2023 IEEE International Conference on Communications; Selected Areas in Communications: Quantum Communications and Information Technology

ATOM: An Efficient Topology Adaptive Algorithm for Minor Embedding in Quantum Computing

Hoang M. Ngo
University of Florida
hoang.ngo@ufl.edu

Tamer Kahveci
University of Florida
tkahveci@ufl.edu

My T. Thai
University of Florida
mythai@cise.ufl.edu

***Abstract*—Quantum annealing (QA) has emerged as a powerful technique to solve optimization problems by taking advantages of quantum physics. In QA process, a bottleneck that may prevent QA to scale up is minor embedding step in which we embed optimization problems represented by a graph, called logical graph, to Quantum Processing Unit (QPU) topology of quantum computers, represented by another graph, call hardware graph. Existing methods for minor embedding require a significant amount of running time in a large-scale graph embedding. To overcome this problem, in this paper, we introduce a novel notion of adaptive topology which is an expandable subgraph of the hardware graph. From that, we develop a minor embedding algorithm, namely Adaptive TOpology eMbedding (ATOM). ATOM iteratively selects a node from the logical graph, and embeds it to the adaptive topology of the hardware graph. Our experimental results show that ATOM is able to provide a feasible embedding in much smaller running time than that of the state-of-the-art without compromising the quality of resulting embedding.** [1]

***Index Terms*—quantum annealing, adaptive hardware topology**

I. Introduction

Quantum computing has shown its supremacy over classical computers as it exponentially reduces the running time of certain computational tasks [1] in cybersecurity [2], machine learning [3], and optimization [4] among others. One of the most outstanding paradigms for quantum computing is Quantum Annealing (QA). It focuses solely on solving optimization problems by utilizing quantum fluctuation effect. QA scales to significantly larger number of qubits. This characteristic enables QA to solve large optimization tasks such as scheduling deep space network [4], designing logistic network [5], finding optimal traffic flow [6], and TDMA scheduling in wireless sensor networks [7], to name a few.

QA-inspired quantum computers solve an optimization problem in three steps. First, the quadratic unconstrained binary optimization (QUBO) representation of the given problem is encoded by a graph, called *logical graph*. Second, the resulting logical graph is embedded into Quantum Processing Unit (QPU) whose topology is represented by another graph, called *hardware graph* such that the logical graph can be obtained from the embedded subgraph of the hardware graph by edge contraction. This embedding process is called **minor embedding**. Finally, quantum computers run the quantum annealing process repeatedly on the embedded QPU to obtain the optimal solution for the given optimization problem. Along with the rapid development in the QPU size, QA-inspired quantum computers are able to solve large optimization problems. The fundamental challenge here is that solving minor embedding problem is non-trivial for the topology of the hardware graph or its induced sub-graphs may not be identical to that of the logical graph. Therefore, embedding process remains to be a bottleneck, especially, when the logical graph size is large. There is an urgent need for an effective minor embedding method which scales to large logical graphs.

Related Work. There are two main approaches to solve minor embedding, namely top-down and bottom-up. Top-down approach aims to find embeddings of complete graphs [8], [9], [10] in the hardware graph. Although the embedding of a complete graph can be the solution for any incomplete graph with smaller or same size, embedding for incomplete graphs, especially sparse graphs may require much fewer qubits than embedding a complete graph. Thus, top-down approach is not effective for embedding sparse logical graphs. To address this problem, Goodrich et al. proposed a post-process the resulting embedding to reduce the number of qubits required [11]. They introduced a new notion of biclique virtual hardware framework, which is constructed by compressing the hardware graph. They find a virtual embedding of the logical graph in the virtual hardware framework using the Odd Cycle Traversal (OCT)-based algorithm. OCT-based method is the-state-of-the-art embedding for general logical graphs. However, the post-process in OCT-based method is not efficient for large logical graphs due to its high complexity.

