Finding Maximum Cliques on a Quantum Annealer

Guillaume Chapuis
Los Alamos National Laboratory

Hristo Djidjev (PI)
Los Alamos National Laboratory

Georg Hahn
Imperial College, London, UK

Guillaume Rizk
INRIA/Irisa, Rennes Cedex, France

Abstract

This paper assesses the performance of the D-Wave 2X (DW) quantum annealer for finding a maximum clique in a graph, one of the most fundamental and important NP-hard problems. Because the size of the largest graphs DW can directly solve is quite small (usually around 45 vertices), we also consider decomposition algorithms intended for larger graphs and analyze their performance. For smaller graphs that fit DW, we provide formulations of the maximum clique problem as a quadratic unconstrained binary optimization (QUBO) problem, which is one of the two input types (together with the Ising model) acceptable by the machine, and compare several quantum implementations to current classical algorithms such as simulated annealing, Gurobi, and third-party clique finding heuristics. We further estimate the contributions of the quantum phase of the quantum annealer and the classical post-processing phase typically used to enhance each solution returned by DW. We demonstrate that on random graphs that fit DW, no quantum speedup can be observed compared with the classical algorithms. On the other hand, for instances specifically designed to fit well the DW qubit interconnection network, we observe substantial speed-ups in computing time over classical approaches.

1 Introduction

The emergence of the first commercially available quantum computers by D-Wave Systems, Inc. (D-Wave, 2016b) has provided researchers with a new tool to tackle NP-hard problems for which presently, no classical polynomial-time algorithms are known to exist and which can hence only be solved approximately (as well as exactly for very small instances). One such computer is D-Wave 2X, which we denote here as DW. It has roughly 1000 units storing quantum information, called *qubits*, which are implemented via a series of superconducting loops on the D-Wave chip. Each loop encodes both a 0 and 1 (or, alternatively, -1 and +1) value at the same time through two superimposed currents in both clockwise and counter-clockwise directions until the annealing process has been completed and the system turns classical (Johnson et al., 2011; Bunyk et al., 2014). The device is designed to minimize an unconstrained objective function consisting of a sum of linear and quadratic binary contributions, weighted by given constants. Specifically, it aims at minimizing the *Hamiltonian*

$$H = H(x_1, \dots, x_N) = \sum_{i \in V} a_i x_i + \sum_{(i,j) \in E} a_{ij} x_i x_j \tag{1}$$

with variables $x_i \in \{0, 1\}$ and coefficients a_i, $a_{ij} \in \mathbb{R}$, where $V = \{1, \dots, N\}$ and $E = V \times V$ (King et al., 2015). This type of problem is known as a *quadratic unconstrained binary optimization (QUBO)* problem. When the coefficients a_i and a_{ij} are encoded as capacities of the *couplers* (links) connecting the qubits, H describes the quantum energy of the system: During annealing, the quantum system consisting of the qubits and couplers tries to settle in its stable state, which is one of a minimum energy, i.e., of a minimum value of H. In order to solve a given optimization problem, one has to encode it as a minimization problem of a Hamiltonian of type (1).

Similarly to the random moves considered in a simulated annealing classical algorithm, a quantum annealer uses quantum tunneling to escape local minima and to find a low-energy configuration of a physical system corresponding to an optimization problem. Its use of quantum superposition of 0 and 1 qubit values enables a quantum computer to consider and manipulate all combinations of variable values simultaneously, while its use of quantum tunneling allows it to avoid hill climbing, giving it a potential advantage over a classical computer. However, it is unclear if this potential is realized by the current quantum computing technology, and by the DW computer in particular, and whether DW provides any quantum advantage over the best available classical algorithms (Rønnow et al., 2014; Denchev et al., 2016).

This article tries to answer these questions for the problem of finding a maximum clique in a graph, an important NP-hard problem with multiple applications including network analysis, bioinformatics, and computational chemistry. Given an undirected graph $G = (V, E)$, a *clique* is a subset S of the vertices forming a complete subgraph, meaning that any two vertices of S are connected by an edge in G. The clique size is the number of vertices in S, and the maximum clique problem is to find a clique with a maximum number of vertices in G (Balas and Yu, 1986).

We will consider formulations of MC as a QUBO problem and study its implementations on DW using different tools and strategies. We will compare these implementations to several classical algorithms on different graphs and try to determine whether DW offers any quantum advantage/speedup.

The article is organized as follows. Section 2 starts by introducing the qubit architecture on the D-Wave chip as well as available software tools. We also describe a QUBO formulation of MC together with its implementations on DW and present methods for dealing with graphs of sizes too large to fit onto the DW chip. Section 3 presents an experimental analysis of the quantum software tools and a comparison with several classic algorithms, both for graphs small enough to fit DW directly as well as for larger graphs for which decomposition approaches are needed. We conclude with a discussion of our results in Section 4.

In the rest of the paper, we denote a graph as $G = (V, E)$, where $V = \{1, \ldots, n\}$ is a set of n vertices and E is a set of undirected edges.

2 Solving MC on Dwave

This section introduces the DW chip architecture and briefly presents three tools provided by D-Wave Inc. to submit quadratic programs to the quantum computer. We also introduce the QUBO formulation of MC needed to submit an MC instance to DW. The section concludes with an algorithmic framework designed to solve instances of MC which are not embeddable on DW.

