Experimental Evaluation of an Adiabiatic Quantum System for Combinatorial Optimization

Catherine C. McGeoch
Amherst College
Amherst MA, USA
ccm@cs.amherst.edu

Cong Wang
Simon Fraser University
Burnaby BC, CA
cwa9@cs.sfu.ca

ABSTRACT

This paper describes an experimental study of a novel computing system (algorithm plus platform) that carries out *quantum annealing*, a type of *adiabatic quantum computation*, to solve optimization problems. We compare this system to three conventional software solvers, using instances from three NP-hard problem domains. We also describe experiments to learn how performance of the quantum annealing algorithm depends on input.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems; F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms

Algorithms, Experimentation, Performance

Keywords

Adiabatic Quantum Computing, Quantum Annealing, D-Wave, Heuristics

1. INTRODUCTION

Adiabatic quantum computation (AQC),[1] proposed in 2000 by Farhi et al. [17], represents a new model of computation as well as a new paradigm in algorithm design. The computational model is polynomially equivalent to the better-known quantum gate model [1], [18], [33]. Theoretical analysis of specific AQC algorithms has returned mixed results to date, with indications of exponential speedups and sometimes slowdowns over conventional algorithms.

This paper presents an experimental study of algorithms based on *quantum annealing* (a type of AQC computation), running on a special-purpose D-Wave Two platform[2] containing 439 quantum bits (qubits). Earlier systems with between 84 and 108 qubits have been used for finding Ramsey numbers [4], binary classification in image matching [28], and 3D protein folding [29].

[1]The term "adiabatic" in this context refers to a quantum process where no population exchange between allowed states of the system occurs.

The "native" problem for this system is a restriction – to *Chimera-structured* inputs defined in Section 2 – of the Ising Spin Model (IM). Native instances can be solved directly on the quantum hardware (QA), while general inputs are solved by a hybrid approach (called Blackbox) that alternates heuristic search with hardware queries.

We report on two experimental projects. First, QA and Blackbox are compared to three conventional software solvers: CPLEX [23], METSlib tabu search (TABU) [27], and a branch-and-bound solver called Akmaxsat (AK) [26] . The solvers are evaluated using instances from three NP-Hard problems: Quadratic Unconstrained Binary Optimization (QUBO); Weighed Maximum 2-Satisfiability (W2SAT), and the Quadratic Assignment Problem (QAP).

In horserace terms, QA dominates on the Chimera-structure QUBO problems: at the largest problem size $n = 439$, CPLEX (best among the software solvers), returns comparable results running about 3600 times slower than the hardware. On the W2SAT problems, Blackbox, AK, and TABU all find optimal solutions within the same time frame. On the QAP problems, Blackbox finds best solutions in 28 of 33 cases, compared to 9 cases for TABU (the next-best solver on these inputs).

Note that these results can be regarded as "snapshots" of performance for specific implementations on specific input sets, nothing more. Experimental studies of heuristics are notoriously difficult to generalize. Furthermore – unlike experiments in the natural sciences – no experimental assessment of heuristic performance can be considered final, since performance is a moving target that improves over time as new ideas are incorporated into algorithms and code. Future research will likely turn up better solvers and strategies.

As a case in point, our second project compares the V5 hardware chip used in our first study to a V6 chip that became operational after the study was completed. V6 is three to five times faster than V5, and can solve problems as large as $n = 502$.

[2]Manufactured by D-Wave Systems Inc., BC, Canada. We thank D-Wave staff for their generous assistance in setting up experiments, collecting data, and helping us to understand AQC and quantum annealing: Zhengbing Bian, Fabian Chudak, Suzanne Gildert, Mani Ranjbar, Geordie Rose, Murray Thom, and Dominic Walliman.

Section 2 presents a basic framework for understanding adiabatic quantum computation, surveys its theoretical underpinnings, and describes how theory relates to practice. Sections 3 and 4 present experimental results. Section 5 offers some remarks about this novel approach to solving problems in combinatorial optimization.

2. BACKGROUND

We start with a brief overview of adiabatic quantum computing. For more information see [9], [24], or [30].

2.1 Adiabatic Quantum Computing

A *Hamiltonian system* is a mathematical formalism to describe a physical process that evolves continuously over time $t : 0 \to T$. The process involves n binary-valued particles that can interact to influence one another's state. The possible *energies* of the system at time t are described by a Hamiltonian $H(t)$, a matrix of size $2^n \times 2^n$. The eigenvectors of $H(t)$ correspond to the eigenstates (measurable states) of the system, and the eigenvalues to the energies of those eigenstates. The *ground state* $\phi(t)$ is the eigenstate with the lowest energy at time t.

Now, suppose P is a minimization problem on binary strings with objective function $f : \{0,1\}^n \to \mathbb{R}$. In AQC the problem variables $X = \{x_1, \ldots, x_n\}$ correspond to the states of the particles in the process, which evolve to a low-energy state that corresponds to a low-cost solution to P.

Under a classical mechanical model of the system, each particle is represented by a bit having state 0 or 1. Under a quantum model, particles are represented by *qubits*, which can exist in superposition. That is, the state of a qubit is represented by a unit vector $x_i = (\alpha, \beta)$ in $\mathbb{C}$, which implies that $|\alpha|^2 + |\beta|^2 = 1$. A quantum state cannot be examined: when a qubit is measured its state "collapses" to 0 with probability $|\alpha|^2$ and 1 with probability $|\beta|^2$.

For convenience let $s = t/T$, so $0 \leq s \leq 1$. An adiabatic quantum algorithm for solving problem P with objective function f is specified by three components:

1. An initial Hamiltonian $H(0)$, chosen so that the initial (quantum) ground state $\phi(0)$ is easy to construct.

2. A final Hamiltonian $H(1)$, that encodes the objective function f, in such a way that ground state $\phi(1)$ corresponds to an optimal solution to P.

3. An *adiabatic evolution path*, a pair of functions $A(s)$ decreasing from 1 to 0, and $B(s)$ increasing from 0 to 1, that control the transition from $H(0)$ to $H(1)$:

$$H(s) = A(s)H(0) + B(s)H(1).$$

By the *adiabatic theorem* [7] (or see [17]), if the transition is carried out slowly enough, a system that is initially in a ground state will stay in a ground state. Thus $\phi(0)$ evolves to a ground state $\phi(1)$ that minimizes f, and when measured, yields an optimal solution X to P.

The adiabatic theorem also states that the amount of time necessary for the ground state to be preserved during transition is bounded below by a function $h(n)/g^2$ where $h(n)$ is polynomial in n (whenever f is) and g is the *minimum spectral gap* of the system, i.e. the minimum difference between the lowest (ground) eigenvalue and the second-lowest eigenvalue of $H(s)$ at any time in the process. Therefore, the algorithm finds an optimal solution to P in polynomial time exactly when g is an inverse polynomial in problem size n.

As a general rule, g is difficult to bound analytically for a given problem or even to compute for a given instance. The next section surveys what is known about spectral gaps and AQC.

2.2 Theoretical Underpinnings

Algorithms. Farhi et al. [17] introduce the notion of adiabatic quantum computation and construct an AQ algorithm (referred to here as AQ_F) to solve several combinatorial problems including 3SAT. In [18], Farhi et al. present numerical simulations suggesting that g is (inverse) polynomial in n for small (but arguably difficult) random instances of Exact 3-Cover. Farhi, Goldstone and Gutmann [16] also give two example problems where AQ_F takes polynomial time while conventional simulated annealing has an exponential lower bound.