In contrast, bottom-up approach directly constructs solutions based on the topology of the logical and the hardware graphs. It computes minor embedding either by using Integer Programming (IP) which find an exact solution from predefined constraints [12], or by using a progressive heuristic method which embeds one node of the logical graph to the hardware graph at a time [13], [14]. Notably, Minorminer method [13] commercially developed by D-Wave Inc. scales well for sparse logical graphs. Intuitively, Minorminer quickly finds an infeasible embedding in the hardware graph, and then re-embeds overlapping nodes until it obtains a feasible one. The first disadvantage of Minorminer is that the complexity of their algorithm depends on the input hardware graph size, so the running time grows considerably for inappropriate input hardware graphs. In addition, the number of re-embedding may be large, which can contribute significantly in total

[1]This work is partially funded by NSF under Award 2111679 and 1908594.

running time.

Contributions. In this paper, we introduce a novel idea of adaptive topology for the hardware graph whose size grows along with the size of embedding as needed. We develop an efficient minor embedding method, called Adaptive TOpology eMbedding (ATOM). Our method follows the bottom-up approach in which we embed one node in the logical graph at a time to an adaptive hardware graph. Unlike existing approaches, ATOM does not require a fixed hardware graph size. This leads to a significant reduction in the running time, and overcomes the aforementioned limitations.

In our experiments, we compare our method with OCT-based [11], and Minorminer [13] which are two state-of-the-art embedding methods in each category as discussed in the related work. Our results demonstrate that ATOM requires significant less running time, while the hardware size and the number of qubits required for embedding are comparable to the best settings of hardware size for two other methods.

Organization. The rest of the paper is structured as follows. Section II introduces the definition of minor embedding and related notions. Our solution, ATOM, and its theoretical analysis are described in Section III. Section IV presents our experimental results. Finally, Section V concludes the paper.

II. Preliminaries

Here, we first present the fundamental terminology needed to understand how QA solves optimization problems. Next, we describe the minor embedding problem in QA.

Recall from Section I that QA consists of three steps. Below, we elaborate on this process. In the first step of QA, given an optimization problem with the set of binary variables $\mathbf{x} = \{x_0, x_1, ..., x_N\}$ and quadratic coefficients $Q_{i,j}$, we represent the problem with QUBO form, and express the optimal solution for the given problem corresponding to the state with lowest energy of final Hamiltonian as $\mathbf{x}^*$ such that:

$$\mathbf{x}^* = \arg\ \min_{\mathbf{x}\in\{0,1\}^N} \sum_{i\leq j} Q_{i,j} x_i x_j \qquad (1)$$

$$\text{for } Q_{i,j} \in \mathbb{R}$$

We encode the QUBO formulation using a graph called logical graph. In the logical graph, each node corresponds to a binary variable x_i. We draw an edge between the nodes corresponding to two variables x_i and x_j if the quadratic coefficient $Q_{i,j}$ is nonzero.

The hardware graph is a representation of the topology of QPU with nodes corresponding to qubits and edges corresponding to qubits' couplers. QA system of D-Wave consists of three QPU topologies, namely Chimera - the earliest topology, Pegasus - the latest topology, and Zephyr - next generation QPU topology. These topologies are all in form of a grid of identical sets of nodes called unit cells. In this paper, we consider the Chimera topology. The reason is that [15] embedding methods in Chimera topology may be translated to Pegasus topology without modification, because Chimera is a subgraph of Pegasus. In other words, the method we develop for Chimera model also works for Pegasus topology. The Chimera topology is in form of $n \times m$ grid of $K_{c,c}$ unit cells as $T(n, m, c)$.

QUBO models given optimization problems as a logical graph, and the topology of QPU used in QA as a hardware graph. In order to solve the final Hamiltonian which QA processes, we need to find an mapping from the logical graph to the hardware graph. Below we formally define the minor embedding problem:

Definition 1. *(Minor embedding). Given a logical graph $P = (V_P, E_P)$ and a hardware graph $H = (V_H, E_H, T)$, minor embedding problem seeks to a mapping $\phi : V_P \to \mathcal{P}(V_H)$ satisfying three conditions:*

1) **Chain connection**: We denote a subset of nodes in H that are mapped from a node u in P as chain $\phi(u)$. Any subgraph H' of H induced by a chain $\phi(u)$ from H is a connected component with $\forall u \in V_P$.
2) **Global connection**: For every edge $(u, v) \in E_P$, there exists at least one edge $(u', v') \in E_H$ such that $u' \in \phi(u)$, and $v' \in \phi(v)$.
3) **One-to-many**: Two chains $\phi(u)$ and $\phi(v)$ in the hardware graph do not have any common nodes with $\forall u, v \in V_P, u \neq v$.

