphs intersect with individual C_2 (left) and C_3 (right) clauses; the red and blue dots indicate locations of weak and strong fields of opposite signs.

On the left, the higher purple box shows a clause intersection with two branches: this splits the clause in two, so that the *subgraph-optimal* solution computed by HFS (nonzero fields are unfrustrated) is incompatible with the *clause-optimal* solution (weak fields are frustrated), a violation of the optimal substructure property. The lower purple box shows a clause intersection with the crossbar: this intersection contains three-fourths of the couplings, enough to ensure that the subgraph-optimal solution aligns with the clause-optimal solution so that the optimal substructure property is not violated. But note that the optimal spin assignment calculated on this clause could be undone if the cause intersects two branches in a later iteration of the algorithm and the later subgraph is accepted as an overall improvement.

As HFS3 iterates, randomly selecting and optimizing subgraphs, spin configurations for individual clauses may be optimized, and possibly later assigned non-optimal values, according to a (random) sequence of crossbar locations in the full graph. The TTB results for HFS3 in Figure 3 show that at $C_{2,8}$ (507 variables), HFS3 succeeds: this is because the cross-

bar intersects a high proportion (one-fourth) of all clauses at each iteration. The probability of a successful outcome decreases as the ratio of branch area to crossbar area grows with problem size, causing R^*_{HF3} to grow beyond the timeout limit.

One way to prevent this scenario is to optimize over larger subgraphs, as does HFS4. A double-wide branch is big enough to find clause-optimal solutions for every CR_2, and just a few iterations are needed to align each clause with a double branch or a double crossbar. In this case, the probability of finding a violation of the optimal substructure property does not grow with problem size. As a result, as shown in Figure 3, HFS4 median TTB to solve $CR_{2,c}$ problems is near its minimum throughout.

On the other hand, as suggested by the right side of Figure 4, HFS4 can be similarly thwarted using C_3 clauses. The next experiment explores this scenario.

4.2 $CR_{3,c}$ problems

This experiment compares performance on CR_3 ring clauses shown on the left side of Figure 5. As before, FM couplings create a 4-ply ring around the exterior perimeter, and weak and strong fields are placed on opposite diagonals to create a majority circuit; note the center cell is isolated.

Figure 6 shows median TTB times on random $CR_{3,c}$ problems of size $c = 9, 12, 15$, corresponding to $n = 642, 1144, 1789$ variables. As predicted by the discussion in the previous section, HFS4 (as well as HFS3) fails to find target solutions B within timeout limits at $c = 15$, as the ratio of branch area to crossbar area increases with problem size.

Comparison to Figure 3 shows that, in contrast to HFS4, the other solvers have generally lower TTB on $CR_{3,12}$ problems than on $CR_{2,12}$ problems (of size common to both graphs). Although both parameterizations of SA reached time-out in the median case at $c = 15$, the dotted min-curves show that in some cases the solver found target solutions in just under the 200-second limit. Both versions of SA were able to solve all $CR_{3,12}$ problems within the time limit. Similarly, comparison of anneal times in Figures 3 and 6 shows that the QPU finds $CR_{3,c}$ problems somewhat easier to solve.

This somewhat nonintuitive result can be attributed to the introduction of isolated (easy-to-solve) cells in clause centers, and the relative sparsity of weak fields in the global problem (a $CR_{2,12}$ has 36 weak cells and a $CR_{3,12}$ has 16 weak cells). This creates less overall frustration for the QPU and SA solvers, and fewer violations of the optimal substructure property for HFS3.

The next experiment explores another variation on clause structures.

4.3 Bipartite clause problems

The right side of Figure 5 shows a CB_3 *bipartite* clause. The blue couplings between cells are ferromagnetic chains with $J_{chain} = -1.5$, connecting pairs of qubits vertically and horizontally. The vertical chains create a set $\mathcal{A}$ of 12 logical nodes, and the horizontal chains create a set $\mathcal{B}$ of 12 logical nodes. This gives a logical biparite graph $K_{12,12}$. The bipartition edges (a, b) are connected with ferromagnetic $J_{cell} = +.5$ couplings inside the cells. This logi-

