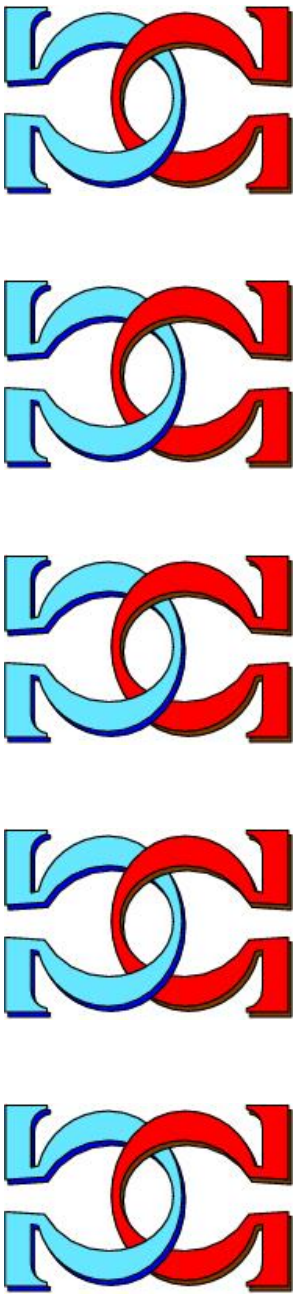

CDMTCS
Research
Report
Series

Graph Minor Embeddings for D-Wave Computer Architecture

Zongcheng Yang
Michael J. Dinneen
Department of Computer Science,
University of Auckland,
Auckland, New Zealand

CDMTCS-503
November 2016

Centre for Discrete Mathematics and
Theoretical Computer Science

Graph Minor Embeddings for D-Wave Computer Architecture

Zongcheng (Lucas) Yang and Michael J. Dinneen
Department of Computer Science, University of Auckland, Auckland, New Zealand
zyan632@aucklanduni.ac.nz mjd@cs.auckland.ac.nz

Abstract

The D-Wave system architecture is designed to deal with quantum annealing to solve computational problems. To run or solve a problem by the D-Wave hardware, we need to first transform the problem into an Ising or Quadratic Unconstrained Binary Optimization (QUBO) instance, then embed Hamiltonians (logical qubit relationships) onto the actual D-Wave hardware which is currently based on Chimera graphs (physical qubit couplings). In order to have better performance of D-Wave's quantum annealing, an efficient algorithm to find good embeddings needs to be obtained. In this paper, we present some heuristic algorithms for minor embedding an arbitrary guest graph onto a host Chimera graph. Our implementations show these new algorithms are practical for sparse graphs with hundreds of vertices. In general, for a given minor embedding, we tried to minimize the maximum number physical qubits representing by any logical qubit and/or the total number of physical qubits used.

1 Introduction

Why is computing important? Because computing can solve things that human beings are not capable of solving due to complexity or the possibility of inconsistent errors, or time. After realizing the importance of computing, both practical and theoretical research continues, people are spending more resources in an effort to develop a better computing model. By Moore's law [24], the number of transistors in an integrated circuit doubles approximately every two years. Although the exponential speed-ups held steady from 1975 until around 2012, it was predicted that growth would slow around recent years since current hardware approaches the physical limits. People are trying to use an alternative technology called quantum computing to maintain the increment.

The primary difference between classical and quantum computing is that instead of using binary bits, quantum computing use quantum bits which can be in a superposition of multiple states. It offers a possibility of solving problems which cannot be solved by using the current classical computers. In this thesis, we will begin with a brief introduction to quantum computing and the adiabatic quantum computer developed by D-Wave Systems [28]. D-Wave claims they have built the world's first commercial quantum computer, which could use the adiabatic method to solve optimization problems.

In order to use the D-Wave computer to solve problems, we have to first transform the problem into an Ising or QUBO instance (explained in Sections 2.2.1 and 2.2.2), then embed Hamiltonians (logical qubit relationships) onto the D-Wave Chimera graphs. It should be noted that the problem of finding minor embedding for an arbitrary graph is an NP-hard problem [26], and searching for a graph minor within a graph which has hundreds of vertices has received little attention in the graph theory literature. As a result, all of the current known exact algorithms are only practical for tens of vertices. In this background, the D-Wave people provide the very first heuristic algorithm for this problem [5]. Based on the heuristic idea we designed and implemented a new heuristic algorithm, and this algorithm is practical for finding minors in graphs which have up to two thousand vertices.

The paper is organized as follows. In the first section, we give an introduction. and provide some preliminary definitions for the minor embedding problem. We then give a brief overview of quantum computing and introduce the adiabatic quantum computer developed by D-Wave Systems, as well as an explanation of how the D-Wave computer works. We follow this with a background section of some current algorithms for the minor embedding problem. Then in our main two sections we present two new heuristic algorithms for embedding arbitrary graphs into the Chimera of the D-Wave architecture and compare the performance of our new algorithms with the aid of tables and plots. Later we compare the running time of these algorithms and discuss the embedding quality of the embeddings obtained by these algorithms. Finally we present the some conclusions and list some limitations for future work.

1.1 Preliminary Definitions

The basic graph definitions may be found in many introductory algorithms textbooks such as [11]. We now recall a few of the graph theoretical terms that we use in this report.

Definition 1 *A **graph** $G = (V, E)$ consists of a finite set of vertices V and a set of edges E, with each edge being incident with an unordered pair of the distinct vertices. The **order** of G is the number of vertices in G. The **size** of G is the number of edges in G. Let u, v be two distinct vertices in a simple undirected graph G, we use uv to denote an **edge** between u and v in G. A **loop** is an edge that connects a vertex to itself.*

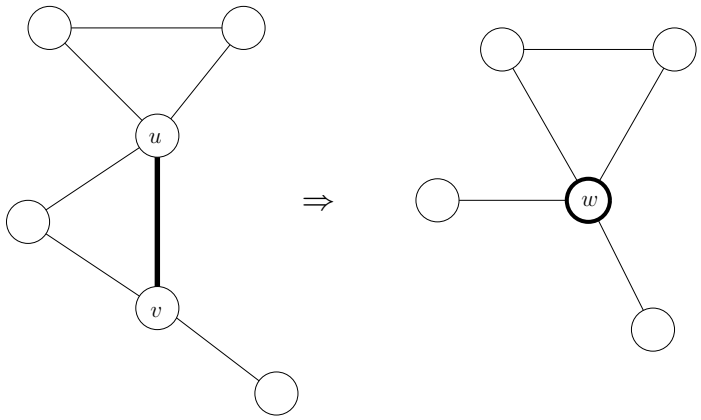

Figure 1: Contracting an edge uv which is merged into a new vertex w.

Definition 2 *The **girth** of a graph is the length of any shortest cycle contained in the graph. If the graph does not contain any cycles, its girth is defined to be infinity.*

Definition 3 *The **eccentricity** $\epsilon(v)$ of a vertex v is the greatest distance between v and any other vertex. The **diameter** of G is the maximum eccentricity of any vertex in G.*

In this paper, only undirected graphs which have no loops and no multiple edges are considered.

1.1.1 Graph minor and embeddings

In graph theory, a graph H is called a **minor** of a graph G if H can be formed from G by deleting edges and vertices and by contracting edges. Now we give the formal definition:

Definition 4 *Graph H is a **minor** of graph $G = (V_G, E_G)$ if a graph isomorphic to H can be obtained from G by repeating the following operations:*

1. *Removing any vertex $v \in V_G$ and any edges $e \in E_G$ in which e is incident to v*
2. *Removing any edge $uv \in E_G$.*
3. *Contracting any edge $uv \in E_G$. This operation removes u,v from V_G and makes a new vertex w incident to all the vertices u,v are incident to.*

An edge contraction is an operation which removes an edge from a graph while simultaneously merging the two vertices that it previously joined. Edge contraction is a fundamental operation in the theory of graph minors. The edge contracting operation are illustrated in Figure 1 with edge uv becoming a new vertex w.

Our main objective is to develop a practical minor embedding algorithm. We now formally define the minor containment problem.

Problem 5 Minor Containment
Input: Graphs $G = (V_1, E_1)$ and $H = (V_2, E_2)$.
Question: Does G contain a minor which is isomorphic to H?

Definition 6 *A **minor embedding** of a guest graph H in a host graph G is defined by the function $\phi : V(H) \to 2^{V(G)}$ such that:*

1. *For each vertex $v \in V(H)$, the subgraph induced by $\phi(v)$ in G is connected.*
2. *For any two distinct vertices u and v in H, $\phi(u) \cap \phi(v) = \emptyset$.*
3. *Suppose u and v are adjacent in H, then there is at least one edge between $\phi(u)$ and $\phi(v)$ in G.*

We call $\phi(v)$ the **vertex model** of v and say that $\phi(v)$ represents v in G.

The general minor containment problem is to decide for any two graphs G and H, if H is a minor of G. Whenever H is fixed, Robertson and Seymour have recently proven the following theorem [22, 23]:

Theorem 7 *For any fixed graph H, there is an algorithm to decide if H is a minor of an input graph G that runs in time $O(n^3)$, where n is the number of vertices of G.*

Robertson-Seymour Theorem is an important theoretical theorem, it implies that for arbitrary host graph G and a fixed guest graph H, there is a polynomial-time $O(n^3)$ algorithm for determining whether or not the graph G contains graph H as a minor. Unfortunately, Robertson and Seymours algorithm is not practical since the actual polynomial-time bound has a large hidden constant. However, when both H and G are part of the input, the minor containment problem is NP-Complete [26]. In this report the focus is practicality: for given graphs H and G, our goal is to design and implement a practical algorithm to determine if guest H is a minor of host G, and give the mapping if it exists.

