Quantum Information Processing (2019) 18:117
https://doi.org/10.1007/s11128-019-2236-3

Efficiently embedding QUBO problems on adiabatic quantum computers

Prasanna Date[1] · **Robert Patton**[2] · **Catherine Schuman**[2] · **Thomas Potok**[2]

Received: 20 September 2018 / Accepted: 28 February 2019

Abstract
Adiabatic quantum computers like the D-Wave 2000Q can approximately solve the QUBO problem, which is an NP-hard problem, and have been shown to outperform classical computers on several instances. Solving the QUBO problem literally means solving virtually any NP-hard problem like the traveling salesman problem, airline scheduling problem, protein folding problem, genotype imputation problem, thereby enabling significant scientific progress, and potentially saving millions/billions of dollars in logistics, airlines, healthcare and many other industries. However, before QUBO problems are solved on quantum computers, they must be embedded (or compiled) onto the hardware of quantum computers, which in itself is a very hard problem. In this work, we propose an efficient embedding algorithm, that lets us embed QUBO problems fast, uses less qubits and gets the objective function value close to the global minimum value. We then compare the performance of our embedding algorithm to that of D-Wave's embedding algorithm, which is the current state of the art, and show that our embedding algorithm convincingly outperforms D-Wave's embedding algorithm. Our embedding approach works with *perfect* Chimera graphs, i.e., Chimera graphs with no missing qubits.

Keywords Adiabatic quantum computing · Embedding · Quadratic unconstrained binary optimization (QUBO)

This research was supported in part by an appointment to the Oak Ridge National Laboratory ASTRO Program, sponsored by the US Department of Energy and administered by the Oak Ridge Institute for Science and Education.

✉ Prasanna Date
datep@rpi.edu

[1] Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180, USA

[2] Computational Data Analytics Group, Oak Ridge National Laboratory, Oak Ridge, TN 37830, USA

Published online: 05 March 2019

1 Introduction

In computer science, NP-hard is the class of problems that cannot be efficiently solved using modern day computers [41]. Some examples of NP-hard problems are traveling salesman problem (TSP) [5], airline scheduling [25], multiprocessor scheduling [7], halting problem [51], Boolean satisfiability [19], protein folding [22], multiple sequence alignment (MSA) [15], genotype imputation [38], etc. A lot of machine learning and deep learning problems are also NP-hard, for example, K-means clustering [21], learning of Bayesian networks [16], some variants of support vector machines (SVM) [20], and optimally training all variants of deep neural networks (DNN) [11], like convolutional neural networks (CNN), recurrent neural networks (RNN), deep belief networks (DBN), autoencoders, generative adversarial networks (GAN).

NP-hard problems can be found in virtually all application areas under the sun, from aviation to logistics, from machine learning to artificial intelligence, from computer science to physical sciences etc. Being able to solve these problems accurately and efficiently would let us save billions of dollars in the aviation and logistics industry, build AI systems like humanoid robots and autonomous vehicles, that are reliable, robust, efficient and easy to use, have a thorough understanding of crucial biochemical processes as well as the human genome, which may even lead to curing diseases like cancer. So, addressing and solving NP-hard problems are extremely crucial for scientific progress.

While optimally solving NP-hard problems is intractable, researchers have successfully used heuristic algorithms to approximately solve them [23,29,47]. An important characteristic of NP-hard problems is that they are reducible to each other, i.e., they can be converted from one to another efficiently [34]. So, a heuristic algorithm that approximately solves TSP can also be used to approximately solve the airline scheduling problem or the protein folding problem or train DNNs and so on, once they have been reduced to TSP. Similarly, if we have an exact algorithm that can optimally solve any one of the NP-hard problems, then in theory, we would have solved all NP-hard problems.

One such NP-hard problem which adiabatic quantum computers are extremely good at (approximately) solving is called the quadratic unconstrained binary optimization (QUBO) problem, where the objective is to minimize a quadratic polynomial over binary variables [37]. We now formally define the QUBO problem.

Let $\mathbb{B} = \{-1, +1\}$ be the binary set. Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{R}$ be the set of real numbers. Given $\mathbb{B}$, $\mathbb{N}$ and $\mathbb{R}$, a general QUBO problem can be defined as follows:

$$\min_{x \in \mathbb{B}^N} F(x) = x^{\mathrm{T}} A x + x^{\mathrm{T}} b + c \tag{1}$$

where $x \in \mathbb{B}^N$ $(N \in \mathbb{N})$ is a vector of binary variables; $A \in \mathbb{R}^{N \times N}$ is a constant weight matrix; $b \in \mathbb{R}^N$ is a constant weight vector; $c \in \mathbb{R}$ is a constant scalar.

A, b and c are called the parameters of the QUBO problem and x is called the problem variable. Given A, b and c, we would like to know the optimal solution (x^*),

which is the configuration of x which would minimize the QUBO function in Eq. 1. We would also like to know the optimal value of the objective function ($F(x^*)$), i.e., the minimum value of the QUBO function evaluated at the optimal solution (x^*).

2 Adiabatic quantum computers

Quantum computers are poised to play a key role in the quest of solving NP-hard problems [40] as they have demonstrated significant speed-ups on certain problems like integer factorization using the Shor's algorithm [48,49] and searching using the Grover's algorithm [27], that were considered hard for classical computers. The type of quantum computers that use the quantum mechanical process of quantum annealing to perform computation are called adiabatic quantum computers, for example D-Wave quantum computers [2,36]. Specifically, the D-Wave machines use quantum annealing to solve QUBO problems (Eq. 1) characterized by A, b and c, and return both optimal solution (x^*) and optimal value ($F(x^*)$) [3,4,8].

Unlike other quantum computing systems where qubits have been realized in terms of polarization of a single photon [42], spin state of a single electron [43], spin state of a nucleus [31], dot spin of a quantum dot [30], etc., the qubits in D-Wave quantum computers are superconducting flux qubits, which are micrometer sized loops of superconductor material interrupted by a number of Josephson junctions [24,53,54]. Clockwise currents in these loops represent the zero state, and counterclockwise currents represent the one state. These qubits are connected according to the Chimera graph shown in Fig. 1—circles represent qubits and curved lines represent inter-qubit connections. Each block of the Chimera graph contains two lines of $L = 4$ qubits. In Fig. 1, we show nine blocks of the Chimera graph arranged in three rows ($M = 3$) and three columns ($N = 3$).

3 Chimera graph restrictions

The structure of the Chimera graph is such that all-to-all connectivity between qubits cannot be achieved, i.e., a given qubit cannot be connected to any other qubit on the graph, it can only be connected to a small number of qubits. This peculiar graph structure, as shown in Fig. 1, imposes certain restrictions on the kind of QUBO problems that can be solved on the D-Wave quantum computers. The D-Wave quantum computers can solve only those QUBO problems which meet the following restrictions:

1. $c = 0$ constant terms are not allowed.
2. $a_{ii} = 0 \, \forall i$ square terms are not allowed in the objective function, meaning the diagonal of the matrix A can only contain zeros.
3. Depending on where a qubit is located on the Chimera graph, it can be connected with at most $L + 1$ or $L + 2$ other qubits. In Fig. 1, $L = 4$, so any qubit can be connected to five or six other qubits based on its location.

The first restriction does not affect the optimal solution of the QUBO problem, because constant terms like c in Eq. 1 do not affect the optimal solution of

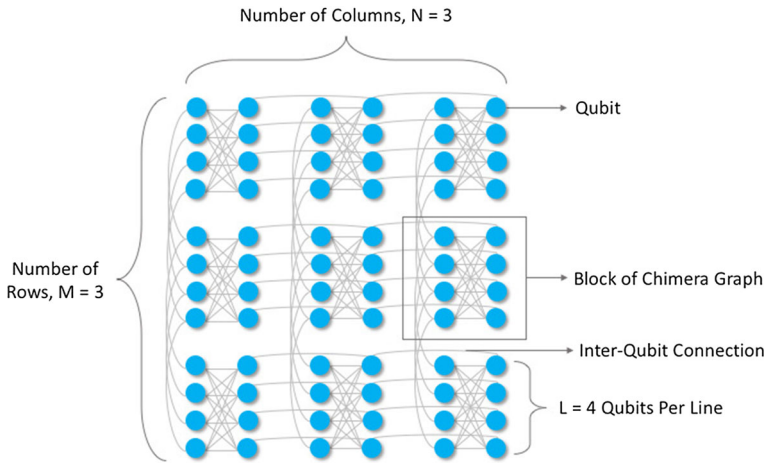


Fig. 1 Chimera graph containing nine Chimera blocks, arranged in three rows ($M = 3$) and three columns ($N = 3$). Circles represent qubits and lines represent inter-qubit connections. Each Chimera block contains two lines of four qubits ($L = 4$). Total number of qubits in this Chimera graph $= 2LMN = 2{\times}4{\times}3{\times}3 = 72$ (Color figure online)

unconstrained optimization problems. So, we can set $c = 0$ without changing the optimal solution, thereby adhering to the first Chimera graph restriction. To elaborate on the second and third restrictions, there do not exist inter-qubit connections from any qubit to every other qubit on the Chimera graph, i.e., all-to-all connectivity does not exist. Also, there is no connection from a qubit to itself, because of which square terms cannot be embedded onto the hardware. In Fig. 1, each qubit in the block $(2, 2)$—i.e., second row and second column—can be connected to at most six other qubits, and the qubits in all other blocks can be connected to at most five other qubits. If a problem requires more than six connections per qubit (which usually is the case in real-world problems), it cannot be embedded on the Chimera graph as it is, and must be modified to make it compatible with the Chimera graph. To do so, a coupling approach is used, which in some way, is similar to quantum entanglement—a quantum mechanical phenomenon that occurs when the quantum states of two or more quantum systems cannot be described independently of each other.

We would like to define two terms here: *Coupling* and *Embedding*. *Coupling* refers to making two or more qubits behave in sync with each other, i.e., when any one of the coupled qubits is in the one state (or the zero state), then all of the other coupled qubits are also in the one state (or the zero state). In Sect. 5, we propose an approach to couple two or more qubits and mathematically prove its correctness. We also present the coupling algorithm (Algorithm 1), which demonstrates that coupling qubits can be done in quadratic time. The second term we want to define is called *Embedding*, which refers to mapping all the variables of a QUBO problem to the physical qubits on adiabatic quantum computers arranged in a Chimera graph. We present our embedding approach in Sect. 6, along with the embedding algorithm (Algorithm 4) and all its subroutines (Algorithms 2, 3). Because of the restrictions imposed by the Chimera

graph, coupling qubits is the only way to embed QUBO problems requiring more than $L+2$ connections for some or all of its variables. It must be noted that our embedding approach works with *perfect* Chimera graphs, i.e., Chimera graphs where all qubits are functional, and no qubit is missing.

4 Related work

Adiabatic quantum computers like the D-Wave quantum computers offer the potential to approximately solve NP-hard problems like the QUBO problem much faster than classical computers. This has sparked considerable interest in the research community, with significant research being undertaken at several academic universities (USC, UNM, etc.) [32,44], government research facilities (ORNL, NASA, etc.) [13,45] and industry research centers (Google, Lockheed Martin, etc.) [1,6]. The main bottleneck preventing widespread usage of adiabatic quantum computers in general, and D-Wave quantum computers in particular, seems to be the embedding problem, i.e., mapping QUBO variables to the qubits on these quantum computers. The embedding problem is a difficult problem in itself. There has been some published work regarding the embedding problem—research papers as well as patents—and we review it here.