2.1 DW hardware and software

2.1.1 The Qubit architecture

DW operates on roughly 1000 qubits. The precise number of available qubits varies from machine to machine (even of the same type) due to manufacturing errors making some of the qubits inoperative. The qubits are connected using a specific type of network called *Chimera* graph, $C_{12,12,4}$ (see Fig. 1), comprised of a lattice of 12×12 cells, where each cell is a 4×4 complete bipartite graph. DW can

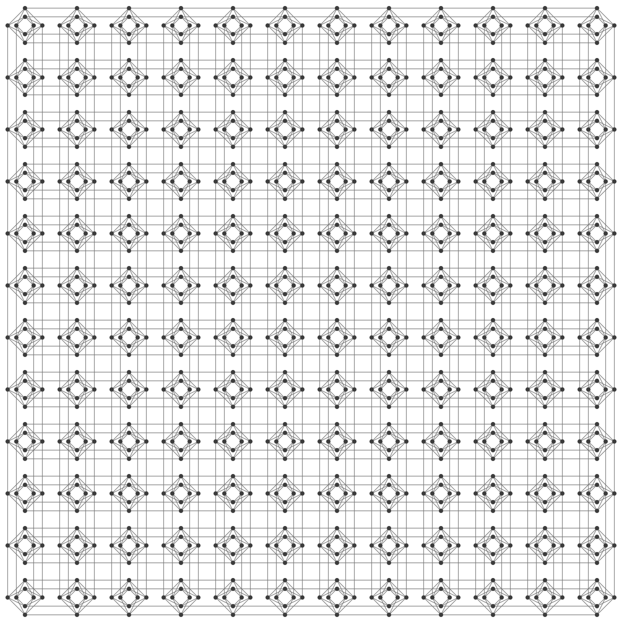


Figure 1: The Chimera $C_{12,12,4}$ graph of 1152 vertices (qubits) and 3360 edges (couplers). LANL's D-Wave 2X chip has usable only 1095 qubits and 3061 couplers due to manufacturing defects.

naturally solve Ising problems where non-zero quadratic terms are represented by an edge in the Chimera graph.

The particular architecture of the qubits implies two important consequences: First, the chip design actually only allows for direct pairwise interactions between two qubits which are physically adjacent on the chip. For pairwise interactions between qubits not physically connected, a *minor embedding* of the graph describing the non-zero structure of the Hamiltonian matrix into the Chimera type graph is needed, which maps a logical variable into one or several physical qubits on the chip. Minor embeddings are hence necessary to ensure arbitrary connectivity of the logical variables in the QUBO. The largest complete graph that the DW can embed in theory has $1 + 4 \cdot 12 = 49$ vertices. In practice, the largest embeddable graph is slightly smaller ($n \approx 45$) due to missing qubits arising in the manufacturing stage.

When more than one qubit is used to represent a variable, that set of qubits is called a *chain*. The existence of chains has two vital consequences, which will play an important role in the analyses of Section 3. On the one hand, the need for chains uses up qubits, which would otherwise be available to represent more variables in the quadratic program, thereby reducing the maximum problem sizes that can directly be solved on DW. This is the reason for the relatively small sizes of $N = 45$ for QUBO problems (1) that fit on DW when the corresponding Hamiltonians are dense (contain nearly all quadratic terms), despite the fact that more than 1000 qubits are available in DW. On the other hand, due to the imperfections of the quantum annealing process caused by environmental noise, limited precision, and other shortcomings, solutions returned by D-Wave do not always correspond to the minimum energy configuration. In the case of chains, all qubits in a chain encode the same variable in (1) and hence should have the same value, but for the reasons outlined above that may not be the case. This phenomenon is called a *broken chain*, and it is not clear which value should be assigned to a variable if its chain is broken. Clearly, chains can be ensured to not break by increasing their coupler weights, but as we will see in the next section this may significantly reduce the accuracy of the solver.

2.1.2 D-Wave solvers

D-Wave Inc. provides several tools that help users submit their QUBO problems to the quantum processor, perform the annealing, apply necessary pre- and post-processing steps, and format the output. This section briefly describes several such tools used in this article.

Sapi Sapi stands for Solver API and provides the highest level of control one can have over the quantum annealer. It allows the user to compute minor embeddings for a given Ising or QUBO problem, to choose the number of annealing cycles, or to specify the type of post-processing. Sapi interfaces for the programming languages *C* and *Python* are available. One can also use a pre-computed embedding of a complete 45-vertex graph, thus avoiding the need to run the slow embedding algorithm.

QBsolv QBsolv is a tool that can solve problems in QUBO format of size that cannot natively fit onto DW. Larger problems (with more variables or more connections than can map onto the corresponding Chimera graph) are analyzed by a hybrid algorithm, which identifies a small number of significant rows and columns of the Hamiltonian. It then defines a QUBO on that subset of variables that fit DW, solves it, and extends the found solution to a solution of the original problem.

QSage In contrast to Sapi or QBsolv, QSage is a blackbox hybrid solver which does not require a QUBO or Ising formulation as input. Instead, QSage is able to minimize any function operating on a binary input string of arbitrary size. For this it uses a tabu search algorithm enhanced with DW-generated low-energy samples near the current local minimum. To ensure that also input sizes larger than the DW architecture can be processed, QSage optimizes over random substrings of the input bits.

2.2 QUBO formulations of MC

Recall that a QUBO problem can be written as

$$\underset{x_i \in \{0,1\}}{\text{minimize}}\ H = \sum_{1 \le i < j \le N} a_{ij} x_i x_j, \tag{2}$$

where the weights a_{ij}, $i \neq j$, are the quadratic terms and a_{ii} are the linear terms (since $x_i^2 = x_i$ for $x_i \in \{0,1\}$).