2 Quantum Computing and D-Wave Computer Architecture

Quantum computing is typically defined as theoretical computation systems which make direct use of quantum-mechanical phenomena, such as superposition and entanglement, to perform operations on data. Quantum computation was first introduced

by Yuri I. Manin [18] in 1980 and Richard Feynman [12] in 1982. It uses quantum bits (qubits) which can represent a zero, a one, or any quantum superposition of those two qubit states. The classical computation requires data to be encoded into binary digits (bits) using a classical physical system, each of which is in one of two states (0 or 1). Unlike classical binary bits, qubits can be operated together in superposition to perform massively parallel computations, on all its possible quantum states simultaneously in a single processing unit, while classical parallel computers have to run with many processing units [6].

Based on the property of superposition, we may use quantum computers to efficiently solve problems which no classical systems would be able to solve within a reasonable amount of time. There is a class of problems called BQP (Bounded error, Quantum, Polynomial time), which can be solved efficiently using a quantum computer with high probability, and BQP is believed to be different from BPP (Bounded error, Probabilistic, Polynomial time), the class of problems considered to be solved by classical computers. Nowadays, people are funding quantum computing research on both practical and theoretical areas, seeking and developing on quantum computers for business, trade, education, and military purposes.

2.1 Quantum algorithms

We should note that the undecidable problems in a classical system remain undecidable using quantum devices, but quantum algorithms might be able to solve some problems faster than classical algorithms. In 1985, David Deutsch [10] established the very first quantum computer model, which is able to simulate any other quantum computer with at most a polynomial slowdown. After Deutsch, many quantum algorithms have been designed for different applications with hundred of thousands times speed-up over the best known classical algorithm based on the classical machine. In 1994, Shor [25] devised an exponential-faster quantum algorithm for factoring integers in $O(n^3)$ time with respect to arbitrary input size, there is no known classical polynomial algorithm for factoring (the problem is still open). Another famous quantum algorithm example is Grover's algorithm [13], Grover described a quadratic faster method over the best classical algorithm for searching an unsorted n-entry database in $O(\sqrt{n})$ time.

2.1.1 Adiabatic quantum computation

In quantum computation theory, there exist multiple types of quantum computers which are based on different computation models to perform calculations. In this paper, we focus on the adiabatic model of computing.

Adiabatic Quantum Computation (AQC) is a theoretical model that based on Born-Fock Adiabatic Theorem [2]. It uses the fact [8] the quantum mechanical system want to stay in the lowest possible energy level. Initially, the quantum qubits start

from a ground state of one Hamiltonian (real-valued interaction matrix), then slowly transform to another until they are able to solve the given problem. During the entire computation, the system must stay in a valid ground state. It solves the given problem by encoding the set of candidate solutions to a problem as the ground state of a Hamiltonian [21].

2.1.2 Quantum annealing

Quantum Annealing (QA) was first proposed in 1998 [15], which used as a meta-heuristic for solving minimization problems. In a quantum system when a particle transfers through a barrier, we can compare the effect to that of a ball rolling over a hill. If the ball does not have enough energy to get over the hill, the ball can "borrow" energy from the hill and travel through it to the other side. By replacing the hill with the given objective function over a set of candidate solutions, we will see the optimal solution is comparable to finding the lowest valley in the range of mountains [19].

2.2 D-Wave computer

The D-Wave computers are produced by the Canadian company D-Wave Systems which specializes in making quantum computers. Based on the idea of quantum annealing and using current hardware, they claim to have built the world's first commercially available quantum computer D-Wave One in 2010 up to the current model D-Wave 2X in 2016 [28]. The computer architecture consists of qubits arranged in cells (shown in Figure 2) and the multiple cells connected together in a grid-like host configuration as a *Chimera graph*. More details given later in Section 2.2.3.

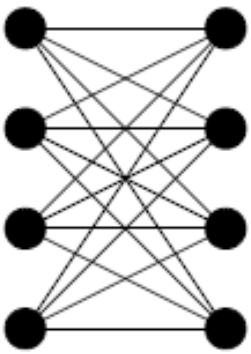

Figure 2: The graph $K_{4,4}$ of qubits for a single cell.

2.2.1 Ising model

D-Wave Systems uses the Ising model to solve discrete optimization problems, it is the standard representation for problems for the D-Wave computer. The model was named after the physicist Ernst Ising. This mathematical model in statistical mechanics consists of discrete variables, these variables represent magnetic dipole moments of atomic spins that can be in one of two states: either $s_i = +1$ for spin-up or $s_i = -1$ for spin-down. The D-Wave hardware can be viewed as a hardware heuristic which minimizes Ising objective functions using a physically realized version of quantum annealing. The Ising energy minimization problem of n variables $s_i \in \{-1, +1\}$ is given by:

$$s^* = \min_s \sum_{(i,j)\in E} s_i J_{(i,j)} s_j + \sum_{i\in V} h_i s_i, \text{where } s_i \in \{-1, +1\}.$$

The spin variables interact with neighbors in a graph $G = (V, E)$, where each edge $(i, j) \in E$ has a non-zero energy interaction $J_{(i,j)}$ and each vertex $i \in V$ has an external energy h_i. Without loss of generality it is convenient to assume that J is upper-triangular of the adjacency matrix of G. If $J_{(i,j)} > 0$ the edge is positive, and if $J_{(i,j)} < 0$ the edge is negative. This graph captures the dependencies between variables which can be exploited to more efficiently minimize the Ising objective functions. The details can be checked in D-Wave System manual [27].

2.2.2 QUBO model

The Ising model tends to be favored by physicists since it uses Ising spin $\{-1, +1\}$. We now describe another model for computer scientists, using binary value $\{0, 1\}$ instead. The QUBO (Quadratic Unconstrained Binary Optimization) [6] is an NP-hard mathematical problem consisting in the minimization of a quadratic objective function $z = x^{\mathrm{T}} Q x$, where x is a n-vector of binary variables and Q is a symmetric $n \times n$ matrix:

$$x^* = \min_x \sum_{i\geq j} x_i Q_{(i,j)} x_j, \text{where } x_i \in \{0, 1\}.$$

The spin variables in the Ising model and the binary variables in the QUBO mode are simply related to each other through the transformation $s = 2x-1$ or $x = (s+1)/2$. In the D-Wave utility packs [27] provide code which applies this translation, we may freely translate between whichever Ising/QUBO representation is more convenient.

2.2.3 Chimera graph

To solve the general Ising/QUBO problems using the machine, we have three known constraints imposed by the current hardware: a small number of qubits, sparse qubit

connectivity, and limited parameter precision. Due to these constraints, only certain pairs of qubits are connected in the hardware.

More formally, a $M \times N \times L$ Chimera graph has $2MNL$ vertices, it consists of an $M \times N$ two-dimensional lattice of blocks each with $2L$ vertices and all of these vertices are arranged into the bipartite graph $K_{L,L}$. Each of these blocks has L connections to each of its neighboring blocks and most of them have four neighbors: right, left, up and down, except for blocks on the edge of the lattice. An $8 \times 8 \times 4$ Chimera graph for the 512 qubit D-Wave Two hardware is shown in Figure 3.

2.3 Using D-Wave to solve QUBO/Ising problems

Our motivation for researching heuristic minor embedding algorithm stems from its importance in solving problems to D-Wave's quantum computer. The D-Wave computer solves problems in the form on an Ising or QUBO formulation described in Sections 2.2.1 and 2.2.2. This is the quadratic optimization [3] over the Boolean variables (the qubits) we have to map to the D-Wave hardware.

There are two main steps in converting a problem into one which can be solved by using the D-Wave hardware.

1. Transform the problem into an Ising or QUBO instance [17].
2. In order to map to the Chimera graph discussed in Section 2.2.3, we use multiple qubits in the hardware graph (physical qubits) to represent the same variable in the Ising model (logical qubits). Hence, the problem must be minor embedded so that it fits within the hardware.

Since only a very limited number of qubits are available on the D-Wave machine, it is really important to find some good embeddings for using the D-Wave computer to solve problems. In general, we want either to minimize the maximum number physical qubits representing any logical qubit, or minimize the total number of physical qubits. It should be noted that a higher quality of embedding leads directly to better hardware performance (i.e. the higher probability of finding or getting closer to an optimal solution).

3 Related Work

We now consider some of the existing algorithms done on the minor embedding problem. First of all, we will give an exact minor embedding algorithm which guarantees to find the solution without any constraints. Next, we will talk about the current heuristic search approach which is feasible for hundreds of vertices.