Majority of work for the embedding problem comes from D-Wave Systems. Cai et al. [14] used a heuristic algorithm for finding a minor graph embedding of QUBO problems—this was the first embedding algorithm proposed by D-Wave. Boothby et al. [12] built on above work to showcase the fast clique minor generation embedding algorithm—this was the second embedding algorithm proposed by D-Wave. In a patent awarded to D-Wave, Roy proposes a two-stage embedding system—in the first stage, sets of connected subgraphs are generated for each decision variable, and in the second stage, connected subgraphs are refined so that each node of the subgraph represents not more than a single decision variable [46]. Continuing along this line of work, Macready and Roy propose a hardware graph decomposition based method for embedding using a two-stage process [39]. In another D-Wave patent, Thom et al. [50] proposed methods for embedding QUBO problems of arbitrary size or inter-variable connectivity. In yet another D-Wave patent, King et al. [33] used automorphism of the problem graph to determine an embedding and also provide an upper bound on the required chain strength. They also use genetic algorithm to generate hyperparameters for the quantum processor. Bian et al. [9,10] explored locally and globally structured approaches for embedding Boolean constraint satisfaction problems on the Chimera hardware.

Apart from D-Wave, private companies like Lockheed Martin and 1QBit Information Technologies, and Government research institutions like Oak Ridge National Laboratory (ORNL) and National Aeronautics and Space Administration (NASA) have published papers and patents on the embedding problem. Klymko et al. [35] from ORNL used a graph-theoretic approach for minor embedding with hard faults and propose an embedding algorithm that scales linearly in time and quadratically in qubit footprint. Venturelli et al. [52] from NASA and other organizations used a graph minor embedding technique for encoding the Sherrington–Kirkpatrick problem onto the Chimera graph of the D-Wave two quantum computer. Adachi et al. [1] employed a heuristic technique for graph minor embedding to embed QUBO

problems on the hardware—this work has been patented by the Lockheed Martin Corporation. Hamilton and Humble from ORNL introduce the minor set cover (MSC) approach for embedding, which can generate complete bipartite graph embeddings for certain hardware [28]. Zaribafiyan et al. [55] presented an embedding method that finds embedding by decomposing the QUBO problem into a product of graphs representative of the input problem—this work has been patented by 1Q Bit Information Technologies Inc. Goodrich et al. [26] from ORNL introduced a biclique virtual hardware layer as a simplified interface to the physical hardware, and present an odd cycle traversal (OCT)-based embedding algorithm by exploiting the bipartite structure in quantum programs.

Although these approaches graphically show the final embeddings, not all of them give the exact underpinnings that encode these embeddings. Moreover, it is not explicitly clear how their embedding algorithms scale with the problem size. Furthermore, empirical comparison of performance of these embedding algorithms to D-Wave's embedding algorithm, which is considered to be the state of the art, does not seem to be shown in the above body of work. Also, if at all any scaling results are presented, they used quantum computers of considerably small size, e.g., the D-wave two quantum computer, which contains 512 qubits. In this work, we present our embedding algorithm, explicitly state the mathematical formulae that govern the embedding, compare the performance of our algorithm to that of D-Wave's algorithm (state of the art) across three performance metrics, and also show scaling results on the D-Wave 2000Q quantum computer, which contains 2048 qubits.

5 Coupling qubits

Depending on where a qubit is located on the Chimera graph (Fig. 1), it can be connected to $L+1$ or $L+2$ other qubits. In Fig. 1, $L=4$, so each qubit can be connected to either five or six other qubits. If we map one qubit to one QUBO variable, then we would be able to connect the QUBO variable to not more than $L+2$ other variables. In order to enable more than $L+1$ or $L+2$ connections for a QUBO variable, it must be mapped to multiple qubits on the Chimera graph that behave in sync with each other, meaning, when any one of them equals $+1/-1$, all of them would equal $+1/-1$. These synced qubits are said to be *coupled* with each other. Coupling qubits can be done mathematically by modifying a general QUBO problem in a way which not only preserves the optimal solution of the original problem, but also adheres to the restrictions stated in Sect. 3.

In this section, we first propose our approach for coupling two qubits. Next, by following a series of claims, we provide a proof of correctness for our coupling approach. Finally, to show that this coupling approach runs efficiently, we present the Coupling Algorithm that intakes a QUBO problem and returns a new QUBO problem which adheres to the Chimera graph restrictions from Sect. 3. The coupling algorithm, however, does not take into account the structure of the Chimera graph. This is done in Sect. 6, where we present the embedding algorithm, that intakes a QUBO problem and leverages the structure of the Chimera graph to return a new QUBO problem which adheres to the Chimera restrictions. The coupling approach is as follows:

Qubit Coupling Approach: Two qubits x_i and x_j can be coupled with each other by setting the weight a_{ij} to a large negative number in Eq. 1.

In order to prove that this coupling approach works, first of all, let a general QUBO problem $(\boldsymbol{\alpha})$ be defined as follows:

$$\boxed{\min_{x^\alpha \in \mathbb{B}^{N^\alpha}} F^\alpha(x^\alpha) = x^{\alpha T} A^\alpha x^\alpha + x^{\alpha T} b^\alpha + c^\alpha} \tag{2}$$

where $x^\alpha \in \mathbb{B}^{N^\alpha}$; $A^\alpha \in \mathbb{R}^{N^\alpha \times N^\alpha}$; $b^\alpha \in \mathbb{R}^{N^\alpha}$; $c^\alpha \in \mathbb{R}$; $N^\alpha \in \mathbb{N}$.

Let the optimal solution for Problem $(\boldsymbol{\alpha})$ be $x^{\alpha *}$ and the optimal value be $F^\alpha(x^{\alpha *})$.

Now, consider another QUBO problem $(\boldsymbol{\beta})$ as follows:

$$\begin{aligned} &\boxed{\min_{x^\beta \in \mathbb{B}^{N^\beta}} F^\beta(x^\beta) = x^{\beta T} A^\beta x^\beta + x^{\beta T} b^\beta} \\ &\text{such that:} \quad a^\beta_{ij} = a^\alpha_{ij} + a^\alpha_{ji} \quad \forall i < j \\ &\qquad\qquad\quad a^\beta_{ij} = 0 \qquad\quad \forall i \geq j \end{aligned} \tag{3}$$

where $x^\beta \in \mathbb{B}^{N^\beta}$; $A^\beta \in \mathbb{R}^{N^\beta \times N^\beta}$; $b^\beta = b^\alpha$; $N^\beta = N^\alpha$.

So, A^β is an upper triangular matrix with all diagonal entries equal to zero as follows:

$$A^\beta = \begin{bmatrix} 0 & a^\alpha_{1,2} + a^\alpha_{2,1} & \cdots & a^\alpha_{1,N} + a^\alpha_{N,1} \\ 0 & 0 & \cdots & a^\alpha_{2,N} + a^\alpha_{N,2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a^\alpha_{N-1,N} + a^\alpha_{N,N-1} \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

Claim Optimal solution of problem $(\boldsymbol{\alpha})$ is the same as that of problem $(\boldsymbol{\beta})$, i.e., $x^{\alpha *} = x^{\beta *}$, and the optimal values, $F^\alpha(x^{\alpha *})$ and $F^\beta(x^{\beta *})$ are related as follows: $F^\beta(x^{\beta *}) = F^\alpha(x^{\alpha *}) - \sum_{i=1}^{N^\alpha} a^\alpha_{ii} - c^\alpha$.

Proof Since the constant c^α does not affect the optimal solution for any general unconstrained optimization problem, it can be eliminated without changing the optimal solution. Furthermore, because $x^\alpha_i, x^\beta_i \in \mathbb{B}\, \forall i$, we have $(x^\alpha_i)^2 = (x^\beta_i)^2 = 1$, regardless of whether x^α_i or x^β_i equals -1 or $+1$. This implies that $\sum_{i=1}^N a^\alpha_{ii}(x^\alpha_i)^2 = \sum_{i=1}^N a^\alpha_{ii}$ $\forall i$, which essentially means that the sum of all the square terms equals trace of the matrix A^α, and can be treated as a constant and eliminated without changing the optimal solution. Finally, for any non-square quadratic term $x^\alpha_i x^\alpha_j$ in problem $(\boldsymbol{\alpha})$ $(i \neq j)$, its coefficient equals $(a^\alpha_{ij} + a^\alpha_{ji})$. In problem $(\boldsymbol{\beta})$, we set $a^\beta_{ij} = a^\alpha_{ij} + a^\alpha_{ji}$ for all $i < j$ and set all $a^\beta_{ij} = 0$ for all $i \geq j$, thereby making A^β an upper triangular matrix. Note that in doing so, the coefficients of all the non-square quadratic terms (x^β_{ij}) remain unchanged

and thus, the optimal solution remains unaffected after this operation. Now, if we subtract the constant c^{α} and the trace of the matrix A^{α} from optimal value of problem $(\boldsymbol{\alpha})$, we end up with the optimal value for problem $(\boldsymbol{\beta})$, because the optimal solution of problem $(\boldsymbol{\beta})$ is not affected by any of the above operations. □

This takes care of the first two restrictions in Sect. 3 because we have ensured that $c^{\beta} = 0$ and $a_{ii}^{\beta} = 0$ for all i in Problem $(\boldsymbol{\beta})$. To tackle the third restriction, we use our coupling approach in the Coupling Algorithm (Algorithm 1), which takes as input the QUBO matrix A^{β} and returns a new QUBO matrix A^{γ} that adheres to all Chimera graph restrictions in Sect. 3. But first, we describe the rationale behind the Coupling Algorithm.

We know that a qubit in the Chimera graph can be connected to not more than $L+1$ or $L+2$ other qubits. Assuming $L = 4$ as in Fig. 1, a qubit can be connected to five or six other qubits. Since a qubit is mapped to a QUBO variable, this implies that a QUBO variable can be connected to at most six other variables when embedded on the Chimera graph. Suppose we have a row in A^{β} having more than five/six nonzero terms in the upper triangular part of the matrix, corresponding to the qubit x_i having more than five connections. In order to embed this row onto the Chimera graph, we need to couple x_i with a new qubit $x_{i'}$ such that $i' > N^{\beta}$. This can be through the following steps:

1. Create a new row and a new column in A^{γ} corresponding to the new qubit $x_{i'}$.
2. In order to make $x_{i'}$ behave exactly like x_i, set $a_{ii'}^{\gamma} = -\Lambda$, where $\Lambda \gg 0$ is a large positive integer, in accordance with our coupling approach—we prove its correctness through the claim below.
3. Now, if we want to relieve the original qubit x_i off its connection to another qubit x_j, set $a_{ji'}^{\gamma} = a_{ij}^{\gamma}$ and set $a_{ij}^{\gamma} = 0$.
4. If the column of A^{γ} corresponding to the jth qubit (x_j) contains more than five/six entries, we will have to create a new qubit $x_{j'}$ by creating a new row and a new column in A^{γ}, and coupling it with qubit x_j by setting $a_{jj'}^{\gamma} = -\Lambda$. Unlike the previous case where we set $a_{ji'}^{\gamma} = a_{ij}^{\gamma}$ and $a_{ij}^{\gamma} = 0$, in this case, we set $a_{j'i'}^{\gamma} = a_{ij}^{\gamma}$ and $a_{ij}^{\gamma} = 0$.

In doing so, we have reduced the number of nonzero entries in the ith row of A^{γ} by one and added an element to the newly created i'th row. Also, depending on whether the jth column of A^{γ} contains less than five elements or not, we keep the number of nonzero entries in the jth column of A^{γ} the same in the first case, or reduce it by one in the second case where we couple the jth qubit with the j'th qubit. Note that because $i' > N^{\beta} \geq i$, all the above operations happen in the upper triangular part of the matrix. After performing the above steps for all the qubits that have more than five/six nonzero entries in either their rows or columns, we would end up with A^{γ} that has $N^{\gamma} > N^{\beta}$ rows and columns, and each row and each column of A^{γ} contains at most five/six nonzero terms.