There are multiple ways to formulate the MC problem as a QUBO. One of the simplest is based on the equivalence between MC and the maximum independent set problem. An independent set S of a graph H is a set of vertices with the property that for any two vertices $v, w \in S$, v and w are not connected by an edge in H. It is easy to see that an independent set of $H = (V, \overline{E})$ defines a clique in graph $G = (V, E)$, where $\overline{E}$ is the complement of set E. Therefore, looking for the maximum clique in G is equivalent to finding the maximum independent set in H. The corresponding constraint formulation for MC is

$$\begin{aligned} &\underset{x_i \in \{0,1\}}{\text{maximize}} && \sum_{i=1}^{N} x_i \\ &\text{subject to} && \sum_{(i,j) \in \overline{E}} x_i x_j = 0, \end{aligned} \tag{3}$$

where $G = (V, E)$ is the input graph and $\overline{E}$ is the complement of E. The equivalent unconstrained (QUBO) minimization of (3), written in the form (2), is

$$H = -A \sum_{i=1}^{N} x_i + B \sum_{(i,j) \in \overline{E}} x_i x_j, \tag{4}$$

where one can determine that the coefficients/penalties A and B can be chosen as $A = 1$, $B = 2$ (see Lucas (2014)). A disadvantage of the formulation (4) is that H contains an order of N^2 quadratic terms even for sparse graphs G, which limits the size problems for which MC can be directly solved on DW.

2.3 Solving larger MC instances

To solve the MC problem on an arbitrary graph, we develop several algorithms that reduce the size of the input graph by removing vertices and edges that do not belong to a maximum clique and/or split it into smaller subgraphs of at most 45 vertices, the maximal size of a complete graph embeddable on DW. Let $G(V, E)$ be a connected graph of n vertices.

2.3.1 Extracting the k-core

The k-core of graph $G = (V, E)$ is the maximal subgraph of G whose vertices have degrees at least k. It is easy to see that if G has a clique C of size $k+1$, then C should be also a clique of the k-core of G (since all vertices in a k-clique have degrees $k-1$). Therefore, finding a maximum clique of size no more than $k+1$ in the original graph G is equivalent to finding such clique in the k-core of G (which might be a graph of much smaller size). One can compute the k-core iteratively by picking a vertex v of degree less than k, removing v and its adjacent edges, updating the degrees of the remaining vertices, and repeating while such vertex v exists. The algorithm can be implemented in optimal $O(|E|)$ time (Batagelj and Zaversnik, 2011).

2.3.2 Graph partitioning

This divide-and-conquer approach aims at dividing G into smaller subgraphs, solve the MC problem in each of these subgraphs, and combine the subproblem solutions into a solution of the original problem. If one uses standard (edge-cut) graph partitioning, which divides the vertices of the graph into a number of roughly equal parts so that the number of *cut edges*, or edges with endpoints in different parts, is minimized, then the third step, combining the subgraph solutions, will be computationally very expensive. Instead, we will use CH-partitioning, recently introduced in Djidjev et al. (2016).

In *CH-partitioning*, there are two levels of dividing the vertices of G into subsets. In the *core partitioning*, the set V of vertices is divided into nonempty *core* sets $C_1, \ldots, C_s$ such that $\bigcup_i C_i = V$ and $C_i \cap C_j = \emptyset$ for $i \neq j$. There is one *halo* set H_i of vertices for each core set C_i, defined as the set of neighbor vertices of C_i that are not from C_i. We define the cost of the CH-partitioning $\mathcal{P} = (\{C_i\}, \{H_i\})$ as

$$\operatorname{cost}(\mathcal{P}) = \max_{1 \leq i \leq s} (|C_i| + |H_i|). \tag{5}$$

The *CH-partitioning problem* is finding a CH-partitioning of G of minimum cost. The next statement shows how CH-partitions can be used for solving MC in larger graphs.

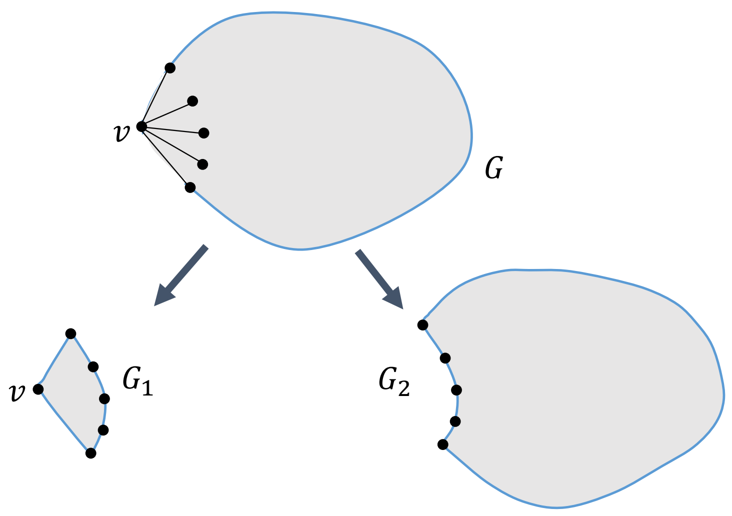


Figure 2: Illustration of the vertex splitting algorithm.

Proposition 1. *Given a CH-partitioning* $(\{C_i\}, \{H_i\})$ *of* G*, the size of the maximum clique of* G *is equal to* $\max_i\{k_i\}$*, where* k_i *is the size of a maximum clique of the subgraph of* G *induced by* $C_i \cup H_i$*.*

Using Proposition 1, solutions to all subproblems defined by a CH-partitioning can be combined into a solution of the original problem at a cost of only $O(s) = O(n)$.

One may conjecture that increasing s in (5) will always reduce the cost, but this is not always the case. If the minimum cost is achieved for $s = 1$, or if some of the parts of the partition are still too large, then the method in the next subsection might be applied.

2.3.3 Vertex splitting

This method is similar to a special case of the previous one, obtained by choosing $s = 2$, letting C_1 contain only a single vertex v, and letting C_2 contain all other vertices $V \setminus \{v\}$. Moreover, while the halo H_1 of C_1 is defined as above, we set $H_2 = \emptyset$. Because C_1 consists of a single vertex, we call such partitioning a *vertex-splitting partitioning*, see Fig. 2. The cost of such a partitioning is equally given by (5).