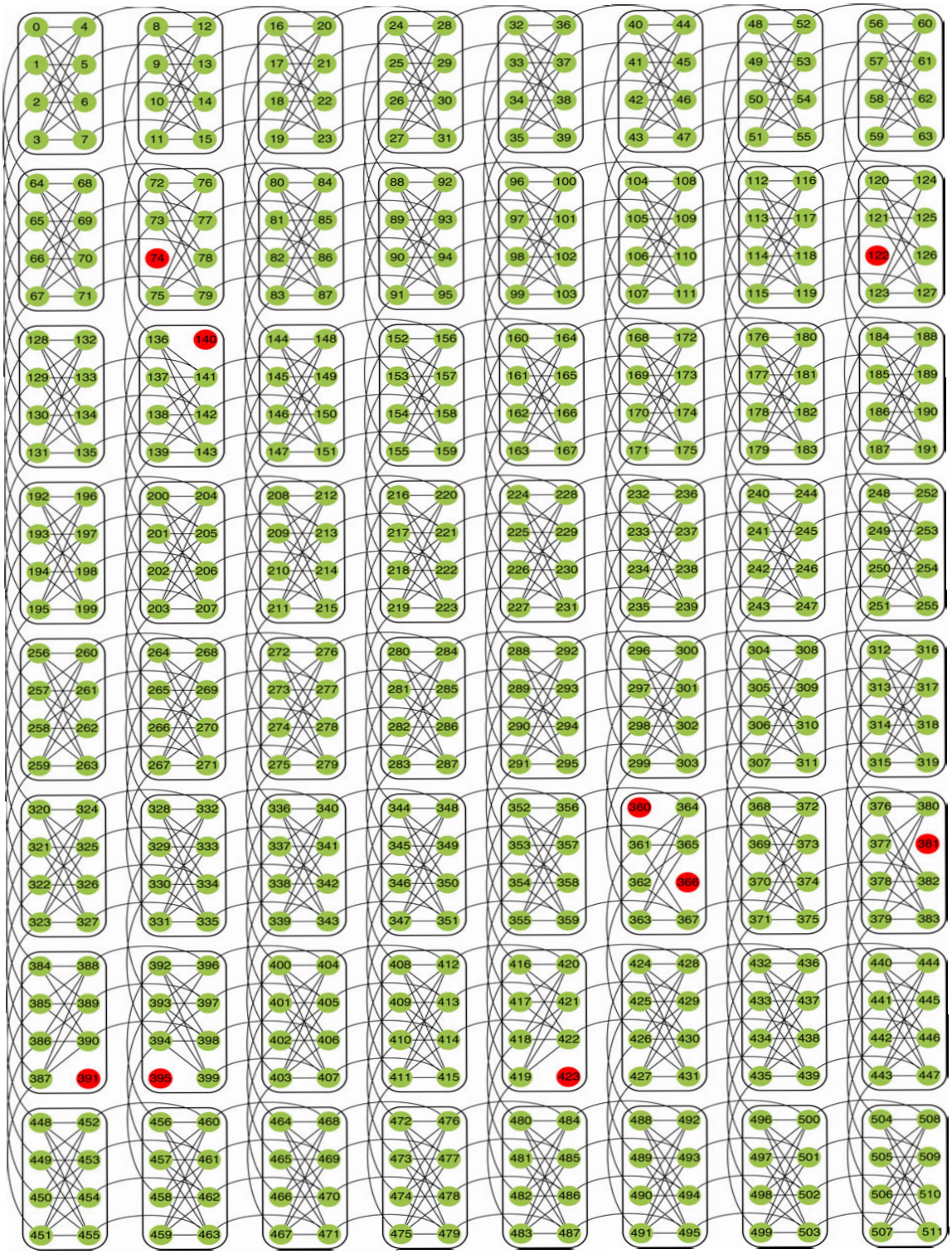

Figure 3: An $8 \times 8 \times 4$ Chimera graph with 512 qubits taken from [20]. Red circles denote faulty qubits in the hardware.

3.1 Exact minor containment algorithm

Based on the theorem by Robertson and Seymour [22, 23], we know there is a polynomial time $O(n^3)$ algorithm to solve the containment problem when the guest graph is fixed. However, the actual running time is not very practical because of the existence of large hidden constant. In 2000, Xiong and Dinneen [30] present a simple algorithm to decide whether a (guest) graph H is a minor of a (host) graph G, where G is connected:

- **Step 1:** Generate all feasible vertex maps from H onto G.
- **Step 2:** For each map, contract edges in G where the endpoints have the same image in H. This gives a resulting graph G'.
- **Step 3:** Test if H is a subgraph of G'. If it is, return true, otherwise iterate step 2.
- **Step 4:** Return false.

A vertex map is a function which maps $V(H)$ to $V(G)$. This approach requires a two-phase process: the first is to generate all feasible vertex models, the second is a validity check. The first phase, based on the idea of set partitions we could easily generating all feasible mappings. For each partition, validity checking is to ensure that all of the vertices in any sub-partition form a connected induced subgraph in the host graph. If all of the induced subgraphs are connected that means there is a way to contract all the vertices in the given set partition into a valid graph G'.

3.1.1 Set partition approach

For the minor embedding problem, the number of vertices in the host graph should be greater or equal to the number of vertices of the guest graph, since extra vertices of host graph can be reduced by edge contractions, this is a many-to-one map. In order to generate all feasible vertex models, we introduce the concept of set partition [16].

Definition 8 *A **set partition** of the set $V = \{1, 2, \ldots, n\}$ is a collection $B_1, B_2, \ldots, B_j$ of disjoint subsets of V whose union is V. Each B_i is called a **block**.*

For any input graph G with order n and input graph H with order k where $k \leq n$, we use $S(n)$ to denote the set of all partitions of $\{1, 2, ..., n\}$ into non-empty subsets, $S(n, k)$ is used to denote the set of all partitions of $\{1, 2, ..., n\}$ into exactly k non-empty subsets, using the idea of set partitions, the algorithm partitions the host graph G into k parts, where periods separate the individual sets. For $n = 4$,

1. $S(4, 1)$**:** 1234

2. $S(4,2)$: 123.4 124.3 134.2 1.234 12.34 13.24 14.23

3. $S(4,3)$: 1.2.34 1.24.3 1.4.23 14.2.3 13.2.4 12.3.4

4. $S(4,4)$: 1.2.3.4

Obviously and naturally, all the partitions above have bijections with the vertex models we used in our algorithm. In order to generate all partitions, the "restricted growth string" can be used based on this sort order. We will not present the details of the algorithm for generating $S(n,k)$, this can be found in [30] for interested readers.

Once we have a set partition of $S(n,k)$, there are $k!$ possible maps and all of them should be tested. For each map, we first pretest to determine if those vertices in G could be formed a connected subgraph. If it is no, then we just skip to the next map. if it is yes, then we have to make sure if each edge in the guest graph H is embedded in the resulting graph G'. To check embedding, we just pick every edge uv in H, if there is a corresponding edge $u'v'$ in G which $\phi(u') = u$ and $\phi(v') = v$. If any of these testings failed, then H is not a minor of G. Otherwise, we can conclude that H is a minor of G. In this way, we do not have to do any edge contractions in G since this operation is very time-consuming.

Instead of doing a two-phase partitioning process, Datt [9] presented an optimized partitioning way which uses a single phase method that enumerates all valid set partitions while avoiding bad sets. A bad set is a partitioning such that there is a sub-partition that cannot be contracted into a single vertex. Datt also gives the partitioning procedure that proves to be faster than the process implemented by Xiong and Dinneen, complete with proof of correctness in his thesis.

As a result, all of the known exact algorithms for the minor-embedding problem are only practical for up to tens of vertices in the host graphs. Recently D-Wave released the general availability of their D-Wave 2X computer, which has a $12 \times 12 \times 4$ Chimera graph with 1,152 qubits architecture. This means we have to find an algorithm to solve the minor-embedding problem for up to a thousand vertices in the host graph, and this is unsolvable by considering exact algorithms.

3.2 Heuristic search approach

The D-Wave software package [27] provides an algorithm for finding a minor embedding. However, the method it uses is not given to the public. Now we introduce a heuristic approach which could solve the minor embedding problem in a reasonable amount of time for the D-Wave computer architecture.

In computer science and artificial intelligence, a *heuristic* is a technique designed for solving problems more quickly when classic methods cannot give an exact solution in a reasonable time frame. This can be achieved by trading optimality, completeness, accuracy, or precision for speed. In order to find embedding with hundreds

of vertices graph, the engineers from D-Wave Systems: Cai, Macready and Roy [5] obtain a heuristic algorithm for finding a graph H as a minor of a graph G with some probability. Before this paper, the heuristic search idea for arbitrary guest graph into hundreds of vertices graph has received little attention in the graph theory literature, possibly due to a lack of known applications. We now give the high-level description of their heuristic algorithm.

Initially, we randomize vertex order in a guest H, then for each vertex v in the randomize order, the heuristic algorithm proceeds by iteratively to construct a vertex model $\phi(v)$, based on the weighted shortest path distances to its neighbors' vertex models; if none of v's neighbors have vertex models yet, we can randomly choose a single vertex from G to be the vertex model for v. At each step, we have to make sure no two of the vertex models share a same vertex in G. Once all the vertices in the guest H matched to some valid vertex models, we find the minor. Otherwise, we rerun the algorithm, if can not find the minor in a certain time, then we suspect no minor embedding exists.

The authors claim this algorithm has been implemented by D-Wave Systems and it has proven to be effective in finding graph minors with hundreds of vertices. Our main algorithm will be based on this heuristic idea, with the hope of finding some better embeddings with respect to various criteria.

4 Improved Heuristic Minor-embedding Algorithm

In this section, we will present some heuristic approaches for embedding arbitrary graphs into the Chimera graph of the D-Wave architecture. Once the embedding has been found, we also give some ideas to minimize the total number of physical qubits used, or the maximum number physical qubits representing any logical qubit.

4.1 A basic heuristic to find minor embeddings

We now give a basic heuristic search of finding an embedding into an arbitrary sized Chimera graph, and this method was originally given by Cai *et al.* [5] in 2014.

For arbitrary input guest graph H and host graph G, recall the *vertex model* in Definition 6, our heuristic algorithm loops through every vertex v in a guest graph H iteratively, and tries to construct the corresponding vertex model $\phi(v)$. We will give the main approach first, then describe how to construct the minimal vertex model.

The heuristic search is described as follows. In the first step, we randomize the vertex order in the guest graph H. For each vertex v in the randomized order, our algorithm proceeds iteratively to construct the minimal vertex model $\phi(v)$ based on the *weighted shortest path search* (explained in the following section) to its neighbors. In the case for some vertices in the guest graph H which do not have any neighbor

vertex model yet, we just randomly select a single vertex in host graph G to be its vertex model. After the first step, we run a fixed number of iterations to reconstruct the vertex models. For each step, try to use only unused vertices for our new vertex model. At last, if there is no overlap between all the vertex models, then we have found a minor embedding.