AQ_F has since been shown to take exponential time on some problems. For example, van Dam and Vazirani [34] construct a family of 3SAT instances for which the gap g is exponentially small. They also describe a problem that is *linear* on a conventional model of computation but exponential for AQ_F. Altshuler, Krovi, and Roland [2] describe a random model of Exact 3-Cover for which g is exponentially small with high probability.

On the other hand, Farhi et al [17], [19], have observed that g can sometimes be improved dramatically by simple adjustments to the algorithm. Choi [14] demonstrates that the initial Hamiltonian of AQ_F can be modified so that the argument of [2] no longer applies to g. She also presents numerical evidence that small g values can sometimes be increased by re-scaling problem weights. Dickson and Amin [15] show analytically that in the case of Maximum Independent Set, there must always exist an adiabatic path for which the exponentially small gaps of [2] do not occur. Somma and Boixo [31] show that sometimes the final Hamiltonian can be transformed in a way that preserves the ground state but (quadratically) amplifies g.

Thus, whether g is (inverse) polynomial in n can depend on algorithm parameters as well as on instance structure and instance weights. Many fundamental questions about the computational power of AQ algorithms remain open.

Models of computation. Aharanov et al. [1] show that a universal AQC model can efficiently simulate the conventional quantum gate model. Together with [18] and [33], this implies that the two quantum models are polynomially equivalent. They also demonstrate a quadratic speedup over any classical algorithm for the search problem, a result analogous to Grover's algorithm [21] for the standard quantum gate model.

Biamonte and Love [5] consider ways to simplify the model of [1], with an eye toward implementability. They show that one simple model — the one that has been realized by D-Wave systems and by others — is QMA-complete but unlikely to be universal for AQC (the class Quantum-Merlin-Authur is the quantum analog of NP).

As is the case with any computing device, there is a gap between the perfect theoretical ideal and the physical reality. Some differences between abstract models and working D-Wave platforms are sketched in the next section.

2.3 Quantum Computing in Practice

The Ising Spin Model (IM) problem is, given real weights J_{ij} and h_i, to find an assignment of *spins* $(-1, +1)$ to variables $S = \{s_1 \ldots s_n\}$ to minimize

$$M(S) \;=\; \sum_{i<j} J_{ij} s_i s_j + \sum_i h_i s_i. \qquad (1)$$

Note that terms with negative J_{ij} are minimized if $s_i = s_j$, terms with positive J_{ij} are minimized if $s_i \neq s_j$, and the sign of h_i has a similar effect on s_i.

A D-Wave platform comprises a conventional Linux front end coupled with an analog unit ("the hardware"). A D-Wave Two hardware chip contains up to 512 *flux qubits*, which are microscopic loops of metal (niobium) that are capable of quantum behavior at low temperatures. That is, electrical currents in the loops can flow in clockwise $(+1)$ or counter-clockwise (-1) direction, or both, when in quantum superposition [20]. Qubits are connected to neighbors according to a graph structure described below. The hardware is controlled by a framework of Josephson junctions that allow individual qubit values to be stored and read, and to influence the states of neighboring qubits.

The front end accepts IM instances and maps them onto the hardware graph so that weights h_i are assigned to vertices (qubits) and J_{ij} to edges in the hardware graph. The hardware carries out a process of quantum annealing to find a minimum-energy state S, as follows.

1. To initialize, qubits are placed in superposition ground states $s_i = (\alpha, \beta)$ such that $\alpha, \beta = \sqrt{.5}$ (each binary state equally likely).

2. The annealing path from initial to final state involves increasing the "influence" of the weights h_i and J_{ij} on quibit states.

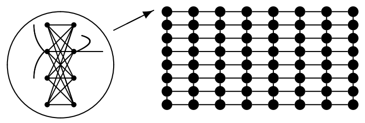
3. At the end of the annealing process, qubits and qubit pairs are weighted according to h_i and J_{ij}. If the system remains in ground state, the collection S of measured qubit states minimizes $M(S)$.

In theory, if the transition is carried out slowly enough, S is an optimal solution to M, for any M. However, theoretical guarantees are based on several assumptions that are not necessarily met by the physical system, as follows.

First, AQC theory assumes there is no influence from the environment, whereas the physical system can be affected by its environment. In particular, if (1) the ambient temperature (energy) is higher than the spectral gap g, or (2) there a sufficient level of electromagnetic noise from an outside source, then with nonzero probability the qubits will "jump" from ground state to a nearby state during the annealling process. For this reason the algorithm tested here is called a quantum annealing algorithm rather than an adiabatic quantum algorithm. There is no guarantee that the final solution will be close to optimal (although in practice it often is).

To increase the probability of a successful transition, the hardware chip is cooled to approximately 0.02° Kelvin and operated inside a well-shielded cabinet. Furthermore the system is configured to run multiple anneals, returning a vector $\chi = (S_1 \ldots S_k)$ of k independently sampled solutions for each input. Typically $k = 1000$ in our experiments.

Second, qubit-to-qubit interactions are restricted to neighbors in a *Chimera* graph structure C_g containing $512 = 8 \cdot g^2$ qubits as shown below. Groups of eight qubits are connected as bipartite graphs; in each group, the 4 left nodes are also connected to their respective north/south grid neighbors and the 4 right nodes are connected to their east/west neighbors. Thus internal nodes have degree 6 and boundary nodes have degree 5.



$C_8, \;\; n = 512$

A D-Wave Two chip is designed to hold 512 qubits arranged as a C_8 Chimera graph, but the manufacturing process leaves some qubits inoperable. Thus computation takes place on a *hardware graph* H that is a subgraph of C_8. IM instances that cannot be mapped to H cannot be solved directly in hardware.

In some applications this is not a problem: the coefficients J_{ij} and h_i are obtained by sampling (an image, for example), and it is not difficult to choose sample sites ((i, j) pairs) beforehand to match H. Furthermore, the mapping requires only that instances be embedded into *a minor* of H. Choi [13] shows that the complete graph K_m can be minor-embedded into the upper triangle of a Chimera graph C_m. Finding good algorithms and heuristics for computing minor-embeddings of H is an interesting problem for future research. If an embedding is not known the problem can be solved by Blackbox, described in the next section.

Finally there has been some debate as to whether D-Wave chips form a "truly' quantum system; this has been demonstrated for smaller (8 and 16 qubit) systems [22], [25] but not, as yet, for the full hardware graph. Boixo et al. [6] report that the hardware demonstrates quantum tunneling (which would be inconsistent with classical annealing), and find some evidence of qubit entanglement. A recent review article in *Science* [12] concludes that the hardware is "at least a little quantum mechanical." Either way, it remains an interesting question to learn how well this novel system competes against conventional approaches to NP-hard optimization problems

2.4 Blackbox

Let G denote the *connectivity graph* of a given IM input M. This graph has n vertices corresponding to variables s_i, and edges (s_i, s_j) present whenever $J_{ij} \neq 0$. As mentioned above, M can be solved by minor-embedding G into the hardware graph H and then carrying out the quantum annealing process.