Because hardware graphs based on QPU topologies are fixed-structured and incomplete graphs, finding minor embedding is a difficult problem.

III. Adaptive topology embedding (ATOM)

Here, we describe our algorithm, called Adaptice TOpology eMbedding (ATOM) for the minor embedding problem. We introduce the notion of adaptive topology of the hardware graph. We leverage the *Chimera* topology to expand a hardware graph $T(n, m, c)$ to a new hardware graph $T(n', m', c)$ with $n' \geq n$, and $m' \geq m$ such that properties of an embedding in $T(n, m, c)$ is still preserved in $T(n', m', c)$. With this finding, we define adaptive topology as a subgraph of the hardware graph which grows along with size of current embedding. To be consistent with the Definition 1, in which hardware graph H is fixed, in our representation below, we assume that the hardware graph H is large enough for every embedding.

A. ATOM Algorithm

Algorithm 1 outlines the ATOM algorithm for minor embedding. The algorithm takes the logical graph $P = (V_P, E_P)$ which encodes the QUBO formulation, and the hardware graph $H = (V_H, E_H, T(n, m, c))$ where $T(n, m, c)$ encodes its initial adaptive topology as input. It finds an embedding $\phi : V_P \to \mathcal{P}(V_H)$ which satisfies the three conditions of minor embedding provided in Definition 1. We call such embedding *feasible embedding*. Recall that a *chain* $\phi(v)$ with $v \in P$ indicates the set of node in the hardware graph H embedded by v. Let us denote a subset of nodes in P with S. We define $P[S]$ as the subgraph of P induced on S. To be simple, we also assume that the logical graph P is a connected component.

The main idea behind ATOM is that nodes in P are progressively embedded to H at a time. We denote the

process to embed one node to H as *turn* of embedding. We denote the embedded set after the ith turn with S_i, and the embedding after the ith turn with ϕ_i. ATOM has three major steps: Initialization, Node Embedding, and Topology Adapting. Initialization determines the value of the set S_0, the embedding ϕ_0, and other variables (lines 1 - 7). By finding a feasible embedding ψ of the k-node subgraph P' of P with the most edges in H, ATOM initializes S_0 as $V(P')$, and ϕ_0 as ψ. After initialization, at each turn, a new unembedded node is selected, and current embedding is updated by adding chains of the new node, and its neighbors in H through *Node Embedding* (lines 9 - 11). If the new node is successfully embedded to H, algorithm 1 updates current embedding with an additional embedding which includes new chains found in *Node Embedding* (lines 16 - 20). Otherwise, algorithm 1 expands current embedding, and current adaptive topology through *Topology Adapting*, and re-embeds the new node with expanded topology (lines 12 - 14). We explain the process of embedding a new node to H in section B, and then describe the process of expanding current embedding in subsection C below. *In the rest of this paper, we omit the proofs of all lemmas and theorems due to page limit.*

B. Node Embedding

Algorithm 2 outlines how an additional embedding is found from current embedding at each turn. The algorithm takes the logical graph $P = (V_P, E_P)$, the hardware graph $H = (V_H, E_H, T(n, m, c))$, the embedding at current turn ϕ, the embedded set at current turn S, and the new node $\bar{v}$ to be included in the embedding as input. The goal of the algorithm is to find a new embedding ϕ' such that combination of ϕ' and ϕ is a feasible embedding for $P[S \cup \{\bar{v}\}]$. Before going any further, we define the reverse mapping from a node in H to a node in P as ϕ^{-1}. If a node $u \in H$ is not embedded by any node $v \in P$ through the embedding ϕ, we call node u as a *free node* with $\phi^{-1}(u) = -1$. In addition, we define a path from $u \in V_H$ to the chain $\phi(v)$ with $v \in V_P$ with length of L_{uv} as $Z_{uv} = \{z_0, z_1, z_2, \ldots, z_{L_{uv}}\}$. If Z_{uv} satisfies the condition in which for $0 \leq i \leq L_{uv} - 1$, $\phi^{-1}(z_i) = -1$ and, $\phi^{-1}(z_{L_{uv}}) = v$, the path Z_{uv} is a *clean path*. Finally, we define the subgraph of H induced from adaptive topology $T(n, m, c)$ as H'.