4.2 Minimal vertex model

The key word "minimal" here does not necessarily mean that we found the best vertex model in terms of global embeddings. We are trying to find a good vertex model with a minimum number of vertices used each step. Suppose we are looking at the vertex model $\phi(v)$ for $v \in V(H)$, and v has some neighbors $u_1, ..., u_k$ which already have vertex models $\phi(u_1), ..., \phi(u_k)$. We want to minimize the number of vertices used in $\phi(v)$ that ensures $\phi(v)$ has an edge between each of $\phi(u_1), ..., \phi(u_k)$. To do this, we have to calculate the shortest-path distance from all the unused vertice in G to every neighbor vertex model $\phi(u_1), ..., \phi(u_k)$. Using a cost function $c(g, i)$ to store the distance information, we compute the total sum of distances $\sum_i c(g, i)$ and select the vertex r^* which has smallest sum as the root of our new vertex model $\phi(v)$. Finally we determine the shortest path from r^* to all its neighbors' vertex models and use the union of those paths as the new model $\phi(v)$. An example is shown in Figure 4.

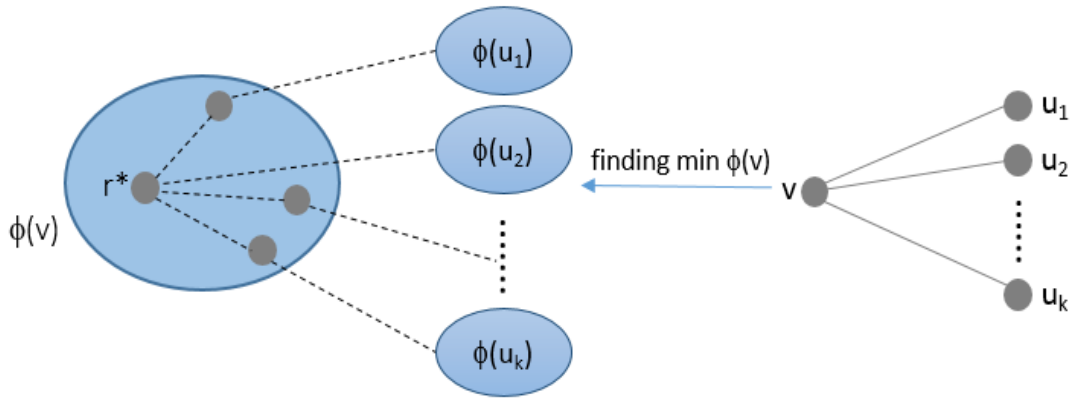

Figure 4: An illustration of finding vertex model $\phi(v)$. Based on the minimum sum distance to select root r^*, then take the union of shortest path from r^* to all neighbors' vertex model as $\phi(v)$.

Unfortunately, for most of the cases we cannot find such a root r^* that the union of the shortest path from the root to all neighbors' vertex models using only unused vertices. In order to solve this problem, we use a weighted shortest paths search. Let C be a constant, the vertex weight defined as:

$$weight(r) = C^{|\{i:r \in \phi(u_i)\}|}$$

We temporarily allow multiple vertex models of the guest graph H to share the same vertex from host graph G, and assigned a big penalty which grows exponentially with the number of vertices of H represented there. The best situation is a vertex model using only unused vertices from host G. Finally, we calculate the sum of the weights of all vertices used in that path and still choose the one with the smallest sum. By using the weighted shortest path search idea, we can make sure the new vertex model use overlapped vertices as few as possible.

4.2.1 Critical choices in implementation

There are some critical choices in implementing this algorithm. To overcome the weighted vertices shortest path search issue, we obtained a modified Dijkstra's algorithm [1] that works with weighted vertices to compute the weighted shortest path. The basic Dijkstra's algorithm works with weighted edges, however, we can easily convert vertex weights into edge weights by moving each arc's weight to its head. Also, when calculating the distance from vertex r to a vertex model $\phi(v)$, we add an extra vertex v' that is adjacent to every vertex in $\phi(v)$. By only calculating the shortest path to v', we do not have to find the shortest path to each vertex in the model. In this way, we can avoid a large computation time.

4.3 Approach with improvements

The most time-consuming part of this heuristic algorithm is computing the shortest path when we are trying to find the root for new vertex model. Since the weights of vertices change in every iteration, we have to recompute the shortest path each time.

4.3.1 Higher degree first with random root selection

Instead of doing initial randomization to get a vertex order for the guest graph, we could sort all the vertices in decreasing order by its degree. In such a way the vertex with a higher degree will find its corresponding vertex model first, resulting vertices have a lower degree would do less work for searching shortest path. Also, in the basic heuristic algorithm we always choose the root for vertex model with minimal cost; this requires a large calculation because we need to compute the length of the shortest path to every vertex in the host graph, from every vertex model. To overcome this issue, we may choose the root randomly, this is especially useful for a "small guest within a large host" type embedding since we do not want waste too much time on searching through all vertices in the host graph. Pseudo-code expresses this procedure.

HeuristicMinorEmbedding1(G,H)
Input: guest graph H with vertices $\{u_1, ..., u_n\}$, host graph G.
Output: vertex models $\phi(u_1), ..., \phi(u_n)$ of an H-minor in host, or "failure".

sort degree in descending order: $u_1, ..., u_n$
set $round = 1$
set $C = diameter(G)$
for $i \in \{1, ..., n\}$
    set $\phi(u_i) = \{\}$
while $max_{r \in V(G)} |\{i : r \in \phi(u_i)\}|$ or $\sum_i |\phi(u_i)|$ is improving, or $round \leq$ ROUNDS
    for $i \in \{1, ..., n\}$ do
        for $r \in V(G)$
            set $weight(r) = C^{|\{i : r \in \phi(u_i)\}|}$
        choose root r^* randomly from unused neighbor
        $\phi(u_i) = r^* \cup \{$paths from r^* to each neighboring vertex model $\phi(u_j)\}$
    set $round = round + 1$
if $|\{i : r \in \phi(u_i)\}| \leq 1$ for all $r \in V(G)$
    return $\phi(u_1), ..., \phi(u_n)$
else
    return "failure"

We terminate the algorithm if there is no improvement in embeddings after a certain number of iterations, the user could set up the minimum number of rounds (ROUNDS) as needed.

4.3.2 First level search

The random root selection idea helps avoid getting stuck in local optima in some case, but it will be exhausted when the input guest gets larger. In this section, we describe a modification to the random root algorithm which only searches the first level of neighbors' vertex model when finding the new root.

Instead of computing the cost of the shortest path to every vertex of G, from every vertex model. We calculate the distance for a small set of vertices which have a higher probability of being a "good root", we call this set the *rootCandidates*. Suppose we are looking for the vertex model $\phi(v)$, and v is adjacent to $u_1, ..., u_k$ which already have vertex models $\phi(u_1), ..., \phi(u_k)$. The *rootCandidates* selects vertices which are adjacent to each of these neighbor vertex model as our set of candidates, then remove the vertices have already used by any other vertex model to ensure all candidates are unused in G. After the initial selection, we still use the weighted shortest path algorithm to get the vertex with minimum cost as our new root r^*.

Computing the shortest path distances between vertex model and all vertices of G takes a large computational time. In first level search, by searching a small set

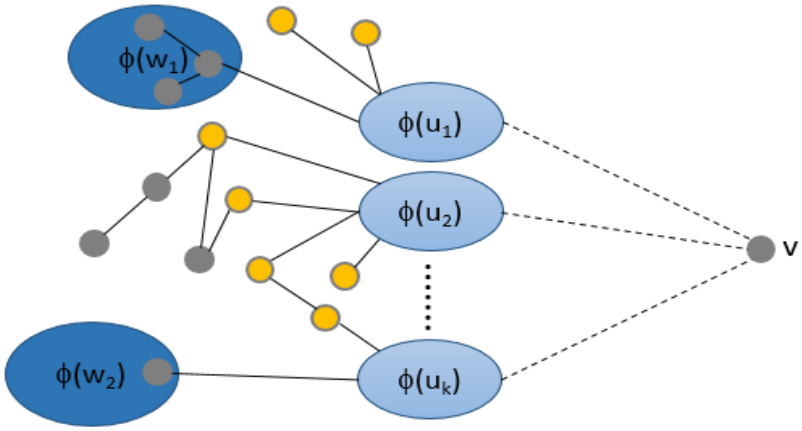

Figure 5: An illustration of using first level search idea to find new root for $\phi(v)$. We only consider the vertices adjacent to neighbors' vertex model as root candidates (colored yellow), then use weighted shortest path search to find the new root r^*.

of vertices we still could find a relative good embedding. Also, this first level search approach will find a better embedding to the random root selection. We give more details in next chapter. Pseudo-code for first level search approach is expressed next.