Let the number of new qubits introduced as per the above description of the Coupling Algorithm be η ($\eta \in \mathbb{N}$). We would like to get an upper bound on η. Say $N^{\beta} > 5$ and all the elements in the upper triangular part of A^{β}, (excluding the diagonal), are nonzero, which means we have a total of $\frac{1}{2}N^{\beta}(N^{\beta}-1)$ nonzero terms in A^{β}. For the ith qubit,

we have $(N^\beta - 1)$ connections, i.e., the ith row and ith column of A^β contain a total of $(N^\beta - 1)$ nonzero terms. Now, since each qubit can be connected to at most five/six other qubits on the Chimera graph, we can assign connections corresponding to the first four/five nonzero terms of the ith qubit directly and use the fifth/sixth connection to couple the ith qubit with another qubit (say i'). We can assign three/four more connections via the coupled qubit i' before we have to couple the i'th qubit with yet another qubit (say i''). In doing so, out of the five/six connections that are available to i', two are used up for entangling it with i and i'' and the remaining three/four are used to connect the qubits which were supposed to connect to the original ith qubit. Thus, to account for all the non-coupled connections possible, each original qubit can be connected to at most four/five other qubits and each entangled qubit can be connected to three/four other qubits. So, an upper bound on the number of entangled qubits for an original qubit is $\left\lceil \frac{(N^\beta-1)-4}{3} \right\rceil = \left\lceil \frac{N^\beta-5}{3} \right\rceil$.

If we add the original qubit to the above number, we get the total number of qubits required to embed all the $N^\beta - 1$ connections of that particular original qubit: $\left\lceil \frac{N^\beta-5}{3} \right\rceil + 1 = \left\lceil \frac{N^\beta-2}{3} \right\rceil$. Finally, for all the N^β original qubits, we would have $N^\beta \left\lceil \frac{N^\beta-5}{3} \right\rceil$ entangled qubits and $N^\beta \left\lceil \frac{N^\beta-2}{3} \right\rceil$ total qubits. So, we get an upper bound for the number of new entangled qubits that must be introduced η: $\eta \leq N^\beta \left\lceil \frac{N^\beta-5}{3} \right\rceil$, and also an upper bound on N^γ: $N^\gamma \leq N^\beta \left\lceil \frac{N^\beta-2}{3} \right\rceil$. From this analysis, we can infer that coupling qubits can be done using quadratic $\mathcal{O}((N^\beta)^2)$ number of qubits, and that the resulting matrix A^γ would be sparse. Furthermore, because we would end up visiting each row and each column of A^β exactly once, coupling qubits can be completed in quadratic $\mathcal{O}((N^\beta)^2)$ time.

To demonstrate the rationale described above, we present the Coupling Algorithm in Algorithm 1, which takes as input the QUBO matrix A^β, encodes all the coupling weights and returns a QUBO matrix A^γ that adheres to the restrictions of Sect. 3. The Coupling Algorithm, however, does not take into account the structure of Chimera graph, and is provided only for demonstration purposes—as a proof of concept for the coupling approach, and to show that it can be computed in quadratic time and space. Here we assume that a qubit can be connected to at most five other qubits on the Chimera graph. Variants of Algorithm 1 can be conceived that can handle the cases where some qubits have five connections and others have six. A brief description of the variables used in the Coupling Algorithm in the order of occurrence is as follows:

- A^β: the input matrix.
- *count*_A^β: an array or list whose ith element represents number of connections for the ith qubit, i.e., number of nonzero elements in the ith row and ith column of A^β.
- Λ: a large positive integer ($\Lambda \gg 0$).
- N^β: number of rows/columns of A^β.
- A^γ: the coupled matrix to be returned.
- N^γ: number of rows/columns of A^γ.

- *coupling_map*: a hash map whose keys contain all the original qubits and values contain a queue, implemented as an array or a list, of qubits that are coupled to a particular original qubit. The values for a particular key k can be accessed by *coupling_map*$[k]$, which in this case is a queue. The ith element of this queue can be accessed by *coupling_map*$[k][i]$.
- *total_qubits*: the total number of qubits used, i.e., total number of rows/columns of A^{γ}.
- *new_qubits*: number of new qubits required for a particular original qubit.
- *count_*A^{γ}: an array or list whose ith element represents number of connections for the ith qubit, i.e., number of nonzero elements in the ith row and ith column of A^{γ}.
- i' and j': placeholder variables to store index of an element to be set in A^{γ}.

A brief description of the math operators and helper functions used in the Coupling Algorithm in the order of occurrence is as follows:

- $\lceil x \rceil$: denotes the ceiling of x, i.e., the smallest integer greater than or equal to x. For example, $\lceil 5.67 \rceil = 6.0$, $\lceil 5.00 \rceil = 5.00$.
- `enqueue`(Q, e): appends an element e to the queue Q (array or list). If Q does not exist, it creates a new queue with single element e. If e itself is an array or a list, it appends all elements of e to Q in the same order.
- `zeros`($shape$): returns an array of shape $shape$ whose elements are all initialized to zero.
- `length`(obj): returns the length of an object obj.
- `dequeue`(Q): deletes the first element of the queue Q.
- `max`(a, b): returns the maximum of numbers a and b.
- `min`(a, b): returns the minimum of numbers a and b.

We now describe the Coupling Algorithm in three phases:

1. In the first phase, we create a map of all the qubits that will be coupled to each of the original qubits and store them in *coupling_map*, which is a hash map. The keys of *coupling_map* contain indices of original qubits from 1 to N^{β}, and the values contain a list of all qubits that have been coupled to the original qubit (including the original qubit itself). In this phase, we also keep a track of the total number of qubits that we use.
2. In the second phase, we declare the output matrix A^{γ}, which is a square matrix of size *total_qubits*. We also declare *count_*A^{γ}, which is an array to keep a track of number of connections assigned to each qubit, for example, the ith element of *count_*A^{γ} keeps track of connections corresponding to the ith qubit. In this phase, we also set entries corresponding to coupled connections in A^{γ} to $-\Lambda$.
3. In the third phase we set all the non-coupled elements of A^{γ} to their respective values in A^{β}, while ensuring that each qubit has no more than five connections, i.e., total number of nonzero elements in the ith row and ith column combined are no more than five.

Now we provide a formal proof of correctness for the Coupling Algorithm, and why we preserve the optimal solution of the original problems **(**$\boldsymbol{\alpha}$**)** and **(**$\boldsymbol{\beta}$**)** when we couple qubits using the coupling approach. We assume that a qubit can be connected

Algorithm 1: The Coupling Algorithm

Function `couple` (A^β, $count\_A^\beta$, Λ):

Input: $A^\beta \in \mathbb{R}^{\mathbb{N}^\beta \times \mathbb{N}^\beta}$, $count\_A^\beta \in \mathbb{R}^{\mathbb{N}^\beta}$
Output: $A^\gamma \in \mathbb{R}^{\mathbb{N}^\gamma \times \mathbb{N}^\gamma}$
Parameters: $\Lambda \gg 0$

`/* PHASE I: Create the Coupling Map */`
$coupling\_map = \{\}$;
$total\_qubits = N^\beta$;
for $i = 1$ **to** N^β **do**
  if $count\_A^\beta[i] > 5$ **then**
    $new\_qubits = \left\lceil \frac{count\_A^\beta - 4}{3} \right\rceil$;
    `enqueue` ($coupling\_map[i]$, $[i,\ total\_qubits,\ \ldots,\ total\_qubits + new\_qubits]$);
    $total\_qubits = total\_qubits + new\_qubits$;
  end
  else
    `enqueue` ($coupling\_map[i]$, i);
  end
end

`/* PHASE II: Declare output matrix and set coupled elements */`
A^γ = `zeros` ([$total\_qubits$, $total\_qubits$]);
$count\_A^\gamma$ = `zeros` ([$total\_qubits$]);
for $i = 1$ **to** N^β **do**
  for $j = 2$ **to** `length` ($coupling\_map[i]$) **do**
    $A^\gamma[coupling\_map[i][j-1], coupling\_map[i][j]] = -\Lambda$;
    $count\_A^\gamma[coupling\_map[i][j-1]] = count\_A^\gamma[coupling\_map[i][j-1]] + 1$;
    $count\_A^\gamma[coupling\_map[i][j]] = count\_A^\gamma[coupling\_map[i][j]] + 1$;
  end
end

`/* PHASE III: Set all elements of output matrix appropriately */`
for $i = 1$ **to** N^β **do**
  for $j = i + 1$ **to** N^γ **do**
    while $count\_A^\gamma[coupling\_map[i][1]] \geq 5$ **do**
      `dequeue` ($coupling\_map[i]$);
    end
    while $count\_A^\gamma[coupling\_map[j][1]] \geq 5$ **do**
      `dequeue` ($coupling\_map[j]$);
    end
    if $A^\beta[i, j] \neq 0$ **then**
      $i' =$ `min` ($coupling\_map[i][0]$, $coupling\_map[j][0]$);
      $j' =$ `max` ($coupling\_map[i][0]$, $coupling\_map[j][0]$);
      $A^\gamma[i', j'] = A^\beta[i, j]$;
      $count\_A^\gamma[i'] = count\_A^\gamma[i'] + 1$;
      $count\_A^\gamma[j'] = count\_A^\gamma[j'] + 1$;
    end
  end
end
return A^γ;

to at most five other qubits. To address the case where some of the qubits have five connections and the others have six, variants of the Coupling Algorithm can be conceived. Furthermore, we do not leverage the structure of Chimera graph in this section, but do so in Sect. 6. To formally prove that coupling qubits x_i and x_j can be done by setting the weight a_{ij} to a large negative number, consider a third QUBO problem $(\boldsymbol{\gamma})$ as follows:

$$\boxed{\min_{x^\gamma \in \mathbb{B}^{N^\gamma}} F^\gamma(x^\gamma) = x^{\gamma T} A^\gamma x^\gamma + x^{\gamma T} b^\gamma}$$

$$\begin{aligned} \text{such that:} \quad \|[a^\gamma_{i,1}\ a^\gamma_{i,2}\ \ldots\ a^\gamma_{i,N^\gamma}]^{\mathrm{T}}\|_0 &\le 5 \quad \forall i \\ \|[a^\gamma_{1,j}\ a^\gamma_{2,j}\ \ldots\ a^\gamma_{N^\gamma,j}]^{\mathrm{T}}\|_0 &\le 5 \quad \forall j \\ a^\gamma_{ij} = -\Lambda \quad &\forall \text{ coupled qubits } x_i,\ x_j,\ i < j \end{aligned} \tag{4}$$

where $x^\gamma \in \mathbb{B}^{N^\gamma}$; $N^\beta \le N^\gamma \le \left\lceil \frac{N^\beta(N^\beta-2)}{3} \right\rceil$; A^γ is the matrix obtained from the Coupling Algorithm (Algorithm 1); $b^\gamma = [(b^\beta)^{\mathrm{T}}\ \ \mathbf{0}^{\mathrm{T}}_{N^\gamma - N^\beta}]^{\mathrm{T}}$, i.e., b^β vector with a zero vector added in order to correct the dimensions; $\|\boldsymbol{a}\|_0$ represents the zero norm of the vector $\boldsymbol{a}$, i.e., number of nonzero elements in $\boldsymbol{a}$; $\Lambda \gg 0$ is a large positive number.

Claim The first N^β elements of the optimal solution of problem $(\boldsymbol{\gamma})$ are the same as those of problem $(\boldsymbol{\beta})$, i.e., $[x_1^{\gamma *}\ x_2^{\gamma *}\ \ldots\ x_{N^\beta}^{\gamma *}]^{\mathrm{T}} = x^{\beta *}$, and the optimal values are related as follows: $F^\gamma(x^{\gamma *}) + \eta\Lambda = F^\beta(x^{\beta *})$.