Proposition 2. *Given a vertex-splitting partitioning of* G*,* $(\{C_1, C_2\}, \{H_1, H_2 = \emptyset\})$*, the size of the maximum clique of* G *is equal to* $\max\{k_1, k_2\}$*, where* k_i *is the size of a maximum clique of the subgraph of* G *induced by* $C_i \cup H_i$*.*

Since $H_2 = \emptyset$, vertex splitting can be used in cases where CH-partitioning fails. Moreover, if there is a vertex of degree less than $n-1$, this method will always create subproblems of size smaller than the original one. However, the total number of subproblems resulting from the repeated use of this method can be too large. A more efficient algorithm can be produced if all the above methods are combined.

2.3.4 Combining the three methods

We use the following algorithm to decompose a given input graph G into smaller MC instances fitting the DW size limit. We assume that the size $k + 1$ of the maximum clique is known; if not, we use the below procedure in a binary-tree search fashion to determine that value. (This increases the running time by a factor $O(\log k) = O(\log n)$ only.) We have also an implementation that, instead of "guessing" the *exact* k, uses lower bounds on k determined by the size of the largest clique found so far.

The algorithm works in two phases.

First, we apply the k-core algorithm on the input graph and then CL-partitioning on the resulting k-core. Consequently, we keep a list L of subgraphs (ordered by their number of vertices), which is initialized with the output of the CL-partitioning step. In each iteration and until all produced subgraphs fit the (DW) size limit, we choose a vertex v from the largest subgraph sg, extract the subgraph ssg induced by v and its neighbors and remove v from sg. The k-cores of the two subgraphs produced at this iteration are then inserted into L.

Second, we compute the maximum clique on DW for any subgraph in L of size small enough.

We also apply another reduction approach, which we refer to as edge k-core, as follows: We choose a random vertex v in sg and remove all edges (v, e) satisfying $|N(v) \cap N(e)| < l - 2$ (here $N(v)$ denotes the set of neighbor vertices of v), as such edges cannot be part of a clique with size larger than l.

Lastly, we describe our procedure for choosing vertices to be removed from sg. A vertex with high degree will potentially greatly reduce the size of sg, however at the expense of also producing a large subgraph ssg. In order to maximize the impact of removing a vertex, we successively try out three choices: a vertex of highest degree, a vertex of median degree and, if necessary, a vertex of lowest degree in sg. If the vertex of lowest degree has degree $|V| - 1$, then sg is a clique: In this case, solving MC on sg can be omitted and l can be update immediately.

3 Experimental analysis

The aim of this section is to investigate if a quantum advantage for the MC problem can be detected for certain classes of input graphs. To this end, we compare the DW solvers of Section 2.1.2 to classical ones on various graph instances–from random small graphs that fit the DW chip to graphs tailored to perfectly fit DW's Chimera architecture. We also evaluate our graph splitting routine of Section 2.3 on large MC instances. First we briefly describe classical solvers that will be used in the comparison.

3.1 Classical solvers

Apart from the tools provided by D-Wave Inc., we employ classical solvers in our comparison, consisting of: A simulated annealing algorithm working on the Ising problem (SA-Ising), a simulated annealing algorithm specifically designed to solve the clique problem (*SA-clique*, see Geng et al. (2007)), softwares designed to find cliques in heuristic or exact mode (the *Fast Max-Clique Finder* fmc, see Pattabiraman et al. (2013)), *pmc* (see Rossi et al. (2013)), and the Gurobi solver (Gurobi Optimization, Inc., 2015).

SA-Ising This is a simulated annealing algorithm working on an Ising problem formulation. The initial solution is a random solution, and a single move in the simulated algorithm is the flip of one random bit.

SA-clique We implemented a simulated annealing algorithm specifically designed to find cliques, as described in Geng et al. (2007). As SA-clique only finds cliques of a user-given size m, we need to apply a binary search on top of it to find the maximum clique size. Its main parameter is a value α controlling the geometric temperature update in each step (that is, $T_{n+1} = \alpha T_n$). A default choice is $\alpha = 0.9996$. A value closer to 1 will yield a better solution but will increase computation time.

Fast Max-Clique Finder (fmc, pmc) These two algorithms are designed to efficiently find a maximum clique for a large sparse graph. They provide exact and heuristic search modes. We use version 1.1 of software *fmc* (Pattabiraman et al., 2013) and *pmc* (github commit 751e095) (Rossi et al., 2013).

Post-processing heuristics alone (PPHa) The DW pipeline includes a post-processing step: First, if chains exist, a majority vote is applied to fix any broken chains. Then a local search is performed to ensure that any solution is indeed a local minimum (the raw solutions coming from DW might not be in a local minimum, see D-Wave (2016a)). For a given solution coming out of the pipeline, one might wonder what the relative contributions of DW and of the post-processing step are. For some small and simple problems, the post processing step *alone* might be able to find a good solution.

We try to answer this issue by solely applying the post-processing step, and by comparing the result with the one obtained by quantum annealing. However, post-processing by DW runs on the DW server and is not available separately.

To enable us to still use the DW post-processing alone, we employ the following procedure. We set a very high absolute chain strength (e.g., 1000 times greater than the largest weight in our Ising problem), and turn on the *auto-scale* feature mapping QUBO weights to the interval $[-1, 1]$. Because of the limited precision of the DW hardware (DW maps all QUBO weigths to 16 discrete values within $[-1, 1]$), chain weights will be set to the minimum value -1 while all other weights will be scaled down to 0. In this way, the quantum annealer will only satisfy the chains rather than the actual QUBO we are interested in. As chains will not be connected to other chains, and as all linear terms will be zero, each chain will be assigned a random value -1 or $+1$. Applying the DW post-processing step to such a QUBO with large chain weights will therefore result in the post-processing step being called with a random initial solution. We hence expect to obtain results stemming from the post-processing step only. This method will be referred to as PPHa, *post-processing heuristic alone.*

Gurobi *Gurobi* (Gurobi Optimization, Inc., 2015) is a mathematical programming solver for linear programs, mixed-integer linear and quadratic programs, as well as certain quadratic programs.