Or, if such an embedding is not known, the problem can be submitted to *Blackbox*, a hybrid solver developed by D-Wave Systems. Blackbox accepts a general problem instance P having objective function $f(X)$ defined on n binary variables $X = x_1, \ldots, x_n$ (where n is bounded by the number of qubits in hardware). It carries out a heuristic search (specfically Tabu search) process, starting with an initial random solution and iterating to step from neighbor to neighbor in the solution space, towards a solution that minimizes $f(X)$.

Blackbox's selection of the neighborhood set at each iteration is guided by a query to the quantum hardware, as follows. Given a current solution X_i, Blackbox generates a neighborhood N_i of solutions that have Hamming distance

1 from X_i, and computes their costs using f. It then builds an Ising Spin Model M that approximates f at these neighbor points, and matches the structure of H. M is sent to the hardware, which returns a sample $N'_i = (X'_1, \ldots X'_k)$ of k solutions. Blackbox then selects a minimum-cost solution from $\{N'_i \cup N_i\}$ that is not on its Tabu list (which records recent solutions and steps in the process), updates the list, and moves to that solution for the next iteration. Thus each iteration requires one hardware query, plus n objective function evaluations for N_i, plus some number $k' \leq k$ of function evaluations for the unique solutions in N'_i. When Blackbox reaches a preset limit on total objective function evaluations, it stops and reports the best solution found.

3. PERFORMANCE EVALUATION

Our first experimental study uses instances from three NP-Hard optimization problems: Quadratic Unconstrained Binary Optimization (QUBO); Weighted Maximum 2-Satisfiability (W2SAT); and the Quadratic Assignment Problem (QAP). Our QUBO tests used Chimera-structured inputs that, when formulated as IMs, can be solved directly in hardware. (QUBO instances can be trivially transformed to IM instances). The other two problems have general structure and are solved by Blackbox.

The software solvers evaluated here are: IBM ILOG CPLEX Optimizer version 12.3 [23]; METSlib Tabu (TABU) an open-source implementation of Tabu search [27]; and Akmaxsat (AK) [26], a branch-and-bound solver designed for W2SAT.

CPLEX and AK are exact solvers that can certify the optimality of their solutions if given enough time and space. QA, TABU, and Blackbox are heuristics that return the best solutions they can find but provide no certification.

For convenience and uniformity, instances for all three optimization problems were transformed to QUBO instances for solution in software, and to IM instances for solution in hardware. The software solvers were invoked by a common Matlab front end that carried out the following tasks: read a QUBO instance, translate it to respective solver input formats, invoke each solver, and record results.

Our experimental work began with a pilot study aimed at finding good parameter settings for the software solvers (primarily on QUBO instances) and evaluating options for timing this diverse collection of platforms and software. The pilot study took about 20 CPU-days of computation. Here are some details.

CPLEX: Since QUBO has a quadratic objective function, the quadratic programming (QP) module was invoked throughout. We found no improvements from using the instance-specific internal tuner or changing default parameters; therefore default settings were used in the main experiments. The Parallel Mode switch was turned off, causing CPLEX to run in single core mode. (Use of Parallel Mode produced speedups around 10 percent on our 4-core platform but also created timing anomalies.)

TABU: Parameters were set as follows: the initial solution is selected uniformly at random; the Tabu list (holding recently-touched variables) is of size $\min(20, n/4)$; the number of unsuccessful iterations before random restart is 3500; and at each restart the new starting solution was obtained by randomly flipping $\max(10, 0.4n)$ bits of the best solution found so far. We also looked at an in-house (proprietary to D-Wave) version of tabu search that is highly tuned for QUBO instances. The in-house version was faster than METSlib TABU but neither could compete with QA on large QUBO instances.

AK: We considered two branch-and-bound programs called `akmaxsat` and `akmaxsat-ls`, that performed well in a recent Max-SAT competition [3]. `Akmaxsat-ls` incorporates a preprocessing step carried out by a third-party standalone program. Although `akmaxsat-ls` performed better in the competition, our pilot comparisons returned mixed results on our data sets. Since the introduction of standalone preprocessing code significantly complexified our runtime measurements, we chose the simpler `akmaxsat` for the main experiments. We do not believe that using `akmaxsat-ls` would have altered our overall conclusions.

Based on our tests we are confident that the solvers represent reasonably well-tuned implementations. However, we make no claim that the solvers tested here represent the most efficient heuristics (or implementations) possible for the problems we tested. As mentioned in Section 1, such claims are rarely appropriate in research on heuristic performance. One reason is that improving any given solver or strategy is an unending process: better ideas are always around the corner. Furthermore, most heuristics can be "tuned" for specific problem domains or instance classes. Highly-tuned heuristics tend to do very well on their target problems (best best case), but their performance may degrade significantly when applied to problems outside their comfort zone (worst worst case). In contrast a general solver might be never best but never worst on any given problem domain.

With this in mind CPLEX, TABU and Blackbox may be considered general-purpose solvers. AK is designed for best performance on W2SAT problems, and the QA hardware is specialized to Chimera-structured IM (or QUBO) instances.

3.1 Timing issues

All software solver tests were carried out on a suite of seven Lenovo ThinkStation S30 0568 workstations, each containing one Intel Xeon E5-2609/2.4GHz Quad-Core processor with 16GB RAM. The operating system was Ubuntu 64-bit 12.04 LTS.

Blackbox runs on a Lenovo d20 workstation containing two Intel Xeon X5550@1.6GHz Quad-Core processors with 16GB RAM. The operating system is Fedora 15. The number of hardware samples per main loop iteration was set to $k = 1000$ and the stopping rule was set to 10^7 function evaluations.

The QA algorithm was run on a hardware chip named Vesuvius 5 (V5) that contains 439 working qubits.

It is a challenging problem to find precise, accurate, and commensurable runtime measurements for these diverse solution strategies. We adopted the following conventions.

All software runtimes are Unix CPU times in units of seconds. The Matlab front end started timing immediately before solver invocation and stopped immediately upon return: thus the tasks carried out by the front end (including all I/O) are not included in our time measurements. All software tests were run on empty systems (with no competing user processes), measuring one solver on one instance, running on one core at a time. The Intel hyperthreading option (which is known to produce timing anomalies) was turned off. In addition to total CPU times, most tests produced "history" trace data, by which each solver recorded time and solution cost whenever a better solution was found.

For a given instance let S_x be the cost of the best solution found by solver x. CPLEX and AK were set to run to pre-set timeout limits (usually 30 minutes) but stop early if they can certify optimality. We distinguish between *find time*, the first time S_x is found by the solver, and *certify* or *finish time*, the total time needed to find and certify a solution. TABU always runs to a pre-set time limit, so finish times are constant and independent of n; the above-mentioned history data is used to obtain find times.

Our Blackbox tests ran as batch requests to a server that carries out individual trials concurrently on an eight-core platform. Because of demands on this resource it was not feasible to replicate our single-core software test environment (which would have increased Blackbox times from days to weeks). Although concurrency shrinks total batch times, it inflates the reported elapsed (wall clock) times per instance by introducing scheduler overhead and cache contention; more importantly, the concurrent processes must contend for access to the hardware chip. Therefore, we cannot with any confidence make direct and precise time comparisons between Blackbox and the software solvers. In what follows we take a conservative approach and describe only rough bounds on Blackbox computation times.