Proof

$$\text{Let} \quad \zeta = \sum_{i=1}^{N^\beta} \sum_{j=i+1}^{N^\beta} |a^\beta_{ij}| + \sum_{i=1}^{N^\beta} |b^\beta_i|$$

$$\text{Pick} \quad \Lambda \gg \zeta$$

The optimal value of problem $(\boldsymbol{\gamma})$ $(F^\gamma(x^{\gamma *}))$ is bounded by:

$$F^\gamma(x^{\gamma *}) \le \zeta - \eta\Lambda \tag{5}$$

Because if not, then we can set all entangled qubits $x_{i'}$ equal to their respective original qubits x_i, i.e., $x_{i'} = x_i$, and get an optimal value at least $\zeta - \eta\Lambda$. This is because the $-\Lambda$ terms appear as $-\Lambda x_i x_{i'}$ in the objective function. By setting the entangled qubit $x_{i'}$ equal to its original qubit x_i, we would make the coefficient of the corresponding $-\Lambda$ term equal to $+1$. Setting entangled qubits equal to their original qubits for all $-\Lambda$ terms, we would have η number of $-\Lambda$ terms, contributing $-\eta\Lambda$ to the objective value.

Note that because $\Lambda \gg \zeta$, the only way to realize the upper bound of Eq. 5 is to have all entangled qubits $x_{i'}$ equal to their original qubits x_i. Once we have ensured equality of entangled qubits to their original qubits, then we can simply replace the entangled qubits with the original ones in problem $(\boldsymbol{\gamma})$. In doing so, we have converted problem

$(\boldsymbol{\gamma})$ into problem $(\boldsymbol{\beta})$ with the only addition of the $-\Lambda$ terms. After replacing the entangled qubit $x_{i'}$ with its original qubit x_i, we would have $-\Lambda$ terms like $-\Lambda x_i^2$. From arguments in the previous Claim, we can treat the square terms as constants in problem $(\boldsymbol{\gamma})$ without changing the optimal solution because $x_i \in \{-1, +1\}, \therefore x_i^2 = 1$. Since constants do not change the optimal solution of unconstrained optimization problems, we can simply discard the $-\Lambda$ terms and retain the optimal solution. In doing so, we have converted problem $(\boldsymbol{\gamma})$ into problem $(\boldsymbol{\beta})$ without changing the optimal solution. Thus, optimal solution of problem $(\boldsymbol{\gamma})$ is the same as that of problem $(\boldsymbol{\beta})$.

As far as the optimal value is concerned, note that given the same optimal solution for problem $(\boldsymbol{\beta})$ and problem $(\boldsymbol{\gamma})$, the only terms that we introduced in problem $(\boldsymbol{\gamma})$ are the $-\Lambda$ terms. Since there are a total of η such terms, we can get the optimal value of problem $(\boldsymbol{\beta})$ by simply adding back $\eta\Lambda$ to the optimal value of problem $(\boldsymbol{\gamma})$. Thus, $F^{\beta}(x^{\beta *}) = F^{\gamma}(x^{\gamma *}) + \eta\Lambda$. □

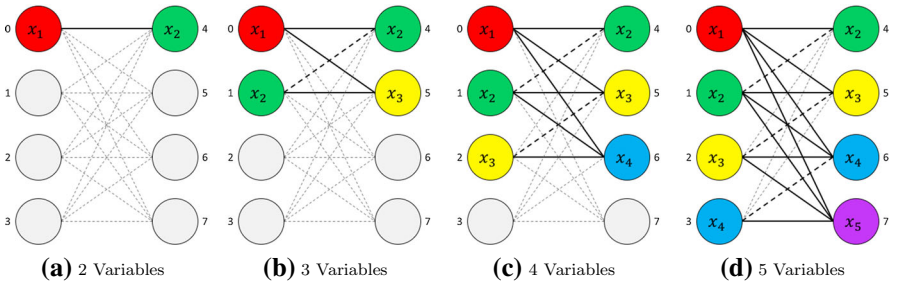
From the first claim, optimal solution of Problem $(\boldsymbol{\alpha})$ equals that of Problem $(\boldsymbol{\beta})$, and from the second claim, optimal solution of Problem $(\boldsymbol{\beta})$ equals first N^{β} elements of optimal solution of Problem $(\boldsymbol{\gamma})$. These claims hence prove that two qubits x_i and x_j can be mathematically coupled with each other by setting the weight a_{ij} to a large negative number in a QUBO problem. This can be done in quadratic time and quadratic space as demonstrated by Algorithm 1. In the next section, we use this result in the Embedding Algorithm, which is the algorithm that embeds a QUBO problem similar to problem $(\boldsymbol{\beta})$ onto the Chimera graph representing qubits of adiabatic quantum computers.

6 The embedding algorithm

We leverage the structure of Chimera graph shown in Fig. 1, and use the coupling approach from Sect. 5—i.e., two qubits x_i and x_j can be coupled with each other by setting the corresponding weight a_{ij} to a large negative number—to come up with an algorithm that can embed a general QUBO problem onto the Chimera graph of adiabatic quantum computers. In this section, a general QUBO problem will refer to a QUBO problem similar to problem $(\boldsymbol{\beta})$ (Eq. 3), where the matrix A^{β} is upper triangular with zeros on the main diagonal, and the constant c^{β} is zero. We begin with some examples that can be embedded on a single Chimera block.

6.1 Single-block embedding

Cases where the QUBO problem has zero or one variable are trivial. In the former case, there is virtually no problem to embed, and in the latter case, we assign -1 to the variable if its coefficient is positive or assign $+1$ if the coefficient is negative, compute the objective function value and return the answer. Slightly more interesting cases that can be accommodated on a single Chimera block are the ones where the QUBO problem has two through $L + 1$ variables. These cases are shown in Fig. 2, where $L = 4$, and up to five QUBO variables can be embedded on a single Chimera block. In Fig. 2, circles represent qubits, solid bold lines represent normal inter-qubit



(a) 2 Variables **(b)** 3 Variables **(c)** 4 Variables **(d)** 5 Variables

Fig. 2 Embedding 2–5 variables on a single chimera block. Solid and dashed bold lines represent normal and coupled inter-qubit connections, respectively. Other dashed lines represent unused inter-qubit connections (Color figure online)

connections, dashed bold lines represent coupled inter-qubit connections, and thin dashed lines represent unused inter-qubit connections. The index of each qubit is indicated by a number on either the left or the right side of the qubit. The variable mapped to a particular qubit is shown inside the circle representing the qubit.

In the two-variable case (Fig. 2a), we map the first variable (x_1) to 0th qubit and second variable (x_2) to 4th qubit. In the three-variable case (Fig. 2b), we map x_1 to qubit 0, x_2 to qubit 4 and x_3 to qubit 5. This enables the connections (x_1, x_2) and (x_1, x_3). However, we also need to accommodate the connection (x_2, x_3), but there is no inter-qubit connection from qubit 4 to qubit 5. So, we map x_2 to qubit 1, couple qubit 1 with qubit 4 and make the connection between qubits 1 and 5, thereby accommodating the connection (x_2, x_3). We follow similar process in the 4-variable case (Fig. 2c), where we map x_1 to qubit 0, x_2 to qubits 1 and 4 (coupled), x_3 to qubits 2 and 5 (coupled), and x_4 to qubit 6. To accommodate the fifth variable (Fig. 2d), in addition to the connections in the 4-variable case, we map x_4 to qubit 3 and couple it with qubit 6, and map x_5 to qubit 7.

Algorithm 2 shows single-block embedding algorithm. We first describe the variables used in the single-block embedding algorithm in the order of their appearance.

- A: input matrix of QUBO problem (A^β in Eq. 3).
- b: input vector of QUBO problem (b^β in Eq. 3).
- M: number of rows of Chimera blocks in the Chimera graph.
- N: number of columns of Chimera blocks in the Chimera graph.
- L: number of qubits per line in a Chimera block.
- V: number of variables in the QUBO problem.
- Λ: a large positive number ($\Lambda \gg 0$). This is the value of the weights that would be assigned to coupled inter-qubit connections, i.e., qubits that have been mapped to the same QUBO variable.
- J: output matrix of the embedding algorithm, implemented as a hash map. The keys of the hash map contain a tuple containing row and column indices of nonzero elements in the embedded matrix, and the values of the hash map contain the values stored at the particular row and column index denoted by key of the hash map.
- h: output vector of the embedding algorithm, implemented as an array or a list. Usually, h is larger than b in size, but has the same number of nonzero elements as b.

– *embeddings*: a list containing qubits mapped to QUBO variables. For example, the ith element of *embeddings* contains a list of all qubits that are mapped to the ith variable of the QUBO problem.

Algorithm 2: Single Block Embedding Algorithm: Subroutine of the Embedding Algorithm

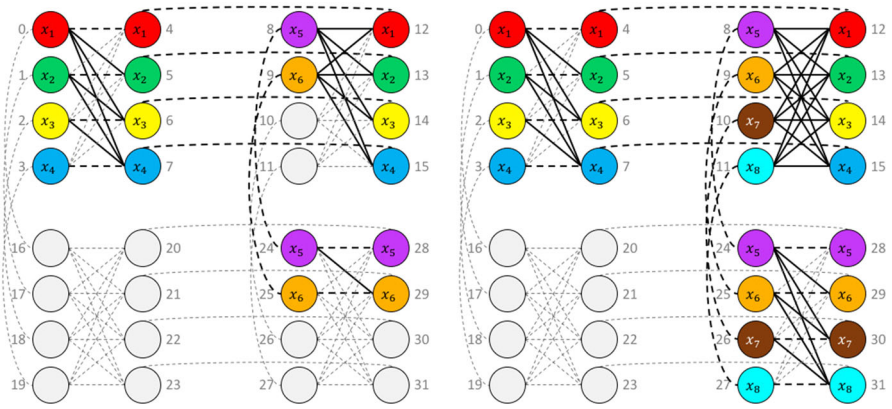
```
1  Function single_block(A, b, M, N, L, V, Λ):
       Input: A ∈ ℝ^{𝕍×𝕍}, b ∈ ℝ^𝕍, M ∈ ℕ, N ∈ ℕ, L ∈ ℕ, V ∈ ℕ, Λ ≫ 0
       Output: J, h, embeddings
2      h = zeros(V + L − 1);
3      J = {};
4      embeddings = [];
5      if V == 0 then
6          return {}, [], [];
7      end
8      else if V == 1 then
9          return {}, b, [[0]];
10     end
11     else if V ≤ L + 1 then
12         for i = 1 to V do
13             embeddings.enqueue([i]);
14             if i == V − 1 then
15                 h[i + L − 1] = b[i];
16             end
17             else
18                 h[i] = b[i];
19                 if i > 0 then
20                     J[i, i + L − 1] = −Λ;
21                     embeddings[i].enqueue(i + L − 1);
22                 end
23             end
24             for j = i + L to L + V − 1 do
25                 J[(i, j)] = A[i, j − L + 1];
26             end
27         end
28         return J, h, embeddings;
29     end
```

In the single-block algorithm, we start off by initializing h, J and *embeddings* on lines 2 through 4 in Algorithm 2. On lines 5 through 10, we check and deal with two base cases, where number of variables V equals 0 or 1. For $V = 2$ through $L + 1$, we compute the J, h and *embeddings* on lines 11 through 27, and return them on line 28. In this way, all-to-all connectivity for $L + 1$ or less variables in a QUBO problem can be achieved using a single Chimera block. Note that the single-block embedding algorithm runs in quadratic time ($\mathcal{O}(V^2)$) and has a linear qubit footprint ($\mathcal{O}(V)$).