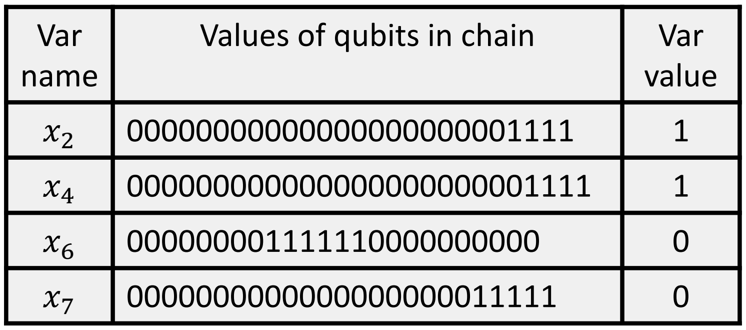
We employ *Gurobi* to solve given QUBO problems (Ising problems can be solved as well, nevertheless *Gurobi* explicitly allows to restrict the range of variables to binary inputs, making it particularly suitable for QUBO instances).

Instead of solving MC directly with Gurobi, we solve the dual problem, that is we computed a maximum independent set on the complement graph.

3.2 Small graphs with no special structure

We generate four random graphs with increasing edge densities for our experiments. We considered edge probabilities ranging from 0.3 to 0.9 in steps of 0.05.

We compare the execution times of DW using the Sapi interface and the different solvers listed in Section 2.1.2. Results are shown in Table 1. For small graphs, every solver returns a maximum clique, therefore the table shows execution times only. We can see that (a) software solvers are much



Var name	Values of qubits in chain	Var value
x_2	00000000000000000000001111	1
x_4	000000000000000000000001111	1
x_6	000000001111110000000000	0
x_7	0000000000000000000011111	0

Figure 3: The first four broken chains (out of 16) produced by DW on a test 45-vertex graph. The first column shows the name of the variable the chain corresponds to and the third column gives the correct value for that variable.

Graph	Max. clique size	Runtime [s]						
		Sapi	PPha	QBsolv	*fmc*	*pmc*	SA	Gurobi
p=0.3	5	0.15	0.15	0.05	$8 \cdot 10^{-6}$	$3 \cdot 10^{-5}$	0.15	102
p=0.5	8	0.15	0.15	0.06	$3 \cdot 10^{-4}$	$5 \cdot 10^{-5}$	0.37	38
p=0.7	13	0.15	0.15	0.04	0.002	8.10^{-5}	0.19	33
p=0.9	20	0.15	0.15	0.04	0.135	8.10^{-5}	0.28	2

Table 1: Running time on 45 vertex random graphs. The edge probability used to generate those graphs is given in the first column. Since for such small graphs, every software returned the correct solution, we only report the running times. Gurobi solves the dual problem, leading to reversed graph densities and timings.

faster than DW, with *pmc* being the fastest by several order of magnitudes; (b) DW and *PPHa* exhibit equal results and execution times. This shows that for these small graphs, even the simple software heuristic included in the DW pipeline is capable of solving the MC problem. In this case it is therefore impossible to distinguish between the contributions from the post-processing heuristic and the actual quantum annealer; (c) Gurobi finds the best solution as well (for the maximum independent set problem, thus timings decrease in the last column of Table 1), but since Gurobi is an exact solver, its running time is higher than the one of the other methods. We note that the timings for Gurobi are for finding the best solution – letting Gurobi run further to subsequently prove that a found solution is optimal requires a far longer runtime.

Moreover, in our experiments, the QUBO matrix is typically very dense, leading to long chains in the embedding. It seems that this is a difficult case for the quantum annealer, as many of these chains are broken, i.e. the physical qubits constituting the chains have different values. Therefore some processing needs to be applied to obtain valid solutions. The most simple one is to apply a majority vote, yet in our experiments this often led to sub-optimal solutions.

As an example, Figure 3 shows the first four broken chains in a typical DW execution of the MC problem on a 45 vertex graph. The chain for x_2 has more zeros and less ones than the one for x_7, yet after the DW postprocessing algorithm was applied, the variables got correct values $x_2 = 1, x_7 = 0$ (with apparently PPHa totally overriding the inferior DW solution). Our experiments with randomly assigned values to broken chains (see the discussion for PPHa in Section 3.1) similarly show that accurate solutions obtained for small graphs are often mostly due to the post-processing algorithm rather than the quantum annealing by DW.

3.3 Graphs of sizes that fit DW

In Section 3.2, we performed experiments with random graphs that can be embedded onto DW. Here we will analyze the behavior of the quantum annealer on graphs of 45 vertices, i.e. the largest graph we are sure to be able to embed.

3.3.1 Chimera-like graphs

Since on small graphs we did not observe any speedup of DW compared to the classical algorithms, we now consider graphs that fit nicely the DW architecture. The largest graph that fits DW is the Chimera graph $\mathcal{C}$, and since formulation (4) uses the complement edges, the largest graph that we can solve MC on is the complement of $\mathcal{C}$. Let $\overline{G}$ denote the complement of any graph G. Note that the graphs $\mathcal{C}$ and $\overline{\mathcal{C}}$ are not interesting for the MC problem since $\mathcal{C}$ is bipartite and hence $\overline{\mathcal{C}}$ consists of two disconnected cliques, which makes MC trivial on this graph.