New questions arise when it comes to definingcomputation times for the V5 chip. Here we report *exclusive access time*, the total time used by the hardware to process a single instance while other instances wait. Exclusive access time is divided into *overhead time* t_1 for initializing the hardware, and *sampling time* t_2, which is the time to anneal and return one sample solution. Thus the total time per input is $T = t_1 + kt_2$, where k is the number of samples. Both t_1 and t_2 can be changed by adjusting the annealing path; however in normal practice they are set to default values when the chip becomes operational. For a fixed path, t_1 and t_2 increase *very* slightly with the number of qubits that must be stored and read. Throughout this section we report maximum times necessary to process the full hardware graph. These correspond to preset finish times that are independent of n; in Section 4 we describe a procedure for estimating hardware find times.

3.2 Chimera-structured QUBO Instances

Our first experiment compares performance on instances for Quadratic Unconstrained Binary Optimization: given a matrix Q of weights Q_{ij}, find an assignment of binary values (0,1) to variables $X = \{x_1 \ldots x_n\}$ to minimize

$$Q(X) \quad = \quad \sum_{i,j} Q_{ij} x_i x_j. \qquad (2)$$

This problem has wide application in machine learning and computer vision: Boros et al. [8] and Tavares [32] present a long list of applications that have been formulated as QUBO problems. Tavares also shows reductions from several classic NP-Hard problems to QUBO. QUBO instances can be transformed to and from IM instances by simple arithmetic.

This test uses QUBO instances with connectivity graphs $G \subset H$, which (after transformation to IM) can be solved directly in hardware. The experiment takes 100 random instances each at problem sizes $n = 32, 119, 184, 261, 349, 439$ (corresponding to subgrids of the hardware graph). Weights are drawn uniformly from the set $\{-1, +1\}$.

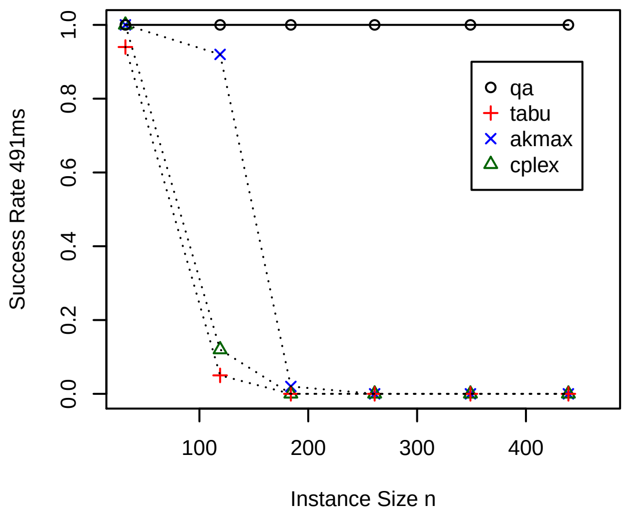


Figure 1: Success rates: proportion of best solutions found in 491ms CPU time (tabu, amax, cplex software) and exclusive access time (QA hardware).

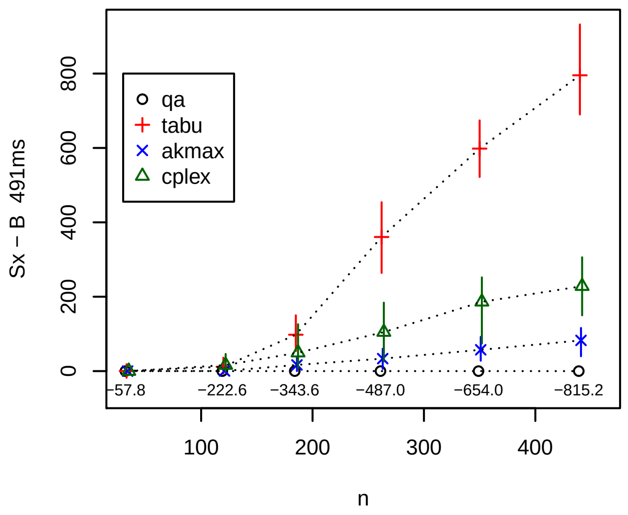


Figure 2: Differences between solution cost S_x in 491ms runtime, and best solution B for each input. Dotted lines connect means for each solver; vertical bars show the range of observations. The numbers at bottom are means $\overline{B}$ for each problem size.

Solutions in half a second.

The V5 hardware chip has setup time $t_1 = 201$ms and sampling time $t_2 = 0.29$ms, and returns $k = 1000$ samples per instance. This works out to $T = 491$ms exclusive access time per input, just under half a second.

Figure 1 compares performance when all four solvers are restricted to 491ms computation time. For each input let S_x be the cost of the best solution found by solver x and let B be the cost of the best solution found among the four solvers. The figure shows the *success rate* equal to the proportion of instances for which $S_x = B$. At $n = 32$ all but TABU enjoy a 100 percent success rate, while at $n = 184$ the software success rates drop to below 3 percent. QA was 100 percent successful at all problem sizes. By letting CPLEX run to 30 minutes we can certify that 585 of the 600 solutions found by QA (97 percent) are not only best, but optimal.

Figure 2 shows how solution cost S_x varies with n. The dotted lines connect mean differences $\overline{D_x} = \overline{(S_x - B)}$ for solver; the vertical lines mark the range of differences observed in 100 trials. The numbers near the bottom of the chart are means $\overline{B}$ for each n. For example, at $n = 439$ TABU produced solutions that were 805 units from B, which was -815.2 on average. Random solutions have mean 0, which suggests that TABU makes very little improvement on its initial random solution in this short time frame.

Time vs solution quality.

Next we focus on the largest problem size $n = 439$ and use the history data to see how performance varies with computation time. The table below shows success rates when the software solvers are allowed longer runtimes. The second column shows each time as a multiple of QA time.

t	$t/491$ms	AK	CPLEX	TABU
1 sec	2.0	0	0	0
30 sec	61.1	0	56	0
1 min	122.2	0	72	0
10 min	1222.0	0	99	0
30 min	3666.0	0	100	0

Clearly CPLEX makes the most use of the extra time: with a 10 minute limit it finds 99 percent of best solutions, and in 30 minutes it is able to find (and certify) optimal solutions for all 100 instances. The other two fail to find any optimal solutions within 30 minutes.

The table below shows mean differences $\overline{D_x}$ for solutions found within in these time limits. All solvers improve, but again CPLEX improves the most (ϵ signifies a positive $\overline{D_{cp}}$ value that rounds to 0).

t	$t/491$ms	D_{ak}	D_{cp}	D_{tb}
1 sec	2.0	72.5	224.7	759.5
30 sec	61.1	52.8	1.7	92.6
1 min	122.2	50.5	$\epsilon > 0$	57.4
10 min	1222.0	40.6	$\epsilon > 0$	48.4
30 min	3666.0	36.0	0.0	37.2

Further analysis in Section 4 shows that QA rarely needs all $k = 1000$ samples to find its best solution. For example if the best solution is returned 200 times, then only $5 = 1000/200$ samples are needed on average. At $n = 439$ the estimated median and mean find times for this problem work out to 202ms and 224ms. Thus we estimate QA could achieve the same success rates with (find) times below a quarter-second.