6.2 Multi-block embedding

If a QUBO problem has more than $L + 1$ variables (in all our examples, $L = 4$), it cannot be embedded on a single Chimera block, and multiple Chimera blocks must be used to achieve all-to-all connectivity. In Fig. 3, we show the embeddings for 6- and 8-variable QUBO problems. These are slightly different from the previous cases which used a single Chimera block. In the six variable case (Fig. 3a), we start off by mapping variable x_1 to qubits 0 and 4, x_2 to qubits 1 and 5, x_3 to qubits 2 and 6 and x_4 to qubits 3 and 7. This enables all-to-all connectivity within variables x_1, x_2, x_3 and x_4 in the Chimera block located in the first row and first column of the Chimera graph. To make connections between these variables and the remaining two variables (x_5 and x_6), we extend x_1, x_2, x_3 and x_4 by mapping them to the qubits 12, 13, 14 and 15, respectively, and coupling the qubits (4, 12), (5, 13), (6, 14) and (7, 15). Furthermore, x_5 and x_6 are mapped to the qubits 8 and 9, respectively. In doing so, we have established connection between the first four variables (i.e., x_1, x_2, x_3, x_4) and the variables x_5 and x_6 in the Chimera block located in the first row and second column. To make the connection within the variables x_5 and x_6, we first map x_5 to qubits 24 and 28, and couple the qubits (8, 24) and (24, 28), thereby extending the scope of x_5. We next map x_6 to qubits 25 and 29, and couple the qubits (9, 25) and (25, 29), thereby extending the scope of x_6. Finally, we make the connection between x_5 and x_6 using the qubits 24 and 29.

We follow a similar procedure for the 8-variable case as shown in Fig. 3b. In addition to all the connections in the 6-variable case, we map the variable x_7 to qubits 10, 26, 30 and the variable x_8 to qubits 11, 27, 31. In doing so, the connections within variables x_1, x_2, x_3, x_4 are made in the Chimera block in the first row and first column (or top left Chimera block) of the Chimera graph. The connections between x_1, x_2, x_3, x_4 and



(a) 6 Variables **(b)** 8 Variables

Fig. 3 Embedding 6- and 8-variable QUBO problems. Solid and dashed bold lines represent normal and coupled inter-qubit connections, respectively. Other dashed lines represent unused inter-qubit connections (Color figure online)

x_5, x_6, x_7, x_8 are made in the Chimera block in the first row and second column of the Chimera graph. Lastly, the connections within variables x_5, x_6, x_7, x_8 are made in the Chimera block in the second row and second column of the Chimera graph.

The examples shown in Figs. 2 and 3 were meant to provide an intuition to the reader on how the Embedding Algorithm works. The main intuition is as follows. The first row of the Chimera graph is used to connect first L variables (in this case, $L = 4$, so x_1, x_2, x_3, and x_4) to all other variables. As a reminder, L is the number of qubits in one line of a Chimera block, which is comprised of two such lines. Second row of the Chimera graph is used to make connections from second set of L variables (i.e., x_5, x_6, x_7, and x_8) to all variables having index $2L + 1 = 9$ and above. The third row will be used to connect the third set of L variables (i.e., x_9, x_{10}, x_{11} and x_{12}) to all variables having index $3L + 1 = 13$ and above, and so on. In general, the ith row of Chimera graph is used to connect the variables having index from $L(i-1)+1$ through Li to variables having index $Li + 1$ and above. Just as the rows of Chimera graph are 'assigned' to a particular set of L variables, so are the columns. The first column corresponds to connections for the first set of L variables, the second column for the second set of L variables and so on. So, connections from first set of L variables (i.e., x_1, x_2, x_3, x_4) to themselves are made in the first row and first column of the Chimera graph. Connections from first set of L variables to the second set of L variables (i.e., x_5, x_6, x_7, x_8) are made in the first row and second column of the Chimera graph. Similarly, connections from first set of L variables to third set of L variables (i.e., x_9, x_{10}, x_{11}, x_{12}) will be made in the first row and third column of the Chimera graph and so on.

6.3 Mapping qubits for embedding

We now extend this intuition into a more formal algorithm. First of all, we must define the notion of *Start Qubit*, *Base Qubit* and *End Qubit*. The *Start Qubit* of a particular variable is a qubit in the first row of the Chimera graph that has been mapped to that variable. If there are multiple qubits in the first row mapped to a variable, then start qubit is the smallest indexed qubit. It is defined in Eq. 6. For example, in Fig. 3b, the start qubit for the variable x_1 is 0, for x_2 is 1, and for x_7 is 10. The *Base Qubit* for a particular variable is the smallest indexed qubit on the main diagonal of the Chimera graph that has been mapped to that variable. It is defined in Eq. 7. For example, in Fig. 3b, base qubit for the variable x_1 is 0, for x_2 is 1, and for x_7 is 26. The *End Qubit* for a particular variable is L (in all the examples so far, $L = 4$) added to the largest indexed qubit that has been mapped to that variable. It is defined in Eq. 8. For example, in Fig. 3b, the end qubit for the variable x_1 is 16, for x_2 is 17, and for x_7 is 34. We define End Qubit in this peculiar way because it is used in conjunction with the `range(start, end, step)` function, which returns a list of numbers starting at `start`, ending at `end` (which is not included), and having a step size of `step`. For example, `range(2, 14, 4)` would return `[2,6,10]`. An example of the way in which we use the Start, Base and End qubits is: `range(start_qubit, base_qubit, vertical_step)`, `range(base_qubit + L, end_qubit, horizontal_step)`, etc. Vertical Step and Horizontal Step are defined in Eqs. 9 and 10, respectively.

Table 1 Indices of start qubit, base qubit and end qubit for an 8-variable QUBO problem (Fig. 3b)

Variable	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
Start qubit	0	1	2	3	8	9	10	11
Base qubit	0	1	2	3	24	25	26	27
End qubit	16	17	18	19	32	33	34	35

Vertical Step is the number of qubits that must be skipped when coupling qubits vertically, i.e., coupling two qubits in different Chimera blocks using a single vertical connection that is available on the Chimera graph. It essentially is the number of qubits in a row of Chimera graph. The vertical step in Fig. 3b, for example, is 16. *Horizontal Step* is the number of qubits that must be skipped when coupling qubits horizontally, i.e., coupling two qubits in different Chimera blocks using a single horizontal connection that is available on the Chimera graph. It essentially is the number of qubits in a block of Chimera graph. The horizontal step in Fig. 3b, for example, is 8. Table 1 shows the start qubits, base qubits and end qubits for all variables in the 8-variable QUBO problem shown in Fig. 3b.

We now generalize the above notion and write it formally. Given a Chimera graph with M rows, N columns and L qubits per line of a Chimera block, and a QUBO problem with V variables, the start qubit, base qubit and end qubit for the ith variable, as well as vertical step and horizontal step can be defined as follows:

$$\text{Start qubit} = 2L \left\lfloor \frac{i-1}{L} \right\rfloor + (i-1) \bmod L \tag{6}$$

$$\text{Base qubit} = 2L \left\lfloor \frac{i-1}{L} \right\rfloor [\min(M, N) + 1] + (i-1) \bmod L \tag{7}$$

$$\text{End qubit} = \text{base qubit} + 2L \left(\left\lceil \frac{V}{L} \right\rceil - \left\lfloor \frac{i-1}{L} \right\rfloor \right) \tag{8}$$

$$\text{Vertical step} = 2LN \tag{9}$$

$$\text{Horizontal step} = 2L \tag{10}$$

where $\lfloor z \rfloor$ and $\lceil z \rceil$ denote the floor and ceiling operators, respectively. Floor of a number z (i.e., $\lfloor z \rfloor$), is the greatest integer less than or equal to z. Ceiling of a number z (i.e., $\lceil z \rceil$), is the smallest integer greater than or equal to z. The mod operator refers to the remainder of division, for example, $a \mod b$ refers to the remainder when a is divided by b.

We use the definitions in Eqs. 6–10 to map QUBO variables to their respective qubits in Algorithm 3. This algorithm is used only when multiple Chimera blocks are required for embedding. Given the Chimera graph architecture, defined by number of rows (M), number of columns (N) and number of qubits per line (L), and the number of variables in the QUBO problem (V), Algorithm 3 returns the mapping of QUBO variables to qubits on the Chimera graph as a hash map. Note that this completely deterministic mapping can be completed in linear time ($\mathcal{O}(V)$) and has a quadratic ($\mathcal{O}(V^2)$) qubit footprint. Also, because the mapping depends only on the Chimera

graph architecture, and not on the specifics of the QUBO problem, it can be run in constant time ($\mathcal{O}(1)$) if the variable-qubit mappings are precomputed and stored as a hash map. In this case, as soon as the user specifies size of the QUBO problem, the embeddings could be looked up from the hash map.

Using Algorithm 3, one can embed at most $V \leq L \min(M, N)$ variables. This is because each row (and each column) of the Chimera graph can be used to represent at most L variables as mentioned previously. So, number of variables having all-to-all connectivity that can be embedded on Chimera graph is constrained by number of rows (M) and number of columns (N) of the Chimera graph. This condition is checked in line 3 of Algorithm 3, after which *v_step* and *h_step* are computed. Next, the start, base and end qubits are computed for each variable in lines 7 through 9 and a new entry is added to the `mapping` hash map in line 14. The key of this new entry is the variable index and the value is a list of all qubits that are mapped to that particular variable. A special condition is checked in line 10 of Algorithm 3—while mapping the last variable in the case where remainder of V and L equals 1, we only account for vertical coupling and ignore base qubit and all the horizontal couplings. This is illustrated in Fig. 4, where $V = 9$, $L = 4$, and $V \mod L = 1$. While embedding the 9th variable (i.e., x_9), we only map the qubits 16 and 48 to x_9. If we hadn't checked for the above-mentioned condition, we would have mapped the qubits 80, 84, and in some cases, the qubit 92, to x_9 as well. In doing so, we would have wasted three qubits and two additional Chimera blocks, which could have been used elsewhere.

We now briefly describe the new variables used in Algorithm 3 here:

- *mapping*: hash map containing the mapping of QUBO problem variables to qubits on the Chimera graph
- *v_step*: vertical step size for a given Chimera graph
- *h_step*: horizontal step size for a given Chimera graph
- *start_qubit*: start qubit of a particular QUBO variable as defined in Eq. 6
- *base_qubit*: base qubit of a particular QUBO variable as defined in Eq. 7
- *end_qubit*: end qubit of a particular QUBO variable as defined in Eq. 8

A brief description of auxiliary functions and mathematical operators used in Algorithm 3 is as follows:

- `range(start, end, step)`: returns a sequence of numbers as a list or array starting at `start`, ending at `end` (not inclusive), and separated by `step`. For example, `range(2,14,4)` would return `[2,6,10]`.
- `concatenate(a, b, c)`: concatenates the arrays/lists a, b and c one after the other.
- `print(string)`: prints the string `string` onto the output console.
- $a \mod b$: denotes the remainder obtained when a is divided by b, for example $5 \mod 2 = 1$.
- $\lfloor x \rfloor$: denotes the floor function, that returns the largest integer smaller than the number x. For example, $\lfloor 4.69 \rfloor = 4.0$, and $\lfloor 4.0 \rfloor = 4.0$.
- $\lceil x \rceil$: denotes the ceiling function, that returns the smallest integer larger than the number x. For example, $\lceil 4.69 \rceil = 5.0$, and $\lceil 4.0 \rceil = 4.0$.