Consider now the graph $\mathcal{C}_1$ obtained by contracting one random edge from $\mathcal{C}$. An *edge contraction* consists of deleting an edge (v_1, v_2) and merging its endpoints v_1 and v_2 into a new vertex v^*. With $\mathcal{N}_1$ and $\mathcal{N}_2$ the set of neighboring vertices of v_1 and v_2, the neighbors of v^* are $\mathcal{N}_1 \cup \mathcal{N}_2 \setminus \{v_1, v_2\}$. Solving the MC problem on $\overline{\mathcal{C}}_1$ requires the embedding of the complement of $\overline{\mathcal{C}}_1$ onto DW, which is $\mathcal{C}_1$. The natural embedding of $\mathcal{C}_1$ onto $\mathcal{C}$ maps v^* onto a chain of two vertices and all other vertices of $\mathcal{C}_1$ onto single vertices of $\mathcal{C}$. Moreover, if we add any edge to $\mathcal{C}_1$, the resulting graph will not be embeddable onto $\mathcal{C}$ any more since $\mathcal{C}_1$ already uses all available qubits and edges of $\mathcal{C}$. We can thus say that $\mathcal{C}_1$ is one of the densest graphs of size $|V| - 1$ than can be embedded onto $\mathcal{C}$.

We can generalize the aforementioned construction to m random edge contractions; the resulting graph $\mathcal{C}_m$ will have $|V| - m$ vertices, will be one of the densest graphs of size $|V| - m$ that fits onto $\mathcal{C}$, and the chains of such an embedding will be the paths of contracted edges. This family of graphs $\overline{\mathcal{C}}_m$ with $0 < m < 1100$ is therefore a good candidate for the best-case scenario for the MC problem: The $\overline{\mathcal{C}}_m$ family are large graphs whose QUBOs can be embedded onto $\mathcal{C}$ and whose solutions of MC are not trivial.

3.3.2 Experiments

We solve the MC problem on the $\overline{\mathcal{C}}_m$ family of graphs using DW's Sapi, *PPHa* and the SA-Ising software, SA-clique, and *fmc*. Figure 4 shows the result. We observe that for graph sizes up to 400, *PPHa* finds the same result as DW. For these small graphs the problem is likely simple enough to be solved by the post-processing step alone. As expected, the simulated annealing algorithms designed specifically for MC (*fmc, pmc*) are behaving better than the general SA-Ising algorithm. The *fmc* software is run in its heuristic mode. The comparatively lower quality results we obtain with *fmc* could be due to the fact that *fmc* is designed for large sparse graphs but run here on very dense graphs.

For large graphs (≥ 800 vertices), DW gives the best solution. (Note we do not know if that solution is optimal.)

3.3.3 Speedup

Since SA-clique seems to be the best candidate to compete against DW, and moreover since it is considered the classical analogue of quantum annealing, we choose to compute the DW speedup relatively to SA-clique on the $\overline{\mathcal{C}}_m$ graph family.

We employ the following procedure: For each graph size, we run DW with 500 anneals and report the best solution. The DW runtime is the total qpu runtime for 500 anneals (approximately

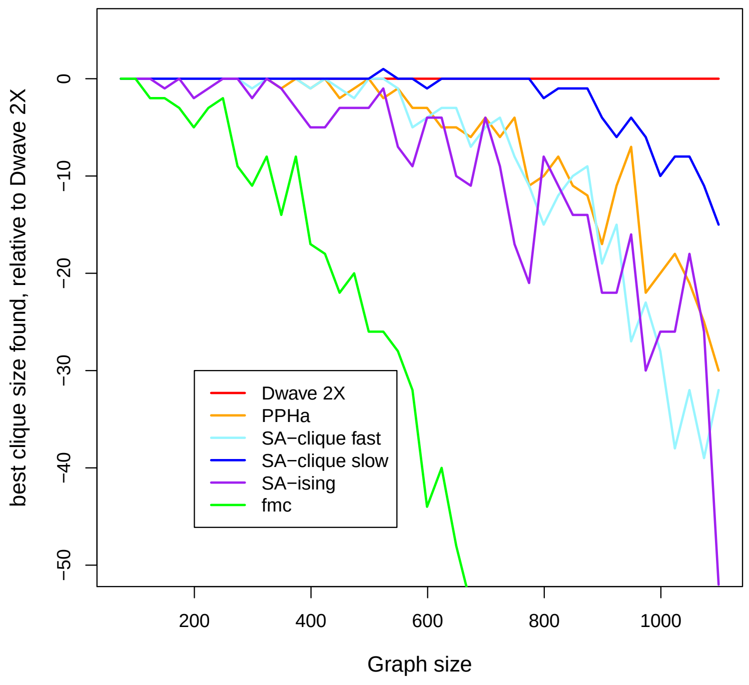


Figure 4: Best clique size found by the different solvers, relatively to the DW result, on the C_m family of graphs.

0.15s). For SA-clique, we start with a low α parameter (i.e., a fast cooling schedule), and gradually increase α until SA-clique finds the same solution as DW. The α for which SA-clique finds the same solution gives us the SA-clique best execution time. The SA-clique algorithm is run on one CPU core of an Intel E8400 @ 3.00GHz. Figure 5 shows the speedup for different graph sizes (of the $\overline{\mathcal{C}}_m$ family). We observe that DW is slower than SA-clique for graphs with less than 200 vertices. For larger graphs, DW gets exponentially faster, reaching a speedup of the order of a million for graphs with 1000 vertices.

Overall, our experiments show that for large graphs whose QUBOs can be embedded onto $\mathcal{C}$, DW is able to find very quickly a solution that is *very* difficult to obtain with classical solvers.