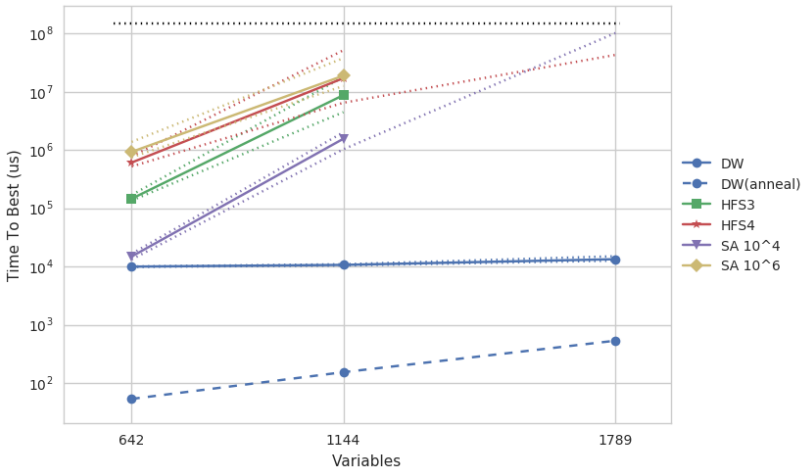

Figure 6: TTB on $CR_{3,c}$ problems for $c = 9, 12, 15$. Missing data points correspond to software timeouts. Solid and dashed lines denote median times, dotted lines denote minimum and maximum observed times. The 200-second timeout limit is shown by the horizontal dotted line at top DW denotes the D-Wave QPU.

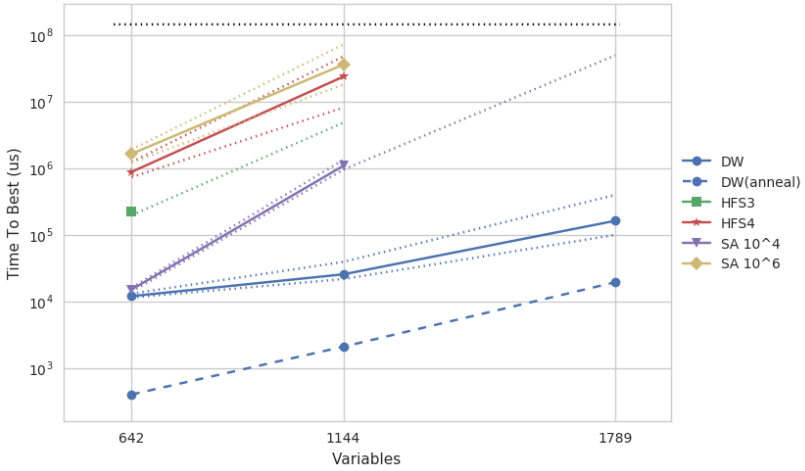

Figure 7: TTB on $CB_{3,c}$ bipartite problems. Missing data points correspond to software timeouts. Solid and dashed lines denote median times, dotted lines denote minimum and maximum observed times. The 200-second timeout limit is shown by the horizontal dotted line at top. DW denotes the D-Wave QPU.

First Experiment

Input	DW anneal	DW total	SA	HFS3	HFS4
$CR_{2,16}$	4.3e3	4.1e4	(2e8)	(2e8)	2.3e5
$CR_{3,15}$	3.2e2	1.2e4	(2e8)	(2e8)	(2e8)
$CB_{3,15}$	1.7e4	1.5e5	(2e8)	(2e8)	(2e8)
max T_{DWA}/3.2e2			625,000	625,000	625,000
max T_{DWT}/1.2e4			16,667	16,667	16,667

Table 1: Summaries of median time-to-best (TTB) in microseconds from each experiment, at the largest problem size. The notation 9.8*e*7 is shorthand for 9.8×10^7. The notation (2e8) indicates the solver timed out at 200 seconds (2e8 microseconds) without finding target solutions matching those found by the D-Wave QPU. The bottom rows show the largest speedups seen for each solver (using the 200-second time limit compared to D-Wave total and anneal times.

cal graph has two ground states, in which the logical sets have different spin orientations, either $(a = \uparrow, b = \downarrow)$ or $(a = \downarrow, b = \uparrow)$.

Weak and strong fields of opposite sign are placed on nodes from $\mathcal{A}$ only, in opposite corners of the clause. Now, as with CR_3 ring clauses, there is one ground state with weak fields frustrated, and a first exited state with strong fields frustrated; in higher energy states, one or more couplings are frustrated. Figure 7 shows TTB for these solvers on $CB_{3,c}$ bipartite problems with $c \in \{9, 12, 15\}$.

Comparison to Figure 6 reveals somewhat mixed results. Most noticeably, the blue dashed curve shows that the QPU finds bipartite problems somewhat more challenging to solve, yielding a 50-fold increase in anneal times compared to ring problems at C_{15}. The comparative unevenness of the solution landscape (a combination of high and low hills) may be a factor here.

The purple curve shows that SA has nearly identical performance on ring and bipartite problems. The red and green curves show that bipartite problems are somewhat harder than ring problems for both HFS3 and HFS4 at C_{12} sizes. Investigation of time components suggests that much of this time difference is due to higher initialization costs and higher costs per optimization, which may be partially attributed to the fact that CB_3 is a denser problem with no isolated center cell.