Algorithm 3: The Mapping Algorithm: Subroutine of the Embedding Algorithm

```
1  Function embedding_helper(M, N, L, V):
       Input: M ∈ ℕ, N ∈ ℕ, L ∈ ℕ, V ∈ ℕ
       Output: mapping
2      mapping = {};
3      if V ≤ L min(M, N) then
4          v_step = 2LN;
5          h_step = 2L;
6          for i = 1 to V do
7              start_qubit = 2L ⌊(i−1)/L⌋ + (i − 1) mod L;
8              base_qubit = 2L ⌊(i−1)/L⌋ (N + 1) + (i − 1) mod L;
9              end_qubit = base_qubit + 2L (⌈V/L⌉ − ⌈(i−1)/L⌉);
10             if (V mod L == 1) and (i == V) then
11                 mapping[i] = range(start_qubit, base_qubit, v_step);
12             end
13             else
14                 mapping[i] = concatenate(range(start_qubit, base_qubit, v_step),
                   [base_qubit], range(base_qubit + L, end_qubit, h_step));
15             end
16         end
17     end
18     else
19         print("Error, too many variables")
20     end
21     return mapping;
```

6.4 Complete embedding algorithm

We now describe the complete Embedding Algorithm in Algorithm 4, which uses the single-block embedding algorithm (Algorithm 2) and the mapping algorithm (Algorithm 3) as subroutines. The Embedding Algorithm takes as input the QUBO matrix A, QUBO vector b, number of rows in Chimera graph M, number of columns in Chimera graph N, number of qubits per line in a Chimera block L, number of variables in the QUBO problem V and a large positive number Λ. It embeds the QUBO problem onto the Chimera graph, and returns a hash map J containing indices of nonnegative elements of the embedded matrix as keys and the numeric values at those indices as the values, an array or a list h corresponding to the vector b in the embedded problem, and the embeddings *embeddings*, as a list, where the ith element of the list contains a list of qubits that are mapped to the ith variable.

First of all, we check if the given QUBO problem can be embedded using a single Chimera graph on line 2. This is possible only if the number of variables in the QUBO problem (V) is less than or equal to one more than the number of qubits per line of a Chimera block (L). If this condition is satisfied, we use the `single_block()` function, described in Algorithm 2 to determine the embedding.

If multiple Chimera blocks are required to embed the given QUBO problem, then we first initialize h, J and *embeddings* on lines 5 through 7 of Algorithm 4. Next, on line 8, we compute the mappings of all QUBO variables to qubits on the Chimera

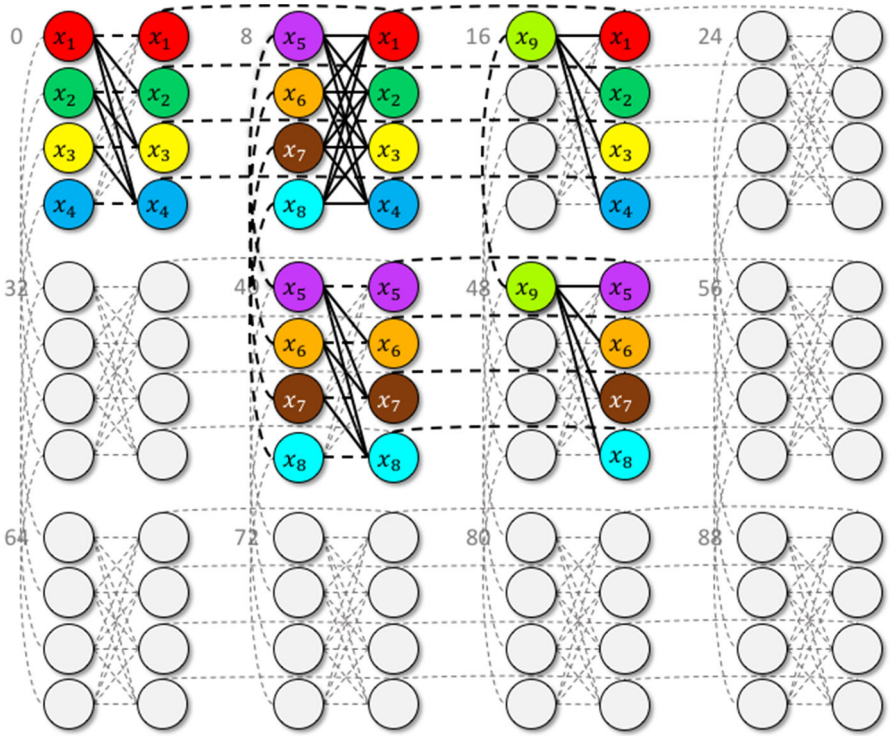


Fig. 4 Embedding a 9-variable QUBO problem on the Chimera graph (Color figure online)

graph using the mapping algorithm (Algorithm 3). *mappings* is a hash map containing QUBO variables as keys and a list of qubits on Chimera graph as values. For each QUBO variable (key), its list of qubits (value) denotes the qubits that have been mapped to that particular variable. We use these mappings to define h on line 10 and also the inter-qubit weights of all the coupled qubits, i.e., all the qubits that are mapped to a particular QUBO variable. Next, on lines 16 through 43, we assign the inter-qubit weights of all other qubits. Specifically, lines 18 through 24 assign weights on the diagonal blocks of the Chimera graph, and lines 25 through 31 assign inter-qubit weights on the off-diagonal upper triangular blocks of the Chimera graph. On lines 32 through 42, we dequeue the appropriate elements of *mapping*.

Note that although all the steps on lines 16 through 43 use four for loops, they run in quadratic time. The for loops on lines 16 and 17 run in increments of L, while the for loops on lines 19, 20, 26 and 27 run in increments of 1. In doing so, each of the V variables of the QUBO problem is visited exactly V times, thus, the algorithm runs in $\mathcal{O}(V^2)$ time and has a quadratic ($\mathcal{O}(V^2)$) qubit footprint. The mapping algorithm (Algorithm 3) however, which is the heart of the embedding algorithm runs in linear time ($\mathcal{O}(V)$).

We now describe the new auxiliary functions used in the Embedding Algorithm (Algorithm 4):

- `single_block(...)`: helper function to embed problems that can be accommodated on a single Chimera block (Algorithm 2).
- `zeros(`*shape*`)`: returns an array of shape *shape* whose elements are all initialized to zero.
- `mapping_algorithm(...)`: helper function that computes the mapping of all QUBO variables to their respective qubits on the Chimera graph.
- `start_qubit(...)`: computes the start qubit as per Eq. 6.
- `exists`: checks the existence of an object. If the object exists, returns *True*, else *False*.

In this way, we have a concrete, deterministic and efficient algorithm for embedding a generic QUBO problem on the Chimera graph that enables all-to-all connectivity for bounded problem sizes.

7 Analyzing performance

Embedding refers to the process of mapping variables of a QUBO problem onto the Chimera graph that represents the qubits of adiabatic quantum computers. An efficient embedding algorithm lets us embed a QUBO problem quickly, using as few qubits as possible and enables us to get a more accurate solution. It also indirectly enables all-to-all connectivity between variables of a QUBO problem that are embedded onto the Chimera graph. Quality of embedding algorithm is crucial for determining the quality of solution that we get from the quantum computer. For example, it is a well-known limitation of current adiabatic quantum computers that maintaining long chains of coupled qubits is extremely difficult. This directly affects the optimal solution that is obtained from the quantum computer. In such a setting, we would prefer an embedding algorithm that uses as few qubits as possible. This avoids long chains of coupled qubits, thereby minimizing the chances of obtaining a poor solution. An inefficient embedding algorithm, on the other hand, would end up using more qubits than necessary, increasing the chances of getting a poor solution. We have described our embedding algorithm in Sect. 6, which is comprised of Algorithms 2, 3 and 4. Now, D-Wave also has their own embedding algorithm, which currently is considered the state of the art. In this section, we compare our embedding algorithm to the D-Wave's embedding algorithm.

Before we compare the two embedding algorithms, we would like to establish what constitutes a good embedding algorithm. In this regard, we look at three characteristics of a good embedding algorithm. First of all, a good embedding algorithm must run as fast as possible, i.e., the *Embedding Time* must be as small as possible. Secondly, it must use as few qubits as possible. A small *Qubit Footprint* decreases the chances of obtaining a poor solution from the D-Wave quantum computer by preventing long chains of qubits. Thirdly, a good embedding algorithm must get us as close to the global minimum value as possible, i.e., the *Accuracy* of the final solution obtained must be high. All these characteristics constitute a good or *efficient* embedding algorithm. We use these characteristics as our performance metrics to compare the two embedding algorithms below.

Algorithm 4: The Embedding Algorithm

Function `embedding(`$A, b, M, N, L, V, \Lambda$`)`**:**
  Input: $A \in \mathbb{R}^{\mathbb{V}\times\mathbb{V}}$, $b \in \mathbb{R}^{\mathbb{V}}$, $M \in \mathbb{N}$, $N \in \mathbb{N}$, $L \in \mathbb{N}$, $V \in \mathbb{N}$, $\Lambda \gg 0$
  Output: J, h, *embeddings*

  `/* Single Block of Chimera Graph */`
  if $V \leq L + 1$ **then**
    return `single_block(`$A, b, M, N, L, V, \Lambda$`)`
  end

  `/* Initialization */`
  h = `zeros(`$2L \lfloor \frac{V}{L} \rfloor + V \bmod L$`)`;
  J = {};
  $embeddings = []$;

  `/* Setting the output vector */`
  $mapping$ = `mapping_algorithm(`M, N, L, V`)`;
  for $i = 1$ **to** V **do**
    h[`start_qubit(`i, L`)`] $= b[i]$;
    $embeddings.enqueue(mapping[i])$;
    for $j = 2$ **to** `length(`$mapping[i]$`)` **do**
      $J[(mapping[i][j-1], mapping[i][j])] = -\Lambda$;
    end
  end

  `/* Setting the non-coupling strengths */`
  for $i = 1, 1 + L, 1 + 2L, \ldots, V$ **do**
    for $j = i, i + L, i + 2L, \ldots, V$ **do**
      if $i == j$ **then**
        for $s = i$ **to** $\min(i + L, V)$ **do**
          for $t = s + 1$ **to** $\min(i + L, V)$ **do**
            $J[(mapping[s][0], mapping[t][1])] = A[s, t]$;
          end
        end
      end
      else
        for $s = i$ **to** $\min(i + L, V)$ **do**
          for $t = j$ **to** $\min(j + L, V)$ **do**
            $J[(mapping[t][0], mapping[s][0])] = A[s, t]$;
          end
        end
      end
      `/* Dequeue assigned qubits from mapping */`
      for $s = i$ **to** $\min(i + L, V)$ **do**
        if $mapping[s]$ *exists* **then**
          $mapping[s].dequeue()$
        end
      end
      for $t = j$ **to** $\min(j + L, V)$ **do**
        if $mapping[t]$ *exists* **then**
          $mapping[t].dequeue()$
        end
      end
    end
  end
  return J, h, *embeddings*;

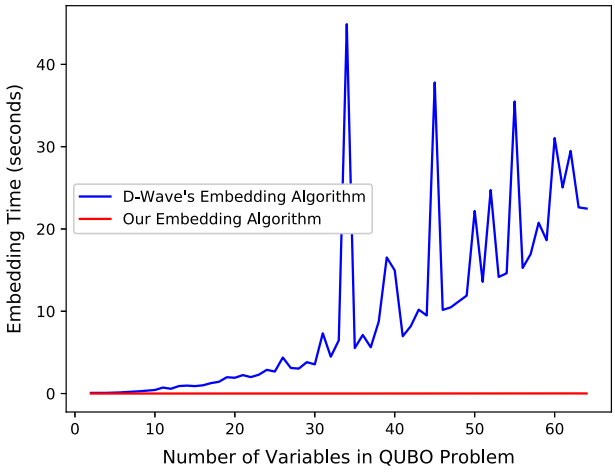


Fig. 5 Embedding time (Color figure online)

In the comparison below, we plot the variation of all three performance metrics with the size of the QUBO problem. The size of a QUBO problem is determined by the size of the vector x in Eq. 1. This is nothing but the number of variables in the QUBO problem, which is shown on the X axis in all the figures described below. We generated all these problems synthetically and randomly, ensuring that the matrix A in Eq. 1 is positive definite. This is required to make sure that the QUBO problem has at least one global minimum.