HeuristicMinorEmbedding2(G,H)
Input: guest graph H with vertices $\{u_1, ..., u_n\}$, host graph G.
Output: vertex models $\phi(u_1), ..., \phi(u_n)$ of an H-minor in host, or "failure".

randomize the vertex order: $u_1, ..., u_n$
set $round = 1$
set $C = diameter(G)$
for $i \in \{1, ..., n\}$
    set $\phi(u_i) = \{\}$
while $max_{r \in V(G)} |\{i : r \in \phi(u_i)\}|$ or $\sum_i |\phi(u_i)|$ is improving, or $round \leq$ ROUNDS
    for $i \in \{1, ..., n\}$ do
        for $r \in V(G)$
            set $weight(r) = C^{|\{i:r \in \phi(u_i)\}|}$
        $\phi(u_i) =$ findMinimalVertexModel(G,$weight$,{u_i: neighbors' vertex model})
    set $round = round + 1$
if $|\{i : r \in \phi(u_i)\}| \leq 1$ for all $r \in V(G)$
    return $\phi(u_1), ..., \phi(u_n)$
else
    return "failure"

FindMinimalVertexModel(G,$weight$,{$\phi(u_j)$})

Input: host graph G with vertex weights, neighboring vertex models $\{\phi(u_j)\}$.
Output: vertex model $\phi(v)$ in G such that there is an edge between $\phi(v)$ and each $\phi(u_j)$.

if all $\phi(u_j)$ are empty
    return random $\{r^*\}$
set $rootCandidates = \cup\{$all unused vertices from first level$\}$
for all $r \in rootCandidates$ and all j do
    if $\phi(u_j)$ is empty
        set $cost(r, j) = 0$
    else if $r \in \phi(u_j)$
        set $cost(r, j) = weight(r)$
    else
        set $cost(r, j) =$ weighted shortest path distance$(g, \phi(u_j))$ excluding $weight(\phi(u_j))$
set $r^* = \text{argmin}_g \sum_j cost(r, j)$
return $\{r^*\} \cup \{$paths from r* to each neighboring vertex model $\phi(u_j)\}$

4.3.3 Optimization on given embeddings

Note that not all minor embeddings result in the same hardware performance [5, 8]. Finding a better embedding leads directly to the better use of D-Wave's quantum annealing in solving quadratic pseudo-Boolean optimization problems. In general, for a minor embedding we would like to minimize:

- maximum number physical qubits representing any logical qubit (vertex model)
- the total number of physical qubits used in host G

Once the embedding has been found, we have to go through all the vertex model to check whether the improvement can be made. One method is depicted in Figure 6.

5 Results and Discussion

The heuristic search algorithms described in the previous section have three different versions:

1. The basic heuristic search through all vertices in the host graph.
2. Higher degree first search with random root selection search.
3. First level root selection search.

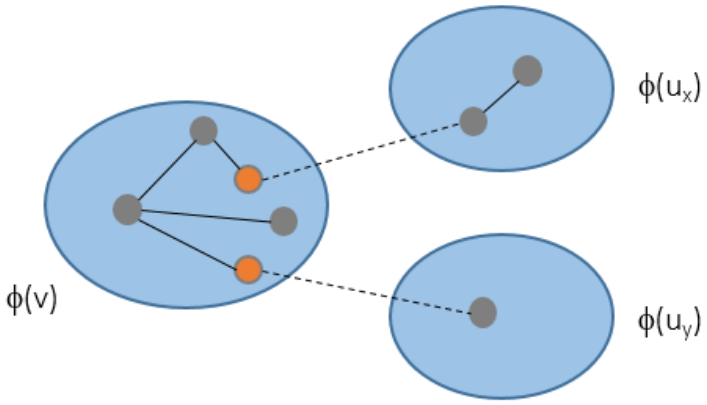

Figure 6: An illustration of minimizing vertex model size. If any vertex appears in only a single shortest path, move it to the model which has a fewer number of vertices. In this case we move the orange vertices to $\phi(u_x)$ and $\phi(u_y)$.

To show the performance of these three algorithms, we considered three sets of guest graphs: complete graphs, random cubic graphs, and a set of famous graphs obtained from [29, 7]. Also we picked three Chimera graphs as host graph with different size ($M \times N \times L$ notation which discussed in Section 2.2.3): $8 \times 8 \times 4$ (D-Wave Two), $12 \times 12 \times 4$ (D-Wave 2X) and $12 \times 14 \times 6$ respectively. For each combination, we run each algorithm a fixed number of times to get the average running time and embedding success probability, and record the time even when the minor embedding was not found. The running time is in seconds using a single core of an Intel i5 3.4GHz processor.

5.1 Embedding into an $8 \times 8 \times 4$ Chimera graph

We started with using some relative smaller graph as the host graph. In this section, we will present result for embedding into D-Wave Two architecture which is an $8 \times 8 \times 4$ Chimera graph of 512 vertices. The number of physical couplers (edges) connecting qubits (vertices) in the hardware is 1472. This means that if a guest graph has more than 512 vertices or 1472 edges, then there is no chance of finding an embedding. In particular, we run the basic heuristic search and first level search algorithm 10 times, the random selection algorithm 100 times, set a time out after a maximum of 1000 seconds. When the guest graph is much smaller than then the host graph, an embedding is often found within seconds, but for borderline cases, it can take minutes before finding an embedding.

5.1.1 Running time and success probability

Tables 1 and 2 show the running time and success probability for the set of famous graph from [29]. Some basic properties of these graphs are shown in Tables 3 and 4. Figure 7 shows how the expected time grows and success probability drops as the number of vertices increases respectively in complete graphs and random cubic graphs.

It is clear that for the same amount of running time, the basic heuristic search method has a much lower chance of finding an embedding than other two methods. The largest part of the algorithm's running time is computing the shortest path distances between vertex model and other vertices of G. This is obviously since for every iteration, the basic heuristic search needs to compute the length of the shortest path to every vertex of host G, from every vertex model. This is reckless. The random selection method avoids getting stuck in root selection step, resulting in a faster algorithm for fewer order graphs. However, we have to note that as the number of vertices increasing in the guest graph, the running time for random selection method increased dramatically at a certain point, and our first level search idea has best running time for borderline size guest graphs. This could be explained by the fact that the random selection method does not perform any pre-calculation before selecting the root, this is acceptable as input guest is small; when guest graph getting large the embedding requires a "good root" for each iteration, otherwise we would easy to have multiple vertex models share the same node, this overlap is not allowed in our embedding. To overcome this problem, the random selection method has to keep randomizing until finding the embedding, hence cost much time. As a result, it seems like using the first level search way to choose root from a small set of candidates may be performing the best, we use a small amount of time to select the "good root" from neighbor level candidates, which has a significant increase in the chance of finding embeddings.

Similarly, the embedding success probability for all three algorithms start off at 1, and at a certain number of graph order quickly drop off towards 0. From the Figure 7 can be seen that random selection method drops off earlier than other two methods, this happens when the embedding is trying to invoke more vertices and the random root may lead to an overlap. In terms of the success rates, there is not a significant difference between the basic heuristic and first level search idea, but the search work we did in first level search is much less than the basic heuristic algorithm, since we only compute the distance for a small set of root candidates. This result is what we expected.

Thus, the first level search approach would be the most practical algorithm of these three, especially in borderline cases. However, the random selection approach may still be useful for some cases with lower degree guest graph.

Table 1: Running time for embedding into an $8 \times 8 \times 4$ Chimera graph.

	Basic Heuristic		First Level Search		Random Selection	
Graph Name	Average Runtime	Success Probability	Average Runtime	Success Probability	Average Runtime	Success Probability
Balaban10Cage	353s	1	7.2s	1	0.18s	0.11
BiggsSmith	519s	0.4	9.9s	0.3	0.41s	0
BlanusaSnark1	60s	1	0.9s	1	0.06s	1
BlanusaSnark2	42s	1	1.6s	1	0.08s	1
Brinkmann	116s	0.9	3.2s	1	0.2s	0.98
BuckyBall	256	0.9	8.9s	0.9	0.3s	0.08
C20	34s	1	0.4s	1	0.04s	1
C30	68s	1	1.3s	1	0.04s	1
C40	91s	1	2.6s	1	0.05s	1
C50	140s	1	2.8s	1	0.07s	0.97
C60	168s	1	3.1s	1	0.1s	0.98
C70	209s	1	4.2s	1	0.16s	0.99
C80	228s	1	4.1s	1	0.17s	1
C90	299s	1	4s	1	0.19s	1
Coxeter	73s	1	3s	1	0.17s	1
Desargues	67s	1	1.4s	1	0.06s	1
DoubleStarSnark	102s	1	7s	1	0.19s	0.99
Dyck	109s	1	4.3s	1	0.12	0.97
Ellingham54	146s	0.9	6s	0.8	0.29s	0.28
Ellingham78	480s	0.6	12s	0.7	0.33s	0.02
FlowerSnark	82s	1	1.2s	1	0.06s	1
Folkman	69s	1	1.3s	1	0.06s	1
Foster	414s	0.2	8s	0.7	0.33s	0
Franklin	30s	1	0.4s	1	0.03s	1
Gray	266s	1	6.8s	1	0.14s	0.11
Grid6x6	167s	1	5.8s	1	0.1s	0.19
Grid6x7	191s	1	7.9s	1	0.15s	0.15
Grid6x8	189s	1	5.8s	1	0.3s	0.22
Grid6x9	312s	0.9	9.9s	0.8	0.02s	0
Grid7x7	221s	1	5.8s	0.9	0.14s	0.02
Grid7x8	224s	1	6.1s	1	0.22s	0
Grid7x9	370s	0.8	11.2s	0.8	0.23s	0
Grid8x8	399s	0.9	10.9s	1	0.24s	0
Harries	301s	1	8.1s	1	0.22s	0
HarriesWong	486s	0.9	8.2s	1	0.21s	0
HoffmanSingleton	333s	0	16s	0	0.35s	0
Horton	512s	0.7	17s	0.5	0.31s	0
K5x7	39s	1	1.7s	1	0.06s	1

Table 2: Running time for embedding into an $8 \times 8 \times 4$ Chimera graph.