Algorithm 2 splits the whole process into two major steps: (1) finding center for new chain of the new node $\bar{v}$ (lines 1 - 8), and (2) assigning free node to connect the new chain to the existing embedding (lines 9 - 14). For the first step, after determining the embedded neighbour set of $\bar{v}$ as $A = \{v | v \in S, (\bar{v}, v) \in E_P\}$, algorithm 2 generates all possible clean shortest paths Z_{uv} from a free node $u \in V_{H'}$ to chains $\phi(v)$ with $v \in A$ included in the set Z by Breath First Search algorithm (lines 3 - 4). If from existing clean paths in Z, no feasible center can be found for the new chain (line 6), algorithm 2 returns an embedding ϕ' with $\phi'(v) = \emptyset$ for $\forall v \in P$, called *empty embedding*. In contrast, the center of the chain for new node $\phi(\bar{v})$ is selected as the free node $\bar{u}$ to which sum of shortest clean paths from $\phi(v)$ with $v \in A$ is minimum

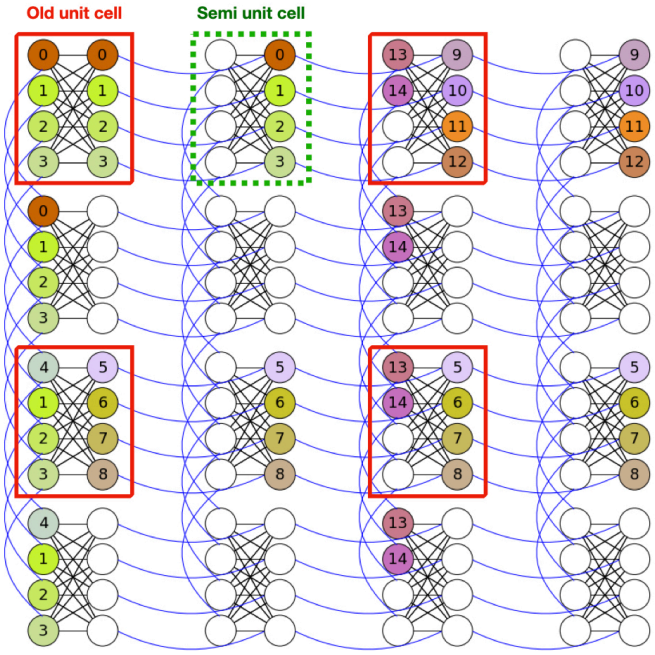


Figure 1: A hardware graph in form of $T(4, 4, 4)$ after expanding. Unit cells with red line border are old unit cells forming the hardware graph before expanding $T(2, 2, 4)$. The unit cell with green dash border is an example for semi unit cell used to connect old unit cells after expanding.

(lines 7 - 8). After finishing the first step, we switch to the second step in which algorithm 2 assigns free nodes in clean paths from $\bar{u}$ to existing chains of nodes in P, and includes these assignment in the additional embedding ϕ' (lines 9 - 13). Our node assignment does not only ensure to generate new global connections (Definition 1) from the new chain $\phi(\bar{v})$ to the existing embedding, and preserve chain connections (Definition 1) in every chains, but it also distributes fairly the cardinality of new embedded nodes to the new chain, and existing chains based on degrees of nodes in P. As a result, it reduces the number of nodes needed in future embedding.

Lemma 1. *Assume that ϕ_{i-1} is a feasible embedding for $P[S_{i-1}]$ and $\bar{v}$ is new node to be embedded in the ith* turn, *the embedding ϕ_i generated with $\phi_i(v) := \phi_{i-1}(v) \cup \phi'(v)$ for $\forall v \in P$ is a feasible embedding of $P[S_i]$ with $S_i = S_{i-1} \cup \{\bar{v}\}$ if embedding ϕ' is not* empty embedding.

In some turns, it is possible that there exists no free node $\bar{u} \in H$ such that all chains $\phi(v)$ with $v \in A$ can connect to $\bar{u}$ by clean paths. We call it *isolated problem*.