The QA hardware solver clearly out-performs the software solvers on instances that match its native structure. QA finds best solutions (97 percent of them optimal) in less than half a second, and results from Section 4 suggest that a quarter-second would suffice.

At $n = 439$, CPLEX, the best among software solvers, can match QA's success rate running 3666 times slower than QA's finish time, or about 7332 times slower than QA's estimated find time.

3.3 Weighted Max 2SAT

In the W2SAT problem, we are given a boolean formula in conjunctive normal form, on n variables, with 2 variables per clause, and weights $W = w_1 \dots w_m$ on the clauses. The problem is to find an assignment of boolean values to the variables to maximize the total weight of satisfied clauses. (W2SAT can be transformed to QUBO by negating literals and replacing $\vee, \wedge$ with multiplication and addition.)

This test uses 120 instances from the 2012 Max-SAT Evaluation competition [3]. The AK code used here outperformed all other complete solvers in that competition. These are random 2SAT formulas containing $n \in (100, 120, 140)$ variables with between 1200 and 1600 clauses. The competition instances have random integer weights from [1,10]; our tests assign random weights from the set

$$[-15/2, -13/2, \dots -1/2, 1/2, \dots 13/2, 15/2],$$

so that each weight is represented with 4 bits of precision. Since the problem connectivity graphs have degree at least 8.5 (more than Chimera graphs), Blackbox is used to solve these problems.

Software solvers were set to timeout after 30 minutes, and Blackbox to timeout after 10^7 objective function evaluations. CPLEX and TABU took around 3 days (elapsed on 7 cores) and AK and Blackbox took a few hours to complete all tests.

In terms of success rates, Blackbox, AK, and TABU were able to find optimal solutions (certified by AK) for all problems. CPLEX was less successful within 30 minute timeouts, as summarized in the table below.

	$n = 100$	120	140
CPLEX find	30	31	33
CPLEX certify	10	13	11
max D_{cp}	7.5	6.5	5.5

For example at $n = 100$, CPLEX was able to find 30 optimal solutions (of 40 instances) and to certify optimality in 10 cases. In all tests the CPLEX solution was never observed to be more than 7.5 units from optimal.

Figure 3 summarizes find times for TABU (left/red), AK (center/blue) and CPLEX (right/green) at each problem size. The bar in the middle of each box marks the median time, box endpoints mark the first and third quartiles, and the circles mark outliers beyond 1.5 times the inter-quartile range. The outliers at the high end and the symmetry of the boxes on this logarithmic y-scale indicate a significant amount of skew in the data.

The dotted line at top marks the 1800-second timeout, which CPLEX hit in 26 of 120 instances. Note that AK times more than double with each increment of n, corresponding to an exponential growth rate (not unusual in branch-and-bound algorithms). We can estimate that AK will hit the 30 minute limit at around $n = 200$.

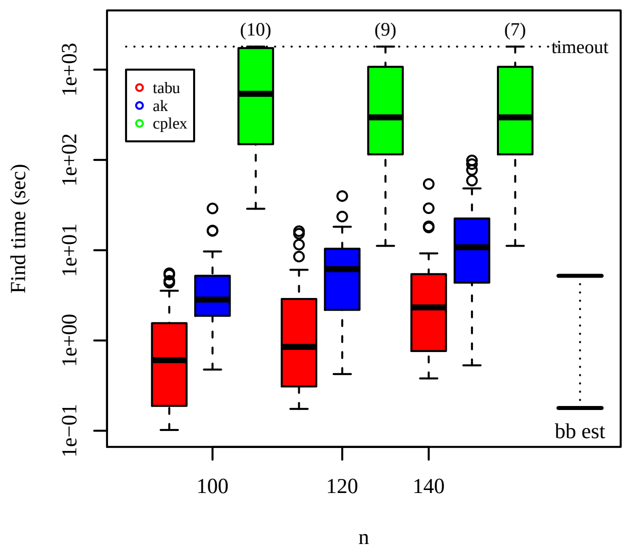


Figure 3: W2SAT software find times, and two estimates of median Blackbox find times (horizontal bars). Note the logarithmic y scale.

For reasons outlined in Section 3.1 it is very difficult to make reliable runtime comparisons between the software solvers and Blackbox. The horizontal bars on the right of Figure 3 show two estimates of median Blackbox time at $n = 140$, calculated as follows.

Since both Blackbox and TABU are based on tabu search we can compare counts of total objective function evaluations and total main loop iterations needed to converge to an optimal solution. Roughly speaking, Blackbox needs fewer iterations but performs more objective function evaluations (plus a hardware query) per iteration.

First, comparing objective function evaluations at $n = 140$ we have median(Blackbox-fevals / TABU-fevals) = 5.6. By this measure Blackbox runs roughly 5.6 times slower than TABU. This estimate is marked by the higher bar in Figure 3. Second, comparing main loop iterations we observe the median ratio of Blackbox iteration counts to TABU iteration counts is 5.2. This estimate is marked by the lower bar in the figure.

In this experiment AK, TABU, and Blackbox were able to solve all problems to optimality well within their half-hour time limits. Based on comparisons of their similar structures, we estimate that median Blackbox find times would be somewhere between 5 times faster and 5 times slower than TABU find times if they ran on similar platforms.

3.4 Quadratic Assignment Problem

In the Quadratic Assignment Problem (QAP) we are given a set of n facilities P and n locations L, a distance function $D(\ell_i, \ell_j)$ between pairs of locations, and a weight function $W(p_i, p_j)$ for pairs of facilities. The problem is to find a 1-to-1 assignment $a(p) \to \ell$ of facilities to locations, to minimize the cost function

$$QAP(a) = \sum_{p,q \in P} W(p,q) \cdot D(a(p), a(q)).$$

Our third experiment uses 33 QAP instances downloaded from a public repository called QAPLib [10]. The set has 16 different problem sizes in $n = 30 \ldots 50$. The best-known solutions for these problems are published on the website; in 23 of 33 cases the published solutions are certified optimal. Details about the instances may be found in Appendix A.

A constrained QAP problem of size n can be transformed to an unconstrained QUBO problem of size n^2 by introducing a penalty function that enforces 1-to-1 assignments. This transformation expands problem sizes to $n_s = 900 \ldots 2500$, and allows the possibility that infeasible solutions may be returned.

Since $n = 439$ is the largest problem the V5 hardware chip can handle, our Blackbox tests use a more compact transformation with an objective function that cannot be represented asa QUBO problem. Under this transformation Blackbox problem sizes are in $n_b = 178 \ldots 395$. Thus instances presented to Blackbox are roughly 5.5 times smaller than instances presented to the software solvers. (It is possible to apply the more compact transformation to TABU and CPLEX but not AK. Ongoing tests have not been completed in time for this report.)

As before, we ran software solvers with 30 minute timeouts and Blackbox to 10^7 objective function evaluations. Computation times for this experiment totaled 6 days for Blackbox and just over 14 CPU-days for the software.

In this test all software solvers ran to full timeouts and neither AK nor CPLEX was able to certify optimality. The table below compares solutions found by each. The first column (NA) shows that AK failed to return any solution at all about 10 of 33 cases. The second column (Infeasible) shows that CPLEX timed out while working on infeasible solutions in 5 cases. The third column shows the number of times each solver found the best solution among these four: here Blackbox had the best success rate overall. Of 9 times when TABU found the best solution, it tied Blackbox in 4 cases and beat Blackbox in 5 cases. The last column shows the number of times each solver found the best published solution; in all cases these known to be optimal.