	Basic Heuristic		First Level Search		Random Selection	
Graph Name	Average Runtime	Success Probability	Average Runtime	Success Probability	Average Runtime	Success Probability
K5x8	48s	1	1.7s	1	0.09s	1
K5x9	62s	1	2.2s	1	0.12s	1
K6x7	73s	1	2.2s	1	0.08s	1
K6x8	72s	1	1.9s	1	0.26s	1
K6x9	91s	1	2s	1	0.24s	1
K7x7	61s	1	1.3s	1	0.11s	1
K7x8	72s	1	1.9s	1	0.16s	1
K8x8	101s	1	3.3s	1	0.17s	1
K8x9	101s	1	4s	1	0.15s	1
K9x9	109s	1	6.8s	0.9	0.15s	1
K10x10	169s	0.7	7.7s	0.8	0.19s	0.99
K11x11	161s	0.9	4.9s	1	0.26s	1
K12x12	199s	0.9	9.6s	0.8	0.47s	0.99
Kittell	206s	1	4.9s	0.9	0.12s	1
Ljubljana	333s	0	7.9s	0	0.35s	0
Markstroem	79s	1	2s	1	0.14s	0.18
McGee	76s	1	3.4s	1	0.20s	1
Meredith	369s	0.3	8.8s	0.4	0.32s	0
MoebiusKantor	39s	1	0.6s	1	0.05s	1
MoserSpindle	6s	1	0.3s	1	0.03s	1
Nauru	101s	1	2s	1	0.06s	1
Q5	279s	0.7	11.1s	0.8	0.21s	0.08
Q6	587s	0.3	20.1s	0.2	0.52s	0
S10	15s	1	1s	1	0.06s	1
S20	21s	1	2.2s	1	0.07s	1
S30	36s	1	5.4s	1	0.09s	1
S40	49s	1	12.8s	0.9	0.11s	0.49
S50	61s	1	14s	1	0.23s	0.94
S60	72s	1	7.1s	1	0.23s	0.91
S70	117s	1	14s	1	0.39s	0.87
S80	144s	1	21s	0.8	0.42s	0.86
S90	168s	0.7	41s	0.9	0.39s	0.92
Schlaefli	617s	0.7	32s	0.5	0.4s	0.28
Sylvester	258s	0.9	7.5s	1	0.19s	0.17
SzekeresSnark	172s	1	5.4s	1	0.17s	0.09
TutteCoxeter	77s	1	3.2s	1	0.23s	1
Wells	141s	0.8	8.1s	0.6	0.16s	0
WienerAraya	248s	0.9	4.8s	1	0.31s	0.88
K7 $\cup$ Q3	62s	1	1.6s	1	0.11s	1

Table 3: Properties of guest graphs—Part 1.

Graph Name	Order	Size	Min degree	Max degree	Diameter	Girth
Balaban10Cage	70	105	3	3	6	10
BiggsSmith	102	153	3	3	7	9
BlanusaSnark1	18	27	3	3	4	5
BlanusaSnark2	18	27	3	3	4	5
Brinkmann	21	42	4	4	3	5
BuckyBall	60	90	3	3	9	5
C20	20	20	2	2	10	20
C30	30	30	2	2	15	30
C40	40	40	2	2	20	40
C50	50	50	2	2	25	50
C60	60	60	2	2	30	60
C70	70	70	2	2	35	70
C80	80	80	2	2	40	80
C90	90	90	2	2	45	90
Coxeter	28	42	3	3	4	7
Desargues	20	30	3	3	5	6
DoubleStarSnark	30	45	3	3	4	6
Dyck	32	48	3	3	5	6
Ellingham54	54	81	3	3	10	6
Ellingham78	78	117	3	3	13	6
FlowerSnark	20	30	3	3	4	6
Folkman	20	40	4	4	4	4
Foster	90	135	3	3	8	10
Franklin	12	18	3	3	3	4
Gray	54	81	3	3	6	8
Grid6x6	36	60	2	4	10	4
Grid6x7	42	71	2	4	11	4
Grid6x8	48	82	2	4	12	4
Grid6x9	54	93	2	4	13	4
Grid7x7	49	84	2	4	12	4
Grid7x8	56	97	2	4	13	4
Grid7x9	63	110	2	4	14	4
Grid8x8	64	112	2	4	10	4
Harries	70	105	3	3	6	10
HarriesWong	70	105	3	3	6	10
HoffmanSingleton	50	175	7	7	2	5
Horton	96	144	3	3	10	6
K5x7	12	35	5	7	2	4
K7 $\cup$ Q3	15	33	3	7	∞	3

Table 4: Properties of guest graphs—Part 2.

Graph Name	Order	Size	Min degree	Max degree	Diameter	Girth
K5x8	13	40	5	8	2	4
K5x9	14	45	5	9	2	4
K6x7	13	42	6	7	2	4
K6x8	14	48	6	8	2	4
K6x9	15	54	6	9	2	4
K7x7	14	49	7	7	2	4
K7x8	15	56	7	8	2	4
K8x8	16	64	5	8	2	4
K8x9	17	72	8	9	2	4
K9x9	18	81	9	9	2	4
K10x10	20	100	10	10	2	4
K11x11	22	121	11	11	2	4
K12x12	24	144	12	12	2	4
Kittell	23	63	5	7	4	3
Ljubljana	112	168	3	3	8	10
Markstroem	24	36	3	3	6	3
McGee	24	36	3	3	4	7
Meredith	70	140	4	4	8	4
MoebiusKantor	16	24	3	3	4	6
MoserSpindle	7	11	3	4	2	3
Nauru	24	36	3	3	4	5
Q5	32	80	5	5	5	4
Q6	64	192	6	6	6	4
S10	11	10	1	10	2	∞
S20	21	20	1	20	2	∞
S30	31	30	1	30	2	∞
S40	41	40	1	40	2	∞
S50	51	50	1	50	2	∞
S60	61	60	1	60	2	∞
S70	71	70	1	70	2	∞
S80	81	80	1	80	2	∞
S90	91	90	1	90	2	∞
Schlaefli	27	216	16	16	2	3
Sylvester	36	90	5	5	3	5
SzekeresSnark	50	75	3	3	7	5
TutteCoxeter	30	45	3	3	4	8
Wells	32	80	5	5	4	5
WienerAraya	42	67	3	4	7	5
n-cubic graph	n	$\frac{3n}{2}$	3	3	-	-
Complete K_n	n	$\frac{n(n-1)}{2}$	$n-1$	$n-1$	1	3

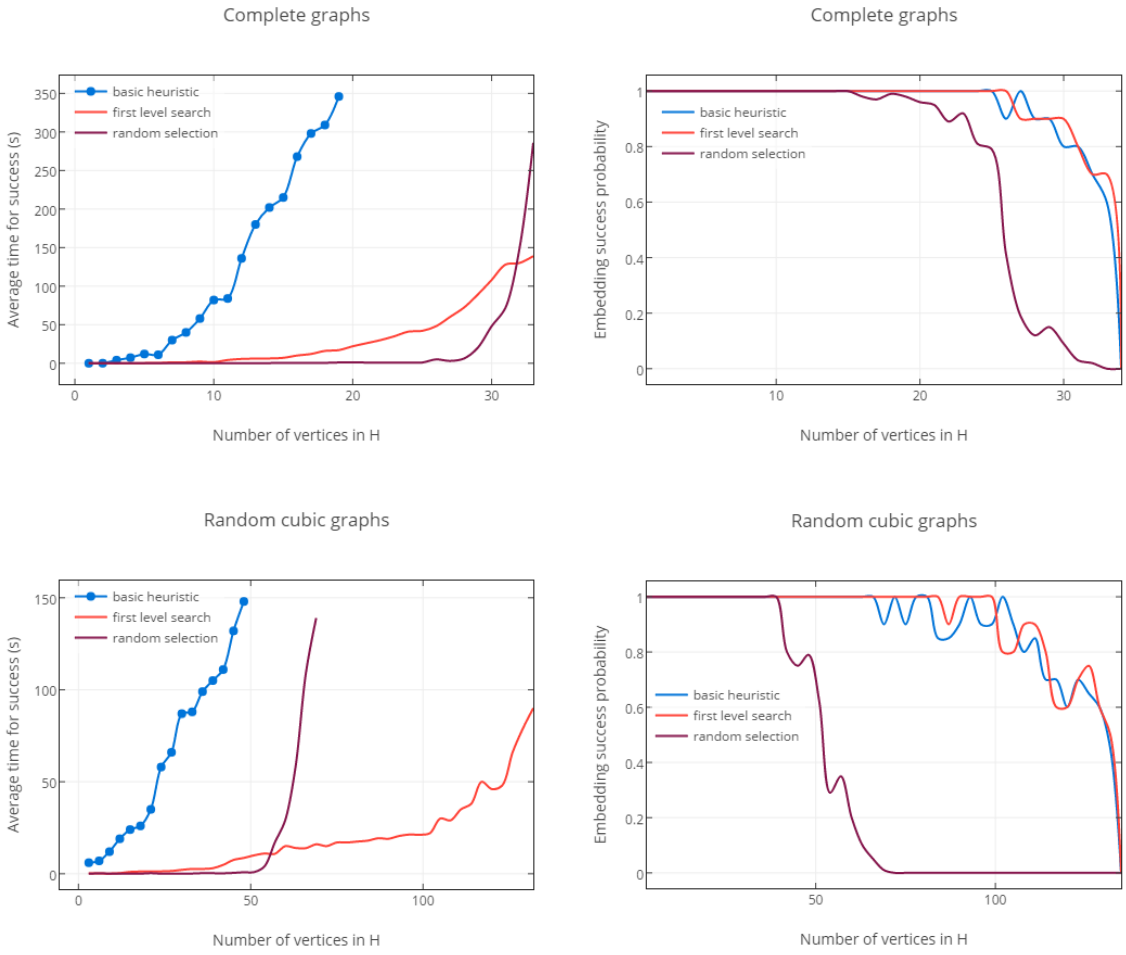

Figure 7: Comparing the performance of three heuristic minor embedding algorithms in finding graph minors into an $8 \times 8 \times 4$ Chimera graph G. Line plots showing how of time grows with the number of vertices increasing in complete graphs and random cubic graphs, as well as the proportion of successful embeddings of these three algorithms. Success probability starts at 1, and at a certain number of vertices, quickly drop off towards 0. It can be seen that success probability of the random selection method drops off significantly earlier than other two methods.

5.1.2 Embedding quality

It has been suggested that the high maximum chain length (the biggest vertex model size) and a large total number of physical qubits used (total number of vertices used in all vertex models) results in worse performance by the hardware [14]. In this section, we show the correlation between the order of guest graph and the quality of its corresponding embedding.