All embedding runs presented in this section were performed on a machine that had one Intel Core i5-4308U Processor CPU, which contained two cores, running up to four threads, operating at 2.8GHz, and having 3 MB cache. The machine also had 16 GB of 1600 MHz DDR3 RAM. We also used the D-Wave 2000Q quantum computer, which at the time, was the largest quantum computer available. It comprised of 2048 qubits and contained 5600 inter-qubit connections.

Figure 5 shows the variation of Embedding Time shown on the Y axis with number of variables in the QUBO problem shown on the X axis. Plot for D-Wave's embedding algorithm is shown in blue, whereas our embedding algorithm is shown in red. While time taken by D-Wave's algorithm is seen to vary abruptly, our algorithm seems to take almost constant time. In reality, our algorithm is seen to vary linearly with number of variables in the QUBO problem, but the slope of the curve is so small that it appears to be constant. This is in contrast to the analysis in Sect. 6, where we determined that our embedding algorithm ran in quadratic time. We attribute this discrepancy to using the Numpy library in Python as part of our code, which uses highly optimized parallelized implementations of basic programming constructs like for loops. If the loop iterations are independent of each other, a number of iterations are small enough, and size of the data on which each iteration operates is small enough, Python can run all the loop iterations in parallel. This makes a linear algorithm, that uses a single for loop, *appear* to run faster than a linear algorithm implemented in series, and similarly, makes a quadratic algorithm, that uses two nested for loops, *appear* to run faster than a

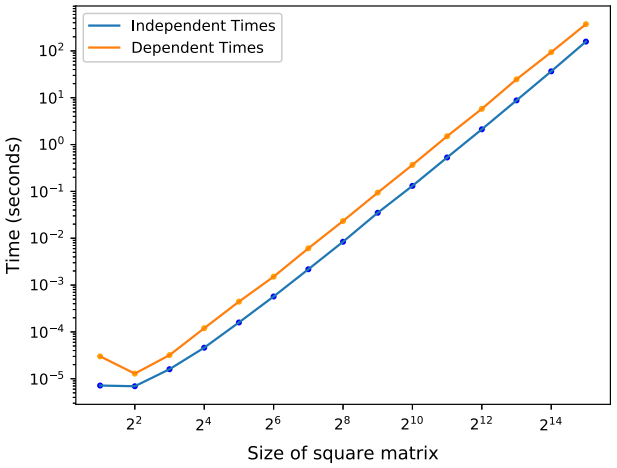


Fig. 6 Scaling study for time taken to set values in the upper triangular part of square matrices. Size of matrices is shown on X axis, and Y axis shows the time taken to complete the operation in Python 2.7. The blue line denotes the case where values being set are independent of each other, while the orange line denotes the case where values being set are dependent on a previously set value in the matrix (Color figure online)

quadratic algorithm implemented in series. We think this is the reason for our quadratic time embedding algorithm seemingly running in linear time in Fig. 5.

To validate our reasoning above, we conducted an experiment in Python 2.7, where we noted the time taken to set elements in the upper triangular part of a matrix A to some value in two cases: (i) In the first case, all elements were set to unity ($A[i, j] = 1 \, \forall i \leq j$), so that all the operations were independent of each other; (ii) In the second case, we made the operations dependent by setting the matrix elements as follows: $A[i, j] = A[i-1, j-1] + 1 \, \forall i \leq j$, so that the value at position (i, j) is dependent on the value at position $(i-1, j-1)$. Before the above operations were performed, the matrix A was initialized to a zero matrix. The above operations were performed using two for loops in Python 2.7. To understand how the performance changes as the size of matrix A increases, we varied the size of A from a 2×2 matrix to a $2^{15} \times 2^{15}$ matrix. The results obtained are shown in Fig. 6. It is evident that when values to be set are independent of each other, the time taken is less as compared to the case where values to be set are dependent on a previously set value. So, when the for loop iterations are independent of each other, Python 2.7 seems to be optimized to run faster. Also, this phenomenon is observed for all sizes of the matrix A. This confirms our reasoning above that it is because of optimized implementation of for loops in Python and Numpy that our quadratic algorithm is seen to run in linear time. As a side note, it must be reiterated that the algorithm that is used to compute the embeddings (Algorithm 3) actually is a linear time algorithm. However, because we end up copying all values of the matrix A^{β} in Eq. 3 to the matrix A^{γ} in Eq. 4, the embedding algorithm (Algorithm 4) becomes a quadratic time algorithm.

On the other hand, D-Wave's algorithm in Fig. 5 takes increasingly more time as the size of the QUBO problem increases with abrupt variations as implied by irregular spikes all over the graph. The reason for this abrupt behavior is that D-Wave uses

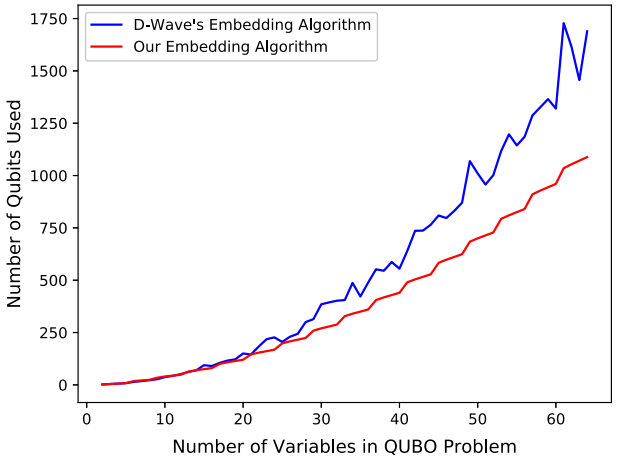


Fig. 7 Comparing embedding algorithms: plot of number of qubits used versus problem size (Color figure online)

a heuristic algorithm to generate the embeddings. The heuristic algorithm does not guarantee that an embedding will be found for sure—sometimes it may not find an embedding, other times it may find an embedding but take a long time, and yet other times it may find an embedding fairly quickly. This behavior of the heuristic algorithm is clearly reflected in Fig. 5. On the other hand, our algorithm is completely deterministic. If the QUBO problem can be accommodated on the Chimera graph of adiabatic quantum computers, then our algorithm will definitely find an embedding. In fact, the very first thing that we check in line 2 of Algorithm 2 is whether the QUBO problem can be accommodated on the Chimera graph or not. Furthermore, we have explicitly determined in Sect. 6.4, the size of the QUBO problems that can be accommodated on a given Chimera graph. To sum up the results for the first performance metric (i.e., Embedding Time), our embedding algorithm completely outperforms D-Wave's embedding algorithm, and this is more apparent for larger sized problems.

Figure 7 shows the variation of number of qubits (or qubit footprint) on the Y axis as number of variables in the QUBO problem shown on the X axis increases. For problems having sixteen or fewer variables, D-Wave's algorithm is seen to use slightly less qubits than our algorithm in some instances. However, as the problem size increases from 16 all the way up to 64, our algorithm convincingly outperforms D-Wave's algorithm on this performance metric as well. Moreover, the gap keeps on increasing and in some instances, our algorithm is seen to use upto 30% less qubits than the D-Wave's algorithm. Even in this case, we see some abrupt variations in the D-Wave's blue curve in Fig. 7. On the other hand, our algorithm seems to vary more regularly with increasing problem size. Abrupt variations in D-Wave's case are once again attributed to their heuristic algorithm, which does not guarantee that an embedding would be found for a given QUBO problem. Moreover, the number of qubits used was found to vary for the same instance of QUBO problem when D-Wave's embedding algorithm was run at different times. This once again highlights the indeterminism of the heuristic algorithm used by D-Wave. In contrast to this, our

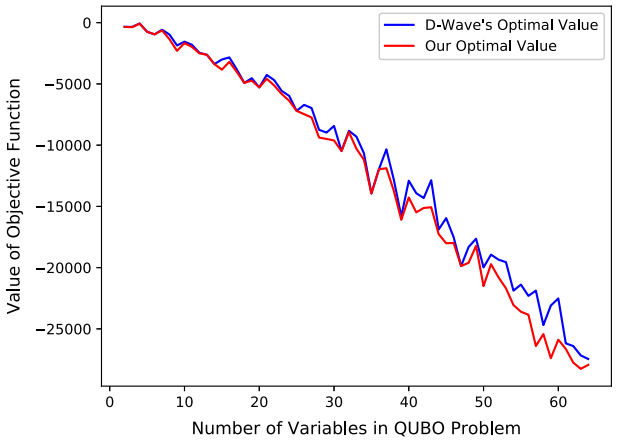


Fig. 8 Comparing embedding algorithms: plot of closeness of obtained solution to the global minima versus problem size: lower objective function values are preferred because the QUBO problem is a minimization problem (Color figure online)

deterministic algorithm used the same number of qubits every single time, produced regular results and guarantees an embedding as long as the QUBO problem can be accommodated on the Chimera graph of the D-Wave hardware. Furthermore, it uses $\mathcal{O}(V^2)$ qubits.

Figure 8 shows the variation of final objective value of the QUBO problem shown on the Y axis with size of the QUBO problem shown on the X axis. Accuracy here refers to how close the final objective function value is to the global minimum. Because we are trying to minimize the objective function, lower objective values are preferred. Because the QUBO problems were generated randomly and synthetically and were not specially fabricated, finding the global minimum of these problems was intractable as the search space became exponentially large. To put in perspective, to find the global minimum value for a 64 variable problem, we would have to search 2^{64} configurations of the variable x. Assuming evaluating one configuration takes one billionth of a second (i.e., 10^{-9} seconds), and operating on a 1 Gigahertz processor which can do a billion evaluations per seconds (i.e., 10^9 evaluations per second), time taken to evaluate 2^{64} configurations would equal $\frac{2^{64}}{10^9} \approx 1.8 \times 10^{10}$ seconds, which is around 584 years! So, it is clearly impractical to find the global minimum of these synthetically generated random QUBO problems. However, we would prefer an embedding algorithm that consistently gets us a smaller value of the QUBO objective function (Eq. 1). In Fig. 8 our embedding algorithm, shown by the red line, consistently converges to a smaller objective function value (shown on the Y axis) as compared to D-Wave's embedding algorithm, shown by the blue line. And this is consistent across all problem sizes from 2-variable QUBO problems to 64-variable QUBO problems.

Another performance metric that is of interest is the *Maximum Chain Length* of the qubit chains, i.e., the maximum number of qubits that have been mapped to any given QUBO variable. This is of significance because as the qubit chains get longer and longer, it is difficult to maintain them on the hardware. As a result, these long qubit

chain have a tendency of breaking during the quantum annealing process. The breakage of these long chains severely compromises the final solution obtained. Because our embedding algorithm is completely deterministic, it is possible to analytically compute the Maximum Chain Length for any embedding configuration as follows:

$$\text{Maximum chain length} = \left\lfloor \frac{V}{L} \right\rfloor \quad \text{if } V \leq L + 1 \tag{11}$$

$$= \left\lceil \frac{V}{L} \right\rceil \quad \text{if } V > L + 1 \tag{12}$$

where V is the number of variables in a QUBO problem, and L is the number of qubits in a line of Chimera block as described in Fig. 1. For instance, in Fig. 2a–d, the Maximum Chain Length is 1 because $L = 4$ and V goes from 2 through 5. Similarly, in Fig. 3a, b, Maximum Chain Length is 2, and in Fig. 4, it is 3 because $V = 9$ and $L = 4$. So, on the real hardware of the D-Wave 2000Q, which has $M = 16$ rows and $N = 16$ columns of Chimera blocks ($L = 4$), a total of 64 QUBO variables can be accommodated having all-to-all connectivity and for a 64 variable QUBO problem ($V = 64$), the Maximum Chain Length in an embedding obtained from our algorithm would be 16 (as per Eq. 12). This is consistent with other embedding approaches [17,18].