	NA	Infeasible	Best	Best Pub
Blackbox	0	0	28	6
AK	10	0	1	1
CPLEX	0	5	2	2
TABU	0	0	9	5

For given instance, let P be the best published solution cost and let S_x be the best solution cost found by solver x. Figure 5 shows ratios $R_x = S_x/P$ for each solver on each instance (ordered by increasing size). The dotted line marks $R_x \geq 1.5$. Cost ratios above the line are printed (not to scale): the higher row of numbers (blue) shows ratios for AK, with N marking the 10 non-solutions; the lower row of numbers (green) shows ratios returned by CP. In all cases Blackbox found solutions within $R_{bb} = 1.18$; TABU found solutions within $R_{tb} = 1.20$ in all but two cases.

In this test Blackbox found best solutions in 28 of 33 cases, although TABU found better solutions in 5 cases and matched Blackbox in 4 cases. Blackbox and Tabu matched

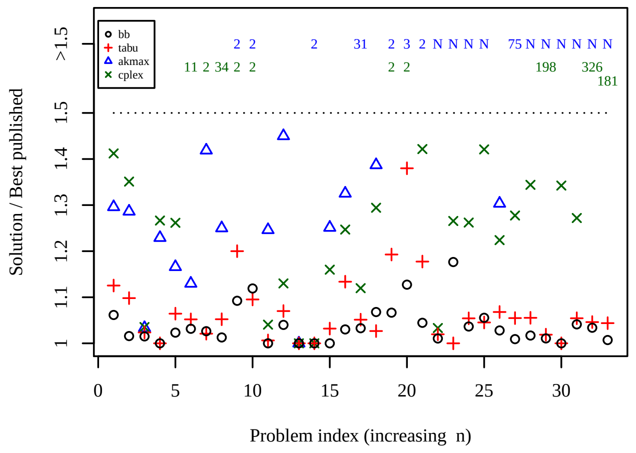


Figure 4: QAP relative solution quality. The y-axis is the ratio of solver solution to best published solution. Points above the line are not to scale (labeled by ratio); "N" denotes ten cases where AK failed to return a solution.

the best published solutions 6 and 5 times, respectively. Solutions were generally within 20 percent of published results, with Blackbox performing slightly better than TABU.

4. COMPARING V5 AND V6

This section looks at performance profiles of two hardware chips named V5 and V6 that became operational in September and December 2012, respectively. We use the same random QUBO instances as described in Section 3.2.

Recall that theoretically, computation time depends on the spectral gap g, which depends on input properties including n as well as on algorithm parameters. It not feasible to calculate g directly, so in normal use the hardware computation time (exclusive access time) is fixed and (nearly) constant in n. Instead the time is $T = t_1 + kt_2$, where t_1 is setup time per instance, t_2 is sampling time, and k is the number of solutions sampled per instance.

Figure 5 shows how sample solutions returned by V5 are distributed, for 20 instances at $n = 439$. A number plotted at position y is a count of how many times solutions with cost y were observed in the sample. For example in the first instance V5 found best-cost solutions (−822) seven times and second-best solutions (−820) 25 times in this sample of $k = 1000$. (In this problem formulation solution costs are even-valued). In many cases the sample distribution is clustered near its minimum value, with no gaps at the low end.

Let S be the best solution cost found in a sample of size $k = 1000$, and let N be the number of times S appears in the sample. Figure 6 shows how N varies with problem size n. The points are jittered by adding random noise to x-coordinates to make overlaps visible. The line connects mean values of N for each sample: at $n = 32$ nearly all of the samples are best solutions S and at $n = 439$ about 25 percent of a sample holds S on average.

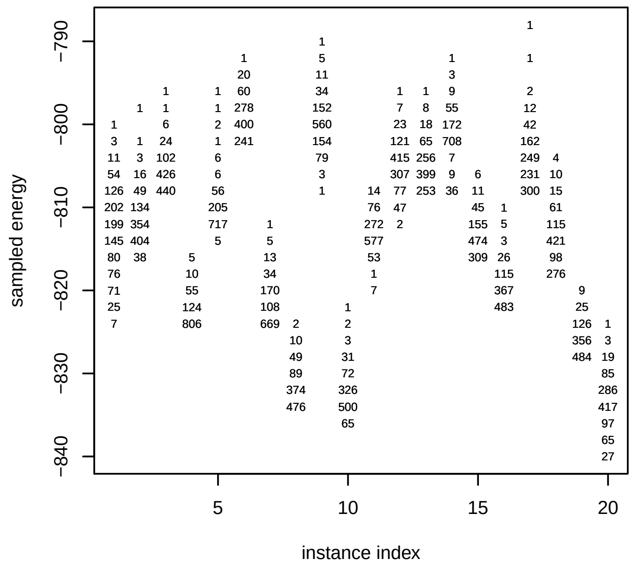


Figure 5: Distribution of V5 solution costs, n=439, 20 instances. A number at position y gives the total number of samples (out of 1000) that returned solutions with cost y.

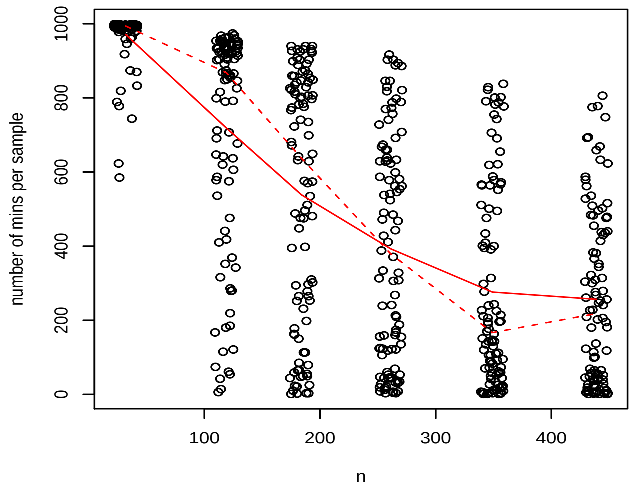


Figure 6: V5: Number of samples N holding the best answer S, in $k = 1000$ samples for each of 1000 inputs. The solid red line shows means and the dotted red line shows medians at each input size.

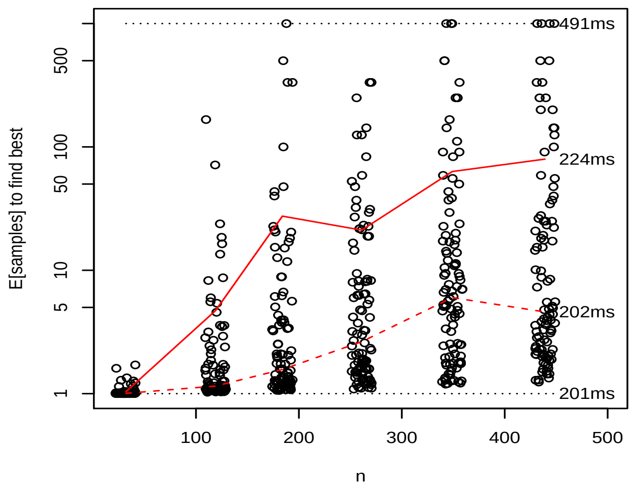


Figure 7: V5: Expected number of samples k^* to observe S (log scale). The solid red line shows means k^* at each problem size; the dotted red line shows medians. The top and bottom lines mark $k = 1000, 1$. Exclusive access times appear at the right.