3.3.4 Topology

Combined, the results of Sections 3.2 and 3.3 demonstrate that the closer the topology of a problem is to the native Chimera graph (Fig. 1) of the DW chip, the more pronounced the advantage of DW over classical solvers.

3.4 Using decomposition for large graphs

We investigate some properties of the graph splitting routine of Section 2.3 which enables us to solve MC instances larger than the size that fits onto the DW chip. In this section, we always use our graph splitting routine to divide up the input graphs into subgraphs of 45 vertices, the largest (complete) graphs that can be embedded on the DW chip.

First, we test our graph splitting routine on random graphs with 500 vertices and an edge probability (edge density) ranging from 0.1 to 0.4 in steps of 0.05. Fig. 6 shows the number of generated subgraphs (or equivalently, the number of solver calls) against the edge probability.

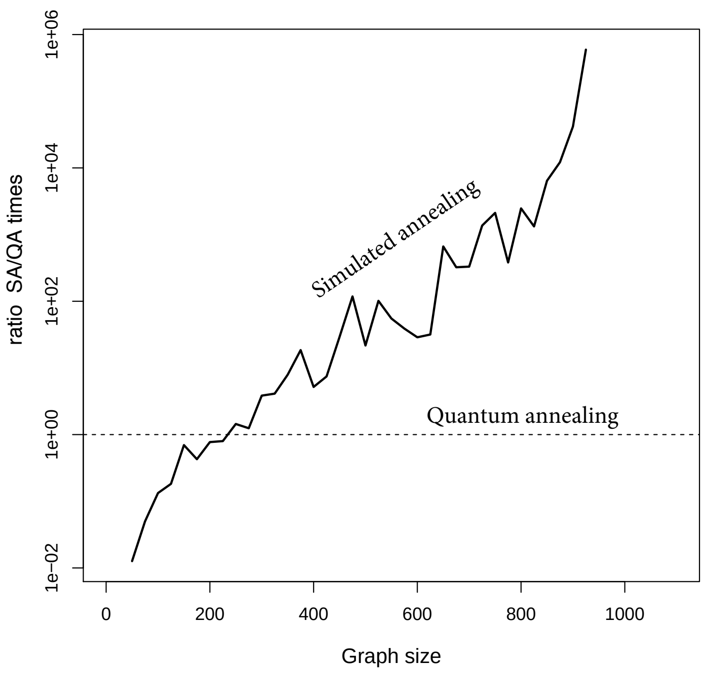


Figure 5: Speedup on artificial graphs designed to fit the Chimera topology.

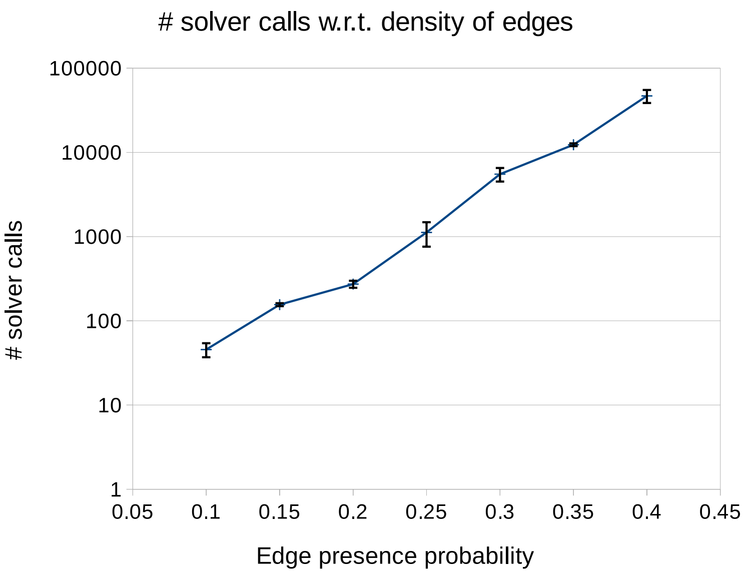


Figure 6: Number of solver calls against edge probability. Log scale on the y-axis.

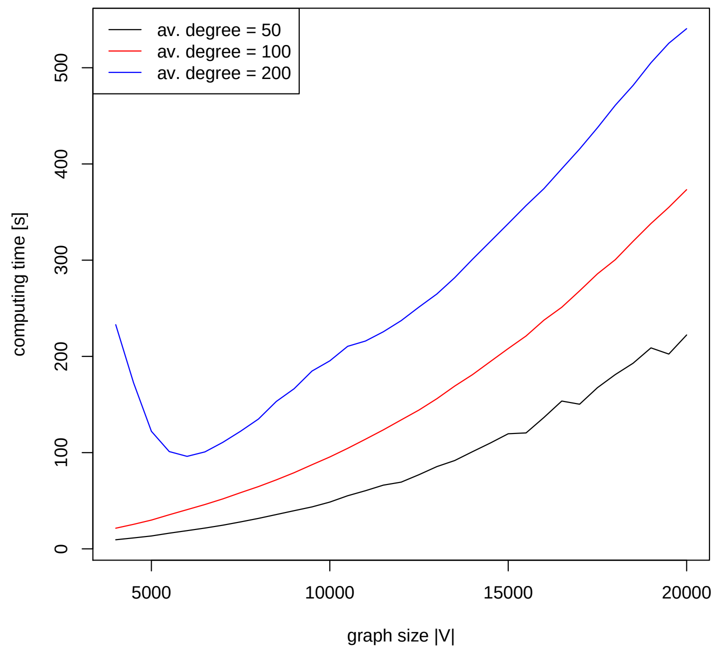


Figure 7: Time of the graph splitting routine as a function of the graph size.

Each data point is the median value of ten runs with the standard deviation given as error bars. The number of solver calls seems to follow an exponential trend with respect to the edge probability.