Based on the information from Figure 8 and Tables 5 and 6, we can see that the physical qubits used increase much more quickly than the order of guest graph. As expected, both basic heuristic search and first level search algorithms resulting the similar correlation between guest graph order and total physical qubits, but the rate of increase is more in the random selection method. Due to the same reason we discussed in the previous section, the random selection method does not perform any pre-computation when choosing the root. It randomly picks a vertex and searches for a local solution. In most of the cases, the new vertex model would involve more vertices since we need to find a path from the root to each neighbor vertex model, this is why total physical qubits used increase dramatically.

It is not hard to see that the maximum chain length is proportional to total physical qubits used for both complete graphs and cubic graphs. In particular, while as the number of vertices increase in the guest graph, the embedding quality for random selection algorithm drops faster in sparse graphs. Interestingly the first level search method produces a slightly better embedding quality than the basic heuristic way on harder problems, even the basic heuristic suffer from more runtime doing the search.

In terms of the both running time and embedding quality, it suggests that the first level search based algorithm may be performing the best in borderline cases and harder problems. But we still suggest that the random selection strategy may give some useful results on particular problems.

5.2 More results on harder problems

We give the performance of first level search algorithm on harder problems, as illustrated in Figures 9 and 10.

Figure 9 shows the embedding performance on D-Wave 2X hardware graph, which is the latest D-Wave architecture released on August 20, 2015. Figure 10 demonstrates our first level search algorithm is still practical for a host graph with two thousand vertices and a guest graph with three hundred vertices.

Table 5: Embedding quality for embedding into an $8 \times 8 \times 4$ Chimera graph.

	Basic Heuristic		First Level Search		Random Selection	
Graph Name	Qubits Used	Maximum Chain	Qubits Used	Maximum Chain	Qubits Used	Maximum Chain
Balaban10Cage	198	16	198	14	288	18
BiggsSmith	303	12	294	12	-	-
BlanusaSnark1	30	4	38	6	68	10
BlanusaSnark2	40	5	49	6	75	10
Brinkmann	55	6	62	6	154	13
BuckyBall	179	14	125	9	208	16
C20	26	3	32	3	36	12
C30	46	7	48	8	60	10
C40	56	6	50	4	65	11
C50	66	4	82	6	83	14
C60	74	4	72	4	80	8
C70	82	3	88	6	116	12
C80	118	7	102	5	133	11
C90	116	6	120	8	127	11
Coxeter	55	7	64	6	165	10
Desargues	26	2	33	3	88	8
DoubleStarSnark	70	7	59	6	158	15
Dyck	157	7	62	5	154	14
Ellingham54	116	9	115	7	258	14
Ellingham78	172	7	177	7	408	32
FlowerSnark	33	5	33	5	90	11
Folkman	44	4	45	4	100	10
Foster	288	14	267	15	-	-
Franklin	18	5	22	5	35	7
Gray	147	13	128	8	288	22
Grid6x6	92	6	84	7	186	12
Grid6x7	85	6	87	6	248	16
Grid6x8	130	10	134	8	272	16
Grid6x9	125	7	120	6	-	-
Grid7x7	131	9	117	7	288	14
Grid7x8	160	13	160	12	-	-
Grid7x9	168	10	137	6	-	-
Grid8x8	170	8	135	6	-	-
Harries	234	11	203	11	-	-
HarriesWong	183	8	193	10	-	-
HoffmanSingleton	-	-	-	-	-	-
Horton	252	15	247	11	-	-
K5x7	24	2	24	3	70	12

Table 6: Embedding quality for embedding into an $8 \times 8 \times 4$ Chimera graph.

	Basic Heuristic		First Level Search		Random Selection	
Graph Name	Qubits Used	Maximum Chain	Qubits Used	Maximum Chain	Qubits Used	Maximum Chain
K5x8	26	7	26	8	81	21
K5x9	33	6	33	4	85	25
K6x7	26	2	26	2	95	14
K6x8	36	5	28	2	73	13
K6x9	35	5	36	4	102	16
K7x7	28	2	28	2	100	16
K7x8	30	2	30	2	91	14
K8x8	32	2	65	5	129	20
K8x9	42	3	42	5	144	22
K9x9	54	10	54	11	163	15
K10x10	70	12	60	3	198	26
K11x11	66	3	66	3	256	23
K12x12	72	3	72	3	240	23
Kittell	69	5	92	10	146	12
Ljubljana	-	-	-	-	-	-
Markstroem	66	10	61	6	90	8
McGee	53	7	40	4	132	10
Meredith	205	15	188	13	-	-
MoebiusKantor	32	3	31	4	48	8
MoserSpindle	13	3	12	3	23	5
Nauru	38	4	40	4	141	10
Q5	119	10	113	8	365	21
Q6	400	16	356	15	-	-
S10	13	3	14	4	14	4
S20	25	5	35	6	27	7
S30	38	8	41	10	43	13
S40	54	14	59	16	59	19
S50	64	14	69	14	74	20
S60	80	20	77	17	86	24
S70	89	19	91	20	104	34
S80	103	23	109	26	121	41
S90	138	30	130	32	156	67
Schlaefli	243	12	258	14	374	25
Sylvester	174	10	182	12	278	18
SzekeresSnark	108	8	96	6	353	21
TutteCoxeter	57	4	64	4	161	14
Wells	151	13	149	11	-	-
WienerAraya	116	7	103	8	272	15
K7 $\cup$ Q3	33	4	30	3	55	6

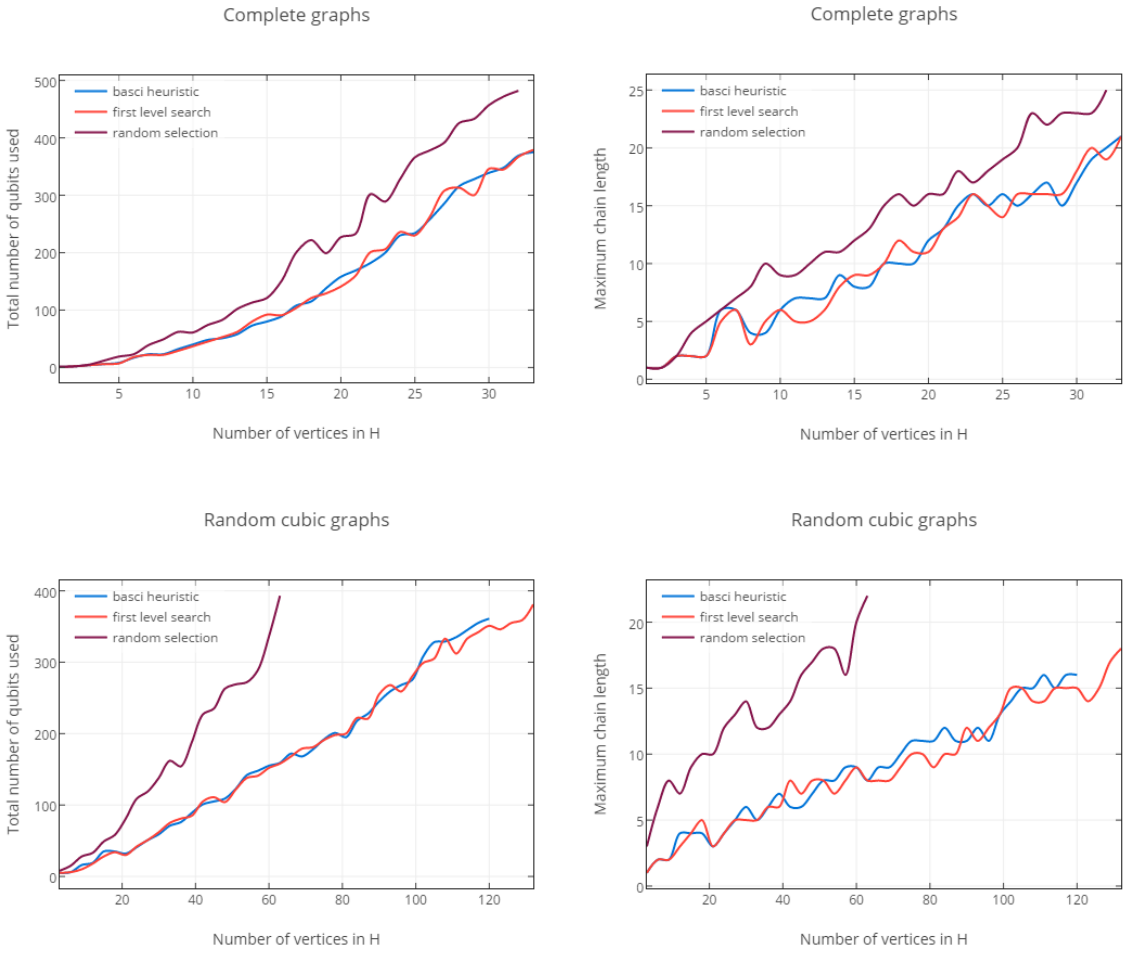

Figure 8: Comparing the embedding quality of three heuristic minor embedding algorithms in finding graph minors into an $8 \times 8 \times 4$ Chimera graph G. Line plots of maximum chain length and the total number of qubits used as increases size in complete graphs and random cubic graphs. It is clear that for the same number of vertices in H, the basic heuristic search and first level search algorithms have both lower maximum chain length and the total number of qubits used, as would be expected.