Of course the hardware succeeds if just one minimum cost solution is returned. We can estimate the expected number of independent samples k^* needed to first observe S as follows. Sampling can be modeled as a Bernoulli trial with success probability equal to $p = N/k$. Therefore the expected number of samples needed to observe the first success is described by a geometric distribution with mean $1/p$. For example, if S appears 200 times in a sample of size 1000 we expect to observe it within the first $k^* = 5 = 1000/200$ samples on average.

Figure 7 shows how k^* varies with n. The red solid and dotted lines show mean and median k^*, respectively, for 1000 instances at each problem size (note the log y scale). Corresponding V5 computation times are shown on the right side of the chart, together with baseline times for $k = 1$ and 1000 samples. For half the inputs at $n = 439$, we could take just 5 rather than 1000 samples to observe the same minimum cost S. On average just under 100 samples (224ms) are needed to find the same solution as 1000 samples (491ms).

These time estimates are based on $t_1 = 201$ms, $t_2 = 0.29$ms for V5. Note that these are maximum times t_1 and t_2 measured independently over several tests at varying input sizes. Although this works out to 491ms at $k = 1000$, in fact no single test had exclusive access time above 430ms.

Figures 8 and 9 present analogous results for a newer V6 hardware chip that has 502 working qubits. Figure 8 shows the proportion N/k of best solutions found, for larger problems $n = 257, \ldots 502$, and larger sample sizes k. Figure 9 shows the expected number of samples needed to find S, with corresponding exclusive access times on the right side. V6 has $t_1 = 36$ms overhead and $t_2 = 0.126$ms sampling time. Thus V6 is 5.6 times faster than V5 at $k = 1$ and 3 times faster at $k = 1000$.

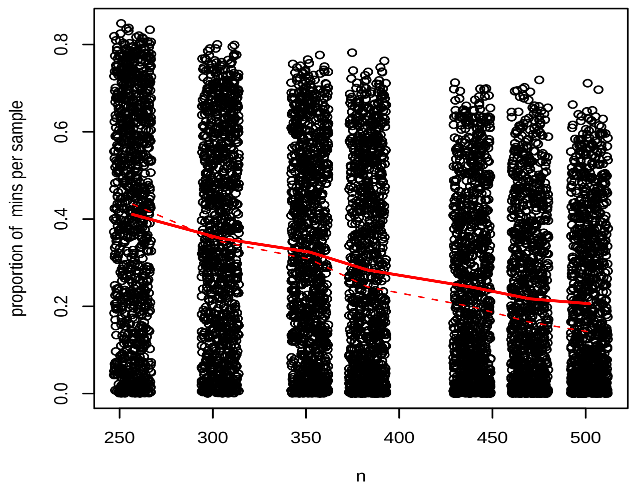


Figure 8: V6: Proportion of samples N/k holding the best answer S. Here $n = 257, 304, 352, 383, 439, 470, 502$. We have $k = 2000$ (at $n \leq 352$), $k = 5000$ (at $383 \leq n \leq 439$) and $k = 10000$ (at $n \geq 470$). The solid red line shows means and the dotted red lines shows medians.

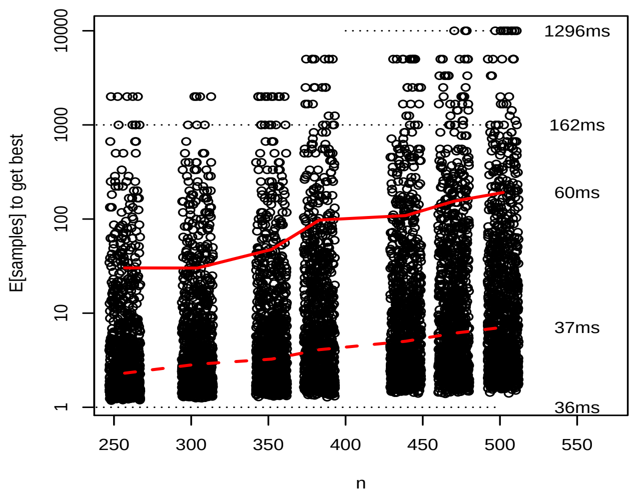


Figure 9: Expected number of samples to observe S. The solid red and dotted red lines show mean and medians over all inputs. The horizontal lines mark $k = 1$, 1000, and 10,000 samples. Exclusive access times are shown on the right.

5. CONCLUSIONS AND FUTURE WORK

This paper reports on the first experiments we know of to compare performance of a quantum annealing system to conventional software approaches, using instances from three NP-Hard optimization problems. In all three experiments the V5 hardware or its hybrid counterpart Blackbox found solutions that tied or bettered the best solutions f ound by software solvers. Performance was especially impressive on instances that can be solved directly in hardware. On the largest problem sizes tested, the V5 chip found optimal solutions in less than half a second, while the best software solver, CPLEX, needed 30 minutes to find all optimal solutions.

We have also carried out some tests to compare V6 to the software solvers; very preliminary results suggest that on QUBO instances with $n = 503$, the hardware can find optimal solutions around 10,000 times faster than CPLEX.

Our results have turned up many ideas for future experiments, including expanding these tests to a wider set of problem domains and solvers; looking at how problem transformations affect performance; developing good algorithms and heuristics for minor-embedding in hardware graphs; and investigating ideas for auto-tuning Blackbox. There is also much work to be done in understanding how performance of hardware chips like V5 and V6 depends on input parameters, including size, density, objective function, and weight distributions and scale.

It would of course be interesting to see if highly tuned implementations of, say, tabu search or simulated annealing could compete with Blackbox or even QA on QUBO problems; some preliminary work on this question is underway.

The open questions about the fundamental capabilities of adiabatic quantum computers, algorithms, and models of computation are myriad, too many to be listed here.

6. REFERENCES

[1] D. Aharonov, W. van Dam, J. Kempe, Z. Landau, S. Loyd, and O. Regev, "Adiabatic quantum computation is equivalent to standard quantum computation," *SIAM Journal of Computing,* Vol 37, Issue 1, p 166-194, 2007. The conference paper appeared in *Proceedings of 45th FOCS,* p. 42-51, 2004.

[2] Boris Altshuler, Hari Krovi, and Jeremie Roland, "Adiabatic quantum optimization fails for random instances of NP-Complete problems," arXiv:0908.2782, December 2009.

[3] Joseph Argelich, Chu Min Li, Felip Manyà and Jordi Planes (organizers), Max-SAT 2012: Seventh Max-SAT Evaluation, `maxsat.udl.cat`. The testbed used in our experiments may be found in the Weighted Max-Sat/Random category.

[4] Zhengbing Bain, Fabian Chudak, William G. Macready, Lane Clark, and Frank Gaitan, "Experimental determination of Ramsey numbers with quantum annealing," arXiv:1201.1842v2 [quant-ph] 10 Jan 2012.