C. Topology Adapting

Next, we show how to expand an adaptive topology to solve the *isolated problem*. Recall that using the Chimera model, the adaptive topology of H as $T(n, m, c)$ is an $n \times m$ grid of $K_{c,c}$ unit cells. Each unit cell $K_{c,c}$ is a bipartite graph with c nodes on each partite. Each node $u \in H$ has a coordinate (x_u, y_u, z_u) in the topology $T(n, m, c)$ with $x_u \in [0, n-1]$ indicates the

row index, $y_u \in [0, m-1]$ indicates the column index, and $z_u \in [0, 2c-1]$ indicates the bipartite index. In a unit cell, nodes on the right partite have connections to corresponding nodes in two adjacent unit cells in the same row while nodes on the left partite have connections to corresponding nodes in two adjacent unit cells in the same column (see in Figure 1).

Algorithm 3 presents the topology adapting algorithm. The algorithm takes current embedding ϕ, and current adaptive topology $T(n, m, c)$ as input. The current embedding of nodes located in the unit cell (x, y) in H is shifted by x rows and y columns to the bottom right corner of H (lines 4 - 6). As a result, there are two kinds of unit cells after expanding, called *empty unit cells* whose nodes are not assigned to any chains and *old unit cells* whose nodes are assigned to chains from the embedding ϕ. Algorithm 3 then, assigns nodes in empty unit cells located in between two old unit cells. These unit cells are called *semi unit cells*. Then, we denote semi unit cells in between two old unit cells in the same row as *row semi unit cells*, and semi unit cells in between two old unit cells in the same column as *column semi unit cells*. The rule of assignment is that nodes on the right partite of a row semi unit cell (x, y) are assigned to the same chain with the corresponding node in the adjacent left old unit cell $(x-1, y)$ while nodes on the left partite of a column semi unit cell (x, y) are assigned to the same chain with the corresponding node in the adjacent top old unit cell $(x, y-1)$. Figure 1 shows the expanding process. We realize that semi unit cells are bridges to keep chain connections, and global connections from ϕ after expanding. It implies that the embedding $\phi^\dagger$ expanded from ϕ_i is a feasible embedding of $P[S_i]$.

Lemma 2. *After expanding the current embedding ϕ to new embedding $\phi^\dagger$ by the Algorithm 3, the addition embedding ϕ' found by the Algorithm 2 with current embedding as $\phi^\dagger$ is not* empty embedding.

We conclude from Lemma 2 that our method avoids *isolated problem* after expanding the current embedding.

Theorem 1. *Given S_i as the embedded set, and ϕ_i as the embedding found by the Algorithm 2 after the ith turn, ϕ_i is a feasible embedding for $P[S_i]$.*

Theorem 1 implies that the resulting embedding ϕ after $|V_P| - |V(P')|$ turns is a feasible embedding of P in H.

Theorem 2. *The Algorithm 1 returns a feasible embedding for P after at most $3|V_P|$ iterations.*

IV. Experiments

In this section, we evaluate our method on synthetic dataset. We compare our method to two state-of-the-art methods: OCT-based [11] , and Minorminer [13] which was developed by D-Wave. Minorminer is the best heuristic method as of now for sparse logical graphs.

We construct logical graphs using three models, namely, Barabási-Albert with initialization by star graphs (BA_{star}), Barabási-Albert with initialization by complete graphs ($BA_{complete}$), and d-Regular. For each model, we generate 38 logical graphs by varying the number of nodes, and average node degree d as $|V_P| \in \{100, 200, \ldots, 1900\}$ and degree $d \in \{10, 20\}$.

Algorithm 1: Algorithm ATOM

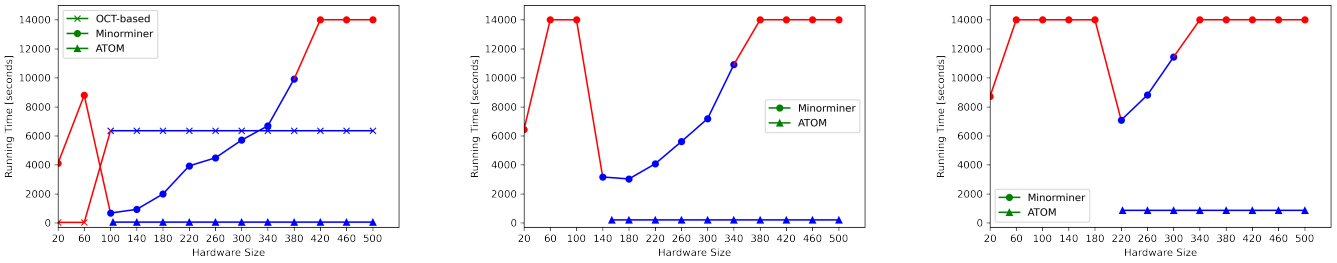
Input: Logical graph $P = (V_P, E_P)$, hardware graph $H = (V_H, E_H, T(n, m, c))$.
Output: A feasible embedding $\phi : V_P \to \mathcal{P}(V_H)$