Second, we investigate the scaling of our graph splitting routine with an increasing graph size $|V|$. Since, with a fixed edge probability, graphs become denser (their vertex degrees increase) as their size goes to infinity, we take an alternative approach and fix the average degree d of each vertex: We then generate graphs of size 3000 to $20,000$ (in steps of 500) using edge probability $p = d/(|V| - 1)$. This ensures that the average vertex degree stays constant as $|V|$ goes to infinity.

We measure both the time t (in seconds) of the graph splitting alone as well as the number n of problems/subgraphs being solved by DW. According to Table 1 (column for DW's interface *Sapi*), the time to solve each subgraph on the DW chip is 0.15 seconds, thus leading to an overall time for computing MC of $t + 0.15 \cdot n$ seconds. Fig. 7 shows average timings from 100 runs for three fixed average degrees $d \in \{50, 100, 200\}$. We observe that if d is relatively large in comparison to $|V|$ (which, in particular, appears to hold for $|V| \approx 5000$ and $d = 200$), the k-core and CH-partitioning algorithms are less effective, while the vertex-splitting routine alone produces too many subgraphs, causing the computing time to get disproportionately high. With increasing the number of vertices, we observe a roughly linear increase of the runtime. As expected, higher average degrees d result in denser graphs and thus higher runtimes.

4 Conclusion

This article evaluates the performance of the DW quantum annealer on maximum clique, an important NP-hard graph problem. We compared DW's solvers to common classical solvers with the aim of determining if current technology already allows to observe a *quantum advantage* for our particular problem. We summarize our findings as follows.

1. The present capacity of around 1000 qubits of the DW chip poses a significant limitation on the MC problem instances of general form that can be solved with DW directly. For random graphs with no special structure and small enough to fit onto DW, the returned solution is of comparable quality to the one obtained by classical methods, even though the highly optimized classical solvers available are usually faster for such small instances.

2. Special instances of large graphs designed to fit DW's Chimera architecture can be solved orders of magnitude faster with DW than with any classical solvers.

3. For MC instances that don't fit DW, the proposed decomposition method offers a way to divide them into pieces that fit DW and to combine all solutions to the created subproblem into an optimal solution of the original problem (assuming DW solves the subproblems optimally, which is usually true, but cannot be guaranteed). Our decomposition methods are highly effective for relatively sparse graphs; however the number of subproblems generated grows exponentially with increasing density. This issue would be alleviated when/if larger D-Wave machines become available.

Overall, we conclude that general problem instances that allow to be mapped onto the DW architecture are typically still too small to show a quantum advantage. But quantum annealing may offer a significant speedup for solving the MC problem, if the problem size is at least several hundred, roughly an order of magnitude larger than what it typically is for general problems that fit D-Wave 2X.

References

Balas, E. and Yu, C. (1986). Finding a maximum clique in an arbitrary graph. *SIAM J Comp*, 15:1054–1068.

Batagelj, V. and Zaversnik, M. (2011). An o(m) algorithm for cores decomposition of networks. *Adv Dat An Class*, 5(2).

Bunyk, P., Hoskinson, E., Johnson, M., Tolkacheva, E., Altomare, F., Berkley, A., Harris, R., Hilton, J., Lanting, T., Przybysz, A., and Whittaker, J. (2014). Architectural considerations in the design of a superconducting quantum annealing processor. *IEEE Trans on Appl Superconductivity*, 24(4):1–10.

D-Wave (2016a). D-Wave post-processing guide.

D-Wave (2016b). Introduction to the D-Wave quantum hardware.

Denchev, V., Boixo, S., Isakov, S., Ding, N., Babbush, R., Smelyanskiy, V., Martinis, J., and Neven, H. (2016). What is the computational value of finite-range tunneling? *Phys Rev X*, 6:031015.

Djidjev, H., Hahn, G., Mniszewski, S., Negre, C., Niklasson, A., and Sardeshmukh, V. (2016). Graph partitioning methods for fast parallel quantum molecular dynamics. *CSC 2016*, 1(1):1–17.

Geng, X., Xu, J., Xiao, J., and Pan, L. (2007). A simple simulated annealing algorithm for the maximum clique problem. *Inf Sciences*, 177(22):5064–5071.

Gurobi Optimization, Inc. (2015). Gurobi optimizer reference manual.

Johnson, M., Amin, M., Gildert, S., Lanting, T. Hamze, F., Dickson, N., Harris, R., Berkley, A., Johansson, J., Bunyk, P., Chapple, E., Enderud, C., Hilton, J., Karimi, K., Ladizinsky, E., Ladizinsky, N., Oh, T., Perminov, I., Rich, C., Thom, M., Tolkacheva, E., Truncik, C., Uchaikin, S., Wang, J., B., W., and Rose, G. (2011). Quantum annealing with manufactured spins. *Nature*, 473:194–198.

King, J., Yarkoni, S., Nevisi, M., Hilton, J., and McGeoch, C. (2015). Benchmarking a quantum annealing processor with the time-to-target metric. *arXiv:1508.05087*, pages 1–29.

Lucas, A. (2014). Ising formulations of many np problems. *Frontiers in Physics*, 2(5):1–27.

Pattabiraman, B., Patwary, M., Gebremedhin, A., Liao, W.-K., and Choudhary, A. (2013). Fast algorithms for the maximum clique problem on massive sparse graphs. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 156–169. Springer.

Rønnow, T. Wang, Z., Job, J., Boixo, S., Isakov, S., Wecker, D., Martinis, J., Lidar, D., and Troyer, M. (2014). Defining and detecting quantum speedup. *Science*, 345:420–424.

Rossi, R., Gleich, D., Gebremedhin, A., and Patwary, M. (2013). A fast parallel maximum clique algorithm for large sparse graphs and temporal strong components. *CoRR, abs/1302.6256.*