[5] Jacob Biamonte and Peter J. Love, "Realizable Hamiltonians for universal quantum computers," Phys. Rev. A 78 012352 (2008).

[6] Sergio Boixo, Tameem Albash, Federico M. Spedalieri, Nicholas Chancellor, and Daniel A. Lidar, "Experimental signature of programmable quantum annealing," arXiv:1212.1739 [quant-ph], 7 December 2012.

[7] Max Born and Vladimir Fock, "Beweis des Adiabatensatzes," *Seitschrift für Physik A*, 51 (3-4): pp 165-189.

[8] Endre Boros, Peter Hammer, and Gabriel Tavares, "Local search heuristics for quadratic unconstrained binary optimization (QUBO)," *Journal of Heuristics*, 13:99-132, 2007.

[9] J. Brooke, D. Bitko, T. F. Rosenbaum and G. Aeppli, "Quantum annealing of a disordered magnet," *Science* 30 April 1999: Vol. 284, no. 5415, pp. 779-781. 284 779 (1999). `www.sciencemag.org/content/284/5515/7794.abstract`.

[10] R.E. Burkard, E. Çela, S. E. Karisch, and F. Rendl, QAPLIB Quadratic Assignment Problem Library: Problem Instances and Solutions. `www.seas.upenn.edu/qaplib/inst.html`.

[11] A. Childs, E. Farhi, and J. Preskill, "Robustness of adiabatic quantum computation," *Physical Review A*, 65:012322, 2002.

[12] Adrian Cho, "Controversial Computer Is at Least a Little Quantum Mechanical," *Science*, 13 May 2011.

[13] Vicky Choi, "Minor-embedding in adiabatic quantum computation: I The parameter setting problem," *Quantum Inf. Process* 7: pp 193-209, 2008.

[14] Vicky Choi, "Adiabatic quantum algorithms for the NP-Complete maximum-weight independent set, exact cover, and 3SAT problems," arXiv:1004.2226v1 [quantum-ph] 13 April 2010.

[15] Niel G. Dickson and M. H. S. Amin, "Does adiabatic quantum optimization fail for NP-Complete problems?", arXiv:1010.0669v3 [quantum-ph] 11 January 2011.

[16] E. Farhi, J. Goldstone, S. Gutmann, "A numerical study of the performance of a quantum adiabatic evolution algorithm for satisfiability," arXiv:quant-ph/0007071v1.

[17] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, "Quantum computation by adiabatic evolution," arXiv:quant-ph/0001106, 2000.

[18] E. Farhi, J. Goldstone, S. Gutman, J. Lapan, A. Lundgren, and D. Preda, "A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem," *Science*, 292(5516): 472-476, 2001.

[19] E. Farhi, J. Goldstone, D. Gosset, S. Gutmann, H. B. Meyer, and P. Shor, "Quantum adiabatic algorithms, small gaps, and different paths," arXiv.org:quant-ph/0909.4766, 2009.

[20] Friedman et al., "Quantum superposition of distinct macroscopic states," Nature (406) 2000.

[21] L. K. Grover, "A fast quantum mechanical algorithm for database search," *Proceedings of the 28th STOC* , May 1996, pp. 212-222

[22] R. Harris et al., "Experimental demonstration of a robust and scalable flux qubit," arXiv:0909.4321 v1 [cond-mat.supr-con] , 24 Sept 2009.

[23] IBM ILOG CPLEX Optimizer, `www-01.ibm.com/software/integration/optimization/cplex-optimizer`.

[24] Enej Ilevski, “Adiabatic quantum computation,” manuscript (lecture notes), Department for Physics, University of Ljubljana, 2010. Email: enej.ilievski@student.fmf.uni-lj.si

[25] M. W. Johnson et al., “Quantum annealing with manufactured spins,” *Nature* 473, 194-198, 12 May 2011.

[26] Adrian Kügel - Dipl. Inf. Universität Ulm, `www.uni-ulm.de/in/theo/m/kuegel`. See also ref [3].

[27] METSlib: An Open Source Tabu Search Metaheuristic framework in modern C++, `projects.coin-or.org/metslib`.

[28] Harmut Neven, Vasil S. Denchev, Marshall Drew-Brook, Jiayong Zhang, William G. Macready, and Geordie Rose, “NIPS 2009 Demonstration: Binary Classification using a hardware implementation of quantum annealing,” NIPS 2009.

[29] Alejandro Perdomo-Ortiz, Neil Dickson, Marshall Drew-Brook, Geordie Rose and Alá Aspuru-Guzik, “Finding low-energy conformations of lattice protein models by quantum annealing,” *Nature Scientific Reports* 2, article number 571, 13 August 2012.

[30] Giuseppe E. Santoro, Roman Martoňák, Erio Tosatti, and Roberto Car, “Theory of quantum annealing of an Ising spin glass,” *Science* 29 March 2002: Vol 295 no. 5564, pp 2427-2430. `www.sciencemag.org/content/295/5564/2424.abstract`.

[31] Rolando D. Somma and Sergio Bioxo, “Spectral gap amplification,” v3, arXiv:1110.2494 [quant-ph] 30 March 2012.

[32] Gabriel Tavares, *New Algorithms for Quadratic Unconstrained Binary Optimization (QUBO) with Applications in Engineering and Social Sciences*, PhD dissertation, Rutgers University, 2008.

[33] W. van Dam, M. Mosca, and U. Vazirani, “How powerful is adiabatic quantum computation?” in *Proceedings 42nd FOCS*, pp 279-287, 2001.

[34] W. van Dam and U. Vazirani, “Limits on quantum adiabatic optimization,” Unpublished manuscript 2001.

Appendix A: QAPLib Problems

file	n	Best Published	Is OPT	Our Best
esc32a	32	130	opt	142
esc32b	32	168	opt	188
esc32c	32	642	opt	*642
esc32d	32	200	opt	208
esc32e	32	2	opt	*2
esc32g	33	6	opt	*6
esc32h	32	438	opt	*438
kra30a	30	88 900	opt	94 370
kra30b	30	91 420	opt	92 860
kra32	32	88 700	opt	91 380
lipa30a	30	13 178	opt	13 376
lipa30b	30	151 426	opt	*151 426
lipa40a	40	31 538	opt	31 869
lipa40b	40	476 581	opt	560 670
lipa50a	50	62 093	opt	62 757
lipa50b	50	1 210 244	opt	*1 210 244
nug30	30	6 124	opt	6 266
sko42	42	15 812		15 956
sko49	49	23 386		23 782
ste36a	36	9 526	opt	10 160
ste36b	36	15 852	opt	17 870
ste36c	36	8 239 110	opt	8 605 808
tai30a	30	1 818 146		1 875 566
tai30b	30	637 117 113	opt	653 908 320
tai35a	35	2 422 002	opt	2 501 550
tai35b	35	283 315 445		302 555 410
tai40a	40	3 139 370		3 254 090
tai40b	40	637 250 948		672 576 232
tai50a	50	4 938 796		5 142 226
tai50b	50	458 821 517		474 357 420
tho30	30	149 936	opt	151 854
tho40	40	240 516		247 236
wil50	50	48 816		49 164

A list of 33 instances from QAPLib used in our tests. The rightmost column shows the best answers found by Blackbox (28 instances) and Tabu (5 instances). Solutions that match the best published solutions are marked with *.