Table 1 summarizes median TTB observed for each solver on each problem, at the largest problem sizes tested. The top section shows results for the first experiment on $CR_{2,16}$ problems, and the bottom section shows results for the second and third experiments, on $CR_{3,15}$ and $CB_{3,15}$ problems. In this table, the notation 9.8*e*7 is shorthand for 9.8×10^7; the notation $(2e8)$ marks a case where a software solver reached the 200 second timeout limit on without finding solutions matching the best-in-pool solution B. The two bottom rows show the maximum runtime speedups observed for QPU anneal and total times, compared to classical solver times, substituting the timeout limit as a lower bound on true computation time. Because of the 9.4 ms lower bound on DW total time and the 2×10^8 μs limit on software times, the maximum *observable* speedup in total time in these experiments is 21,276.

5 Discussion

This report describes a new family of native inputs for D-Wave QPU topologies, called *clause problems* that can be used to study local constraint structures, which appear to be common in inputs translated from general NP-hard problems to the D-Wave native topology. Although these inputs have minimal global structure, they can be designed to create significant performance challenges for two native classical solvers, simulated annealing (SA) and Hamze-de Frietas-Selby (HFS). A series of experiments is used to expose the key properties that make these inputs hard.

The first experiment shows that relative performance of SA and the QPU on both weak-strong clusters and CR_2 rings can be explained by the presence of weak and strong fields of opposite sign, and blocks of ferromagnetic couplings between and within cells. The fields lure SA into a local minimum early in the anneal: that is, starting from a random initial state, it is probabilistically drawn to a local minimum where both weak and strong fields are unfrustrated but couplings are frustrated; from there it must make its way across high hills created by the couplings, to discover the ground state. We tried many parameterizations of the code we have implemented, and all combinations failed to negotiate these landscapes within the time limits. It is possible that more success would be found with a different anneal schedule, perhaps that converges relatively slowly in early stages of the anneal.

The first experiment also exposes a fundamental vulnerability of HFS to inputs that create violations of the optimal substructure property. In the case of HFS3 such violations are frequent (whenever clauses intersect branches); for HFS4 the violations are rare. This creates a qualitative difference in performance of these two parameterizations on these inputs.

The second experiment extends this argument to show that HFS4 similarly fails on $CR_{3,c}$ clauses. An obvious strategy for coping with failure on $CR_{3,c}$ clauses is to implement HFS5, with subgraphs having triple-wide branches. However, further extension of this algorithm strategy is not likely to yield a fast heuristic: the cost of optimizing a subgraph with k-wide branches grows as $\Theta(n^k)$, suggesting that the optimization step of HFS5 would be about 2000 times slower than HFS4 on a C_{16}-sized problem.

Another possibility is to develop a version of HFS that randomly optimizes "clause-shaped" subgraphs rather than "tree-shaped" subgraphs of the Chimera graph. However, it would not be difficult to develop inputs that create violations of the optimal substructure property for that version as well. Extending this algorithmic strategy to its logical end to cover multiple contingencies yields a brute force algorithm with $\Theta(n^n)$ computation time. The larger point is that all heuristics within the substructure-optimization paradigm, including HFS, perform poorly on inputs for which substructure-optimal solutions do not align (or rarely align) with globally-optimal solutions.

The third experiment reinforces our intuition that performance of all of these heuristic solvers is tied less to the specific inputs used, but rather to whether the underlying properties that drive performance are present. Additional tests not shown here suggest that performance profiles are robust to other changes in, e.g., the number and magnitude of fields, placement of fields on the four cells, and to a lesser extend, magnitude and number of coupling weights. Some very preliminary tests on other native solvers, including GPU implementations of quantum Monte Carlo and Parallel Tempering, suggest that those

annealing-based solvers experience difficulties similar to those of SA on these types of inputs; however further exploration of the large parameter space offered by those solvers is needed before firm conclusions can be drawn.

These results suggest that SA and HFS, which generally show good performance on native inputs having random weights (such as random Chimera-structured problems), might experience difficulties on inputs that are translated from general NP-hard problems, if these properties are present. For example, problems that are transformed and embedded tend to contain ferromagnetic chains of large weight, which would create the rugged landscapes that SA finds difficult to traverse. In the case of HFS, chains that cross multiple branches of the optimized subtrees (perhaps on a diagonal or stairstep fashion) would represent violations of the optimal substructure property. Our understanding of the underlying mechanisms that drive performance of quantum annealing processors is quite primitive, and it is extremely difficult to predict how well D-Wave QPUs will perform by comparison. This is an interesting question for future experimental research.

References

[1] S. Boixo, T. Alabash, F. M. Spedalierie, N. Chancellor, and D. A. Lidar, "Experimental signature of programmable quantum annealing," *Nature Communications*, vol. 4, no. 2067, 2012.