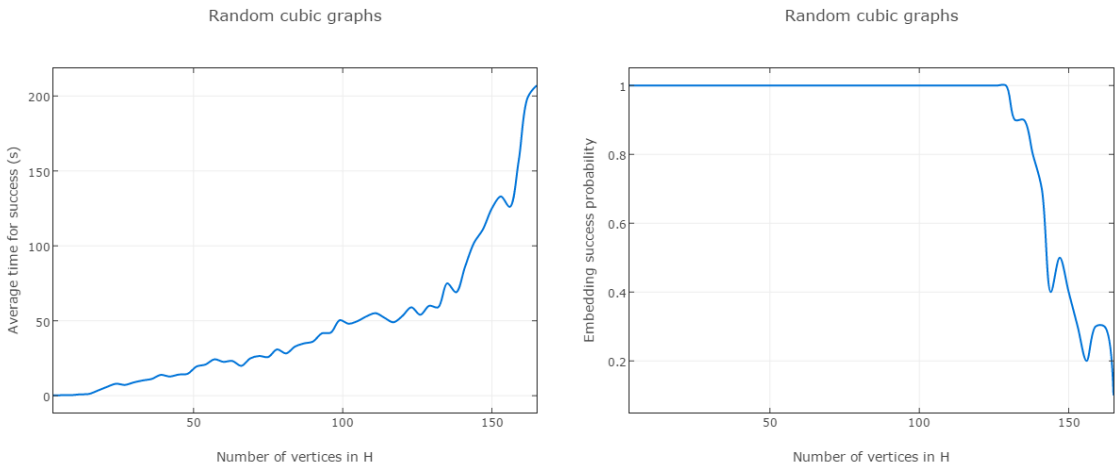

Figure 9: The performance of embedding random cubic graphs into a $12 \times 12 \times 4$ Chimera graph (D-Wave 2X) with 1152 vertices. Success probability quickly drops off towards to 0 when the guest graph has more than 150 vertices.

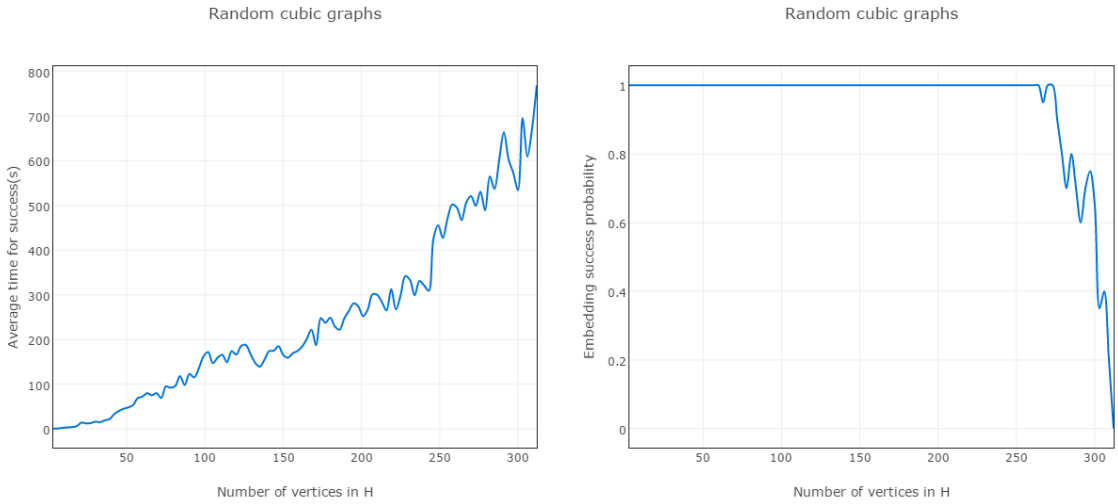

Figure 10: Line graph showing the running time and probability of embedding random cubic graphs into a $12 \times 14 \times 6$ Chimera graph with 2016 vertices. The algorithm could embed cubic graph which has 300 vertices with high probability.

6 Conclusions

6.1 Summary

The D-Wave hardware Chimera graph is designed to have large treewidth and large automorphism group [4], which means exact algorithms are of no practical use at the current scales. In this paper, we reviewed the current algorithms on the minor embedding problem. We also presented two new heuristic algorithms for finding graph minors in sparse host graph and guest graph with hundreds of vertices.

Using a heuristic search method to find graph minors was originally inspired by Cai [5]. Based on this idea we presented the random selection approach and the first level search approach. The random selection method always randomly picks root from its neighbor level, in this way we avoided large computation time on the most time-consuming part, but due to the bad embedding quality, this algorithm is not practical for borderline cases. The first level search algorithm has been implemented and tested, based on large experiment results, it has proven to be effective in finding minors with hundreds of vertices.

Finding a better algorithm for minor embedding problem leads directly to the better use of D-Wave's quantum annealing in solving quadratic pseudo-Boolean optimization problems. The host graph architecture produced by the D-Wave company is slightly different in each processor, as certain qubits of insufficient quality are disabled, the future versions of the hardware may have totally different architecture. Our first level search algorithm requires both host graph and guest graph as part of the input, so we believe this approach will be useful in providing a practical way of finding minor embeddings for D-Wave computer architecture.

6.2 Limitations

We now describe some of the limitations when using our heuristic search algorithms and the D-Wave computer to solve problems:

- The heuristic search algorithm is an entirely probabilistic algorithm and provides no guarantees of finding an embedding.
- There is no strong evidence of quantum computer can get a speed-up since the work of embedding imposes a heavy penalty.
- Although quantum computers may be faster than classical computers for some problem types, quantum computers are unable to compute anything which cannot be computed classically.

6.3 Future research

Based on our results here, we also list some potential future research:

- The heuristic search algorithm presented here takes a large running time to calculate the shortest path distances, it can be improved significantly by looking for a "good root" without too much computation. How to find a good root of vertex models is a topic for future research.
- Since we know the host Chimera architecture has a bounded degree, more focus on exploiting this property should be continued.
- A better embedding result directly to a better performance of D-Wave's quantum annealing in solving QUBO problems. Investigating the correlation between embedding quality and D-Wave computer performance needs to be done in the future.

References

[1] Michael Barbehenn. A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices. *IEEE transactions on computers*, (2):263, 1998.

[2] Max Born and Vladimir Fock. Beweis des adiabatensatzes. *Zeitschrift für Physik*, 51(3-4):165–180, 1928.

[3] Endre Boros and Peter L. Hammer. Pseudo-Boolean optimization. *Discrete applied mathematics*, 123(1):155–225, 2002.

[4] Paul I. Bunyk, Emile M. Hoskinson, Mark W. Johnson, Elena Tolkacheva, Fabio Altomare, Andrew J. Berkley, Roy Harris, Jeremy P. Hilton, Trevor Lanting, Anthony J. Przybysz, et al. Architectural considerations in the design of a superconducting quantum annealing processor. *Applied Superconductivity, IEEE Transactions on*, 24(4):1–10, 2014.

[5] Jun Cai, William G. Macready, and Aidan Roy. A practical heuristic for finding graph minors. *arXiv preprint arXiv:1406.2741*, 2014.

[6] Cristian S. Calude, Elena Calude, and Michael J. Dinneen. Guest column: Adiabatic quantum computing challenges. *ACM SIGACT News*, 46(1):40–61, 2015.

[7] Cristian S. Calude and Michael J. Dinneen. Solving the broadcast time problem using a d-wave quantum computer. Technical Report CDMTCS 473, Center for Discrete Mathematics and Theoretical Computer Science, The University of Auckland, New Zealand, 2014.

[8] Vicky Choi. Minor-embedding in adiabatic quantum computation: I. the parameter setting problem. *Quantum Information Processing*, 7(5):193–209, 2008.

[9] Niraj Datt. Practical graph containment algorithms. *University of Auckland, Honors degree thesis*, 2009.

[10] David Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 400, pages 97–117. The Royal Society, 1985.

[11] Michael J. Dinneen, Georgy Gimel'farb, and Mark C. Wilson. *Introduction to Algorithms, Data Structures and Formal Languages*. Pearson Education New Zealand, 2nd edition, 2009.

[12] Richard P. Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6):467–488, 1982.

[13] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[14] Richard J. Hughes and Colin P. Williams. Quantum computing: The final frontier? *Intelligent Systems and their Applications, IEEE*, 15(5):10–18, 2000.

[15] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse Ising model. *Physical Review E*, 58(5):5355, 1998.

[16] Donald L. Kreher and Douglas R. Stinson. *Combinatorial algorithms: generation, enumeration, and search*, volume 7. CRC press, 1998.

[17] Andrew Lucas. Ising formulations of many NP problems. *arXiv preprint arXiv:1302.5843*, 2013.

[18] Yuri I. Manin. Quantum computing and Shor's factoring algorithm. *arXiv:quant-ph/9903008v1*, 1999.

[19] Laurence M. McFarlane. Embedding bounded treewidth graphs into the D-Wave computer architecture. *University of Auckland, Honors degree thesis*, 2014.

[20] Kristen L. Pudenz, Tameem Albash, and Daniel A. Lidar. Error-corrected quantum annealing with hundreds of qubits. *Nature communications*, 5, 2014.

[21] Timothy Resnick. Sudoku at the intersection of classical and quantum computing. Technical Report CDMTCS 475, Center for Discrete Mathematics and Theoretical Computer Science, The University of Auckland, New Zealand, 2014.

[22] Neil Robertson and Paul D. Seymour. Graph minors. XIII. the disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995.

[23] Neil Robertson and Paul D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004.

[24] Robert R. Schaller. Moore's law: past, present and future. *Spectrum, IEEE*, 34(6):52–59, 1997.

[25] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. IEEE, 1994.

[26] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 2012.

[27] D-Wave Systems. Programming with QUBOs. 2013.

[28] D-Wave Systems. Meet D-Wave. *http://www.dwavesys.com/our-company/meet-d-wave*, 2016.

[29] Stein A. William. Sage mathematics software (version 6.2). *The Sage Development Team, http://www.sagemath.org*, 2014.

[30] Liu Xiong and Michael J. Dinneen. The feasibility and use of a minor containment algorithm. Technical Report 171, Department of Computer Science, The University of Auckland, New Zealand, 2000.