1 Let the subgraph of H induced by $T(n, m, c)$ as H'.
2 Let the densest k-subgraph of P be P'.
3 Let the complete embedding of P' in H' be ψ.
4 Initialize ϕ_0 such that $\phi_0(v) := \psi(v)$ for $\forall v \in V_P$.
5 $S_0 := V(P')$
6 $i := 1$
7 $w[v] := 0$ for $\forall v \in V_P$
8 $isolated := False$
9 **while** $i \leq |V_P| - |V(P')|$ **do**
10 **if** *not* $isolated$ **then**
11 $\bar{v}$:= arg $\min_{v \in V_P \setminus S} \sum_{u \in S, (u,v) \in E_P} w_u$
12 ϕ' := NODE_EMBEDDING$(P, H, \phi_{i-1}, S_{i-1}, \bar{v})$
13 **if** $\exists v \in V_P$ *such that* $\phi'(v) \neq \emptyset$ **then**
14 $isolated$:= False
15 Initialize ϕ_i such that $\phi_i(v) := \phi_{i-1}(v) \cup \phi'(v)$ for $\forall v \in V_P$
16 $S_i := S_{i-1} \cup \{\bar{v}\}$; $w[\bar{v}] := i$; $i := i + 1$
17 **else**
18 $isolated$:= True
19 ϕ_{i-1}, T:= TOPOLOGY_ADAPTING(H, ϕ_{i-1})
20 **return** $\phi_{|V_P|-|V(P')|}$

We use the following three criteria to evaluate our method, and the competing methods.

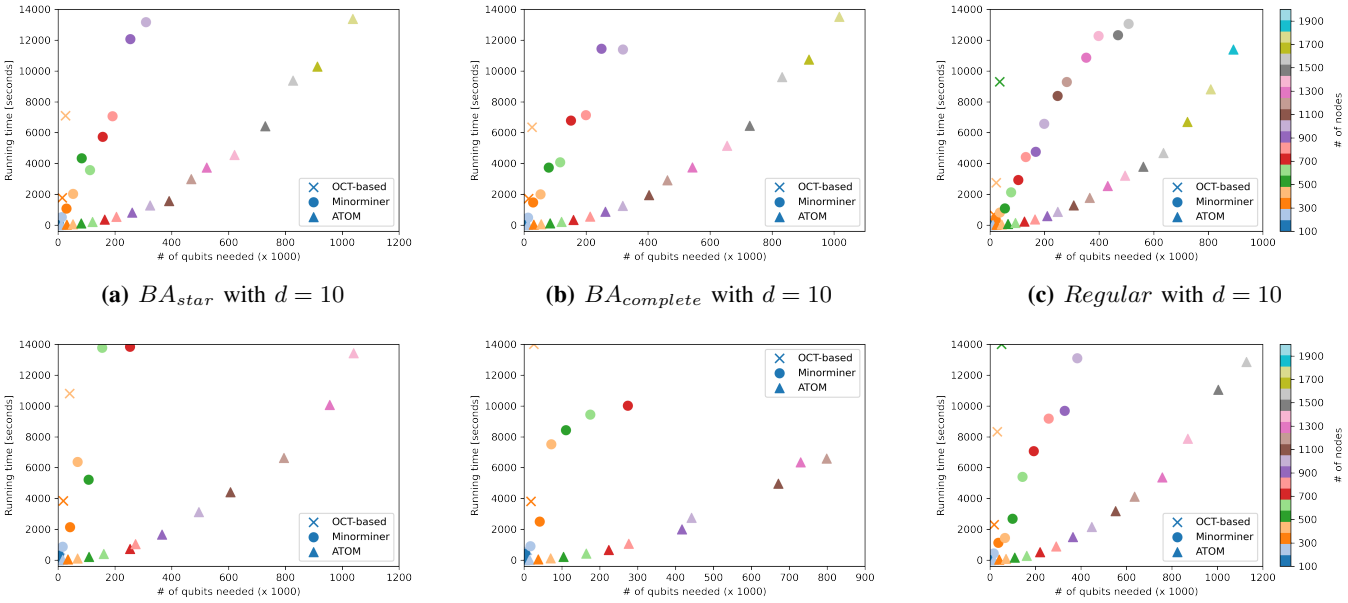
1) *Minimum hardware size*. We compute the smallest hardware size needed to include the resulting embedding. Since Minorminer, and OCT-based take a specific hardware size as input, we run their method with the hardware graph following the topology $T(n, n, 4)$ with $n \in \{20, 60, \ldots, 500\}$. Unlike these methods, ATOM adapts the hardware size to the given problem instance. Thus, we report the size of final adaptive topology as the minimum hardware size of ATOM.
2) *Running time*. We measure the time needed to return a feasible embedding, or the time needed to claim that no feasible embedding exists for the given hardware size. We set the *time limit* to 14000 seconds.
3) *The number of qubits needed*. We measure the number of nodes in H which are mapped by a node in P in the resulting embedding.