[2] S. Boixo et al., "Evidence for quantum annealing with more than one hundred qubits," *Nature Physics*, vol. 10, pp. 218–224, 2014.

[3] P. I. Bunyk et al., "Architectural considerations in the design of a superconducting quantum annealing processor," *IEEE Transactions on Applied Superconductivity*, 2014.

[4] N. G. Dickson et al., "Thermally assisted quantum annealing of a 16-qubit problem," *Nature Communications*, 2013.

[5] M. W. Johnson, M. H. S. Amin, S. Gildert, T. Lanting, F. Hamze, *et al.*, "Quantum annealing with manufactured spins," *Nature*, vol. 473, no. 7346, 2011.

[6] T. Lanting, A. J. Przbysz, A. Y. Smirnov, F. M. Spedalieri, M. H. Amin, *et al.*, "Entanglement in a quantum annealing processor," *Phys. Rev. X*, vol. 4, no. 021041, 2014.

[7] W. Vinci, T. Albash, A. Mishra, P. Warburton, and D. A. Lidar, "Distinguishing classical and quantum models for the D-Wave device," 2015, arXiv 1403.4228v4.

[8] Z. Bian, F. Chudak, R. Israel, B. Lackey, W. G. Macready, and A. Roy, "Mapping constrained optimization problems to quantum annealing with applications to fault diagnosis," 2016, arXiv 1603.03111v1.

[9] I. Trummer and C. Koch, "Multiple query optimization on the D-Wave 2X adiabatic quantum computer," *VLDB*, 2016.

[10] D. Venturelli, D. J. J. Marchand, and G. Rojo, "Quantum annealing implementation of job-shop scheduling," 2015, arXiv 1506.08479v1.

[11] J. King, S. Yarkoni, J. Raymond, I. Ozfidan, A. D. King, M. M. Nevisi, J. P. Hilton, and C. C. McGeoch, "Quantum annealing amid local ruggedness and global frustration," *D-Wave Technical Report Series*, no. 14-1003A-B, 2016.

[12] S. Boixo et al., "Computational role of multiqubit tunneling in a quantum annealer," 2015, arXiv 1502.05754v1.

[13] V. S Denchev, S. Boixo, S. V. Isakov, N. Ding, R. Babbush, V. Smelyanskiy, J. Martinis, and H. Neven, "What is the computational value of finite range tunneling?" *Phys Rev. X*, vol. 6, no. (3) 031015, 2015.

[14] S. Mandrà, Z. Zhu, W. Wang, A. Perdomo-Ortiz, and H. G. Katzgraber, "Strengths and weaknesses of weak-strong cluster problems: A detailed overview of state-of-the-art classical heuristics vs quantum approaches," *ArXiv 1604.01746v1*, 2016.

[15] J. King, S. Yarkoni, M. M. Nevisi, J. P. Hilton, and C. C. McGeoch, *Benchmarking a quantum annealing processor with the time-to-target metric*, arXiv 1508.05087v1, 2015.

[16] T. Rønnow, Z. Wang, J. Job, S. Boixo, S. Isakov, D. Wecker, J. Martinis, D. Lidar, and M. Troyer, "Defining and detecting quantum speedup," *Science*, vol. 345, no. 6195, pp. 420–424, 2014.

[17] W. G. Macready and D. H. Wolpert, "What makes an optimization problem hard?" *Complexity*, vol. 5, pp. 40–46, 1996.

[18] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[19] D. H. Wolpert, "What the No Free Lunch theorems really mean; how to improve search algorithms," *SFI working paper 2012-10-017*, 2012.

[20] S. Isakov, I. N. Zintchenko, T. R. nnow, and M. Troyer, "Optimized simulated annealing for Ising spin glasses," *Arxiv 10401.1084v2*, 2014.

[21] A. Selby, "Github repository: QUBO-Chimera," *Github.com/alex1770/qubo-chimera*, 2013.

[22] ——, "Efficient subgraph-based sampling of Ising-type models with frustration," 2014, arXiv 1409.3934v1.

[23] F. Hamze and N. de Freitas, "Intracluster moves for constrained discrete-space MCMC," *Proceedings of UAI2010*, 2010, arXiv 1203.3484.

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms (3rd edition)*. MIT Press, 2009.

[25] A. Ferrante, G. Pandurangan, and K. Park, "On the hardness of optimization in power law graphs," *Theoretical Computer Science*, vol. 393, pp. 220–230, 2008.

[26] Y. Shen, D. T. Nguyen, and M. T. Thai, "Hardness complexity of optimal substructure problems on power-law graphs," *Handbook of Optimization in Complex Networks*, 2011.

[27] X. Wang and J. Tian, "Dynamic programming for NP-hard problems," *Procedia Engineering*, vol. 15, pp. 3396–3400, 2011.