Thus, our embedding algorithm is seen to outperform D-Wave's embedding algorithm comprehensively and consistently across the following three performance metrics: embedding time, number of qubits used and accuracy, and it generates embeddings that have a Maximum Chain Length that is consistent with other embedding approaches.

8 Conclusion

Adiabatic quantum computers like the D-Wave 2000Q quantum computer can potentially solve the QUBO problem, which is an NP-hard problem, using the quantum mechanical process called quantum annealing. Being able to solve the QUBO problem would mean we can solve virtually any NP-hard problem like TSP, protein folding, MSA, etc., thereby enabling significant scientific progress. However, before we can solve QUBO problems on quantum computers, they must be *embedded* onto the Chimera graph, representing qubits of adiabatic quantum computers. Embedding refers to the process of mapping variables of QUBO problem onto physical qubits of the adiabatic quantum computers, which are arranged in a Chimera graph configuration. An efficient embedding algorithm lets us embed a QUBO problem fast, uses less qubits and gets the objective function value closer to the global minimum value. In this work, we first propose a coupling approach to couple two or more qubits (i.e., make them behave in sync with each other) in Sect. 5, mathematically prove that it works, and then present the coupling algorithm (Algorithm 1) to show that coupling can be done in quadratic time and space. Next, we use our coupling approach to couple qubits in our embedding algorithm (Algorithm 4) and also present all of its subroutines (Algorithms 2, 3) in Sect. 6. Finally, we compare the performance of our embedding

algorithm to that of D-Wave's embedding algorithm in Sect. 7. For this comparison, we use three performance metrics: embedding time, qubit footprint, and accuracy. We show that our embedding algorithm convincingly outperforms D-Wave's embedding algorithm on all the three performance metrics.

References

1. Adachi, S.H., Davenport, D.M., Henderson, M.P.: Quantum-assisted training of neural networks. US Patent App. 14/702,203 (2015)
2. Albash, T., Lidar, D.A.: Adiabatic quantum computation. Rev. Modern Phys. **90**(1), 015002 (2018)
3. Amin, M.H.: Methods of adiabatic quantum computation. US Patent 8,504,497 (2013)
4. Amin, M.H., Steininger, M.F.: Adiabatic quantum computation with superconducting qubits. US Patent 7,135,701 (2006)
5. Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press, Princeton (2006)
6. Barends, R., Shabani, A., Lamata, L., Kelly, J., Mezzacapo, A., Las Heras, U., Babbush, R., Fowler, A.G., Campbell, B., Chen, Y., et al.: Digitized adiabatic quantum computing with a superconducting circuit. Nature **534**(7606), 222 (2016)
7. Baruah, S., Bertogna, M., Buttazzo, G.: Multiprocessor Scheduling for Real-time Systems. Springer, Berlin (2015)
8. Biamonte, J.D., Berkley, A.J., Amin, M.: Physical realizations of a universal adiabatic quantum computer. US Patent 8,234,103 (2012)
9. Bian, Z., Chudak, F., Israel, R.B., Lackey, B., Macready, W.G., Roy, A.: Mapping constrained optimization problems to quantum annealing with application to fault diagnosis. Front. ICT **3**, 14 (2016)
10. Bian, Z., Chudak, F., Macready, W., Roy, A., Sebastiani, R., Varotti, S.: Solving sat and maxsat with a quantum annealer: foundations and a preliminary report. In: International Symposium on Frontiers of Combining Systems, pp. 153–171. Springer, Berlin (2017)
11. Blum, A., Rivest, R.L.: Training a 3-node neural network is NP-complete. In: Advances in Neural Information Processing Systems, pp. 494–501 (1989)
12. Boothby, T., King, A.D., Roy, A.: Fast clique minor generation in chimera qubit connectivity graphs. Quantum Inf. Process. **15**(1), 495–508 (2016)
13. Britt, K.A., Humble, T.S.: High-performance computing with quantum processing units. ACM J. Emerg. Technol. Comput. Syst. (JETC) **13**(3), 39 (2017)
14. Cai, J., Macready, W.G., Roy, A.: A practical heuristic for finding graph minors. Preprint (2014). arXiv:1406.2741
15. Carrillo, H., Lipman, D.: The multiple sequence alignment problem in biology. SIAM J. Appl. Math. **48**(5), 1073–1082 (1988)
16. Chickering, D.M., Geiger, D., Heckerman, D., et al.: Learning Bayesian networks is NP-hard. Technical Report, Citeseer (1994)
17. Choi, V.: Minor-embedding in adiabatic quantum computation: I. The parameter setting problem. Quantum Inf. Process. **7**(5), 193–209 (2008)
18. Choi, V.: Minor-embedding in adiabatic quantum computation: II. Minor-universal graph design. Quantum Inf. Process. **10**(3), 343–353 (2011)
19. Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, pp. 151–158. ACM, New York (1971)
20. Cristianini, N., Shawe-Taylor, J., et al.: An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press, Cambridge (2000)
21. Dasgupta, S.: The hardness of k-means clustering. Department of Computer Science and Engineering, University of California, San Diego (2008)
22. Dill, K.A., MacCallum, J.L.: The protein-folding problem, 50 years on. Science **338**(6110), 1042–1046 (2012)
23. Dubois, O., Dequen, G.: A backbone-search heuristic for efficient solving of hard 3-sat formulae. IJCAI **1**, 248–253 (2001)
24. Esteve, D., Vion, D., Devoret, M., Urbina, C., Joyez, P., Pothier, H., Orfila, P.F., Aassime, A., Cottet, A.: Superconducting quantum-bit device based on josephson junctions. US Patent 6,838,694 (2005)

25. Etschmaier, M.M., Mathaisel, D.F.: Airline scheduling: an overview. Transp. Sci. **19**(2), 127–138 (1985)
26. Goodrich, T.D., Sullivan, B.D., Humble, T.S.: Optimizing adiabatic quantum program compilation using a graph-theoretic framework. Quantum Inf. Process. **17**(5), 118 (2018)
27. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the 28th Annual ACM Symposium on Theory of Computing, pp. 212–219. ACM, New York (1996)
28. Hamilton, K.E., Humble, T.S.: Identifying the minor set cover of dense connected bipartite graphs via random matching edge sets. Quantum Inf. Process. **16**(4), 94 (2017)
29. Huang, W., Yu, J.X.: Investigating TSP heuristics for location-based services. Data Sci. Eng. **2**(1), 71–93 (2017)
30. Imamog, A., Awschalom, D.D., Burkard, G., DiVincenzo, D.P., Loss, D., Sherwin, M., Small, A., et al.: Quantum information processing using quantum dot spins and cavity QED. Phys. Rev. Lett. **83**(20), 4204 (1999)
31. Kane, B.E.: A silicon-based nuclear spin quantum computer. Nature **393**(6681), 133 (1998)
32. Keating, T., Goyal, K., Jau, Y.Y., Biedermann, G.W., Landahl, A.J., Deutsch, I.H.: Adiabatic quantum computation with Rydberg-dressed atoms. Phys. Rev. A **87**(5), 052314 (2013)
33. King, A.D., Israel, R.B., Bunyk, P.I., Boothby, T.J., Reinhardt, S.P., Roy, A.P., King, J.A., Lanting, T.M., Evert, A.J.: Systems and methods for embedding problems into an analog processor. US Patent App. 15/487,295 (2017)
34. Kleinberg, J., Tardos, E.: Algorithm Design. Pearson Education India, Noida (2006)
35. Klymko, C., Sullivan, B.D., Humble, T.S.: Adiabatic quantum programming: minor embedding with hard faults. Quantum Inf. Process. **13**(3), 709–729 (2014)
36. Landahl, A.: Adiabatic quantum computing. In: APS Four Corners Section Meeting Abstracts (2012)
37. Lewis, M., Glover, F.: Quadratic unconstrained binary optimization problem preprocessing: theory and empirical analysis. Networks **70**(2), 79–97 (2017)
38. Li, Y., Willer, C., Sanna, S., Abecasis, G.: Genotype imputation. Ann. Rev. Genom. Hum. Genet. **10**, 387–406 (2009)
39. Macready, W., Roy, A.P.: Systems and methods that formulate problems for solving by a quantum processor using hardware graph decomposition. US Patent 9,875,215 (2018)
40. Nielsen, M.A., Chuang, I.: Quantum Computation and Quantum Information. AAPT (2002)
41. Papadimitriou, C.H.: Computational Complexity. Wiley, London (2003)
42. Pittman, T., Jacobs, B., Franson, J.: Probabilistic quantum logic operations using polarizing beam splitters. Phys. Rev. A **64**(6), 062311 (2001)
43. Pla, J.J., Tan, K.Y., Dehollain, J.P., Lim, W.H., Morton, J.J., Jamieson, D.N., Dzurak, A.S., Morello, A.: A single-atom electron spin qubit in silicon. Nature **489**(7417), 541 (2012)
44. Potok, T.E., Schuman, C.D., Young, S.R., Patton, R.M., Spedalieri, F., Liu, J., Yao, K.T., Rose, G., Chakma, G.: A study of complex deep learning networks on high performance, neuromorphic, and quantum computers. In: Workshop on Machine Learning in HPC Environments (MLHPC), pp. 47–55. IEEE, New York (2016)
45. Rieffel, E.G., Venturelli, D., O'Gorman, B., Do, M.B., Prystay, E.M., Smelyanskiy, V.N.: A case study in programming a quantum annealer for hard operational planning problems. Quantum Inf. Process. **14**(1), 1–36 (2015)
46. Roy, A.P.: Systems and methods that formulate embeddings of problems for solving by a quantum processor. US Patent 9,501,747 (2016)
47. Shmygelska, A., Hoos, H.H.: An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem. BMC Bioinformat. **6**(1), 30 (2005)
48. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: 1994 Proceedings., 35th Annual Symposium on Foundations of Computer Science, pp. 124–134. IEEE, New York (1994)
49. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Rev. **41**(2), 303–332 (1999)
50. Thom, M.C., Roy, A.P., Chudak, F.A., Bian, Z., Macready, W.G., Israel, R.B., Boothby, T.J., Yarkoni, S., Xue, Y., Korenkevych, D., et al.: Systems and methods for analog processing of problem graphs having arbitrary size and/or connectivity. US Patent App. 15/448,361 (2017)
51. Turing, A.M.: On computable numbers, with an application to the entscheidungsproblem. a correction. Proc. Lond. Math. Soc. **2**(1), 544–546 (1938)

52. Venturelli, D., Mandra, S., Knysh, S., O'Gorman, B., Biswas, R., Smelyanskiy, V.: Quantum optimization of fully connected spin glasses. Phys. Rev. X **5**(3), 031040 (2015)
53. Zagoskin, A.M.: Qubit using a josephson junction between s-wave and d-wave superconductors. US Patent 6,459,097 (2002)
54. Zagoskin, A.M.: Quantum computing method using magnetic flux states at a Josephson junction. US Patent 6,563,311 (2003)
55. Zaribafiyan, A., Marchand, D., REZAEI, S.S.C.: Method and system for generating an embedding pattern used for solving a quadratic binary optimization problem. US Patent App. 15/344,054 (2017)