Environment. We implement ATOM in C++. We use Minorminer, and OCT-based code provided by their authors. For OCT-based method, we select *Fast-OCT-reduce* version as it is the most effective version of OCT-based. We run all



(a) # of nodes = 400 **(b)** # of nodes = 600 **(c)** # of nodes = 900

Figure 2: Running time of OCT-based, Minorminer, and ATOM for different hardware sizes. Logical graphs are generated using $BA_{complete}$ model with the number of nodes = 400, 600, 900 in **(a)**, **(b)**, **(c)** respectively. Blue points represents for hardware graph size for which the corresponding method can find a feasible embedding. Red points represents infeasible cases. In **(b)** and **(c)**, we do not report OCT-based as it cannot return any feasible embedding.



(a) BA_{star} with $d = 10$ **(b)** $BA_{complete}$ with $d = 10$ **(c)** $Regular$ with $d = 10$

(d) BA_{star} with $d = 20$ **(e)** $BA_{complete}$ with $d = 20$ **(f)** $Regular$ with $d = 20$

Figure 3: Running time and the number of qubits needed for embedding synthetic instances using BA_{star}, $BA_{complete}$, and $Regular$ graph models for degree $d \in \{10, 20\}$. Different colors indicate sizes of logical graphs ranging from 100 to 1900.

experiments on a Debian GNU/Linux machine with Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz.

a) Evaluation of the effect of hardware size: Recall that the fundamental contribution of this paper is that our method adapts to the given hardware size unlike existing state-of-the-art methods. We first evaluate how this affects the cost of solving the minor embedding problem. Figure 2, shows the minimum hardware size needed to find a feasible embedding from OCT-based, Minorminer, and ATOM for different logical and hardware graph sizes. Our resuts demonstrate that ATOM outperforms competing methods for all logical and hardware graph sizes. The performance of both OCT-based, and Minorminer depends on how we select hardware sizes, while ATOM computes the minimum hardware size from final embedding. We observe that, OCT-based, and Minorminer return infeasible solutions if selected hardware size is too small. In addition, Minorminer can result in large running time for inappropriate hardware sizes. Second, minimum hardware size from ATOM is comparable to two other methods in all cases. Thus, we conclude that ATOM is effective for finding embedding on real QPU topology whose size is always fixed.

b) Evaluation of the number of qubits needed: The success of a QA algorithm depends on the number of qubits it uses as well as the time it takes to embed the given optimization problem to so many qubits. Ideally, it is preferable to embed into a small number of qubits in short amount of time. In figure 3, we show the relation between the running time, and the number of qubits needed for three methods. We observe that

Algorithm 2: NODE EMBEDDING

Input: Logical graph $P = (V_P, E_P)$, hardware graph $H = (V_H, E_H, T(n, m, c))$, current embedding ϕ, current embedded set S, new node $\bar{v}$
Output: Additional embedding ϕ'

Let the subgraph of H induced by $T(n, m, c)$ as $H' = (V_{H'}, E_{H'})$.
Let the embedded neighbour set of $\bar{v}$ be $A = \{v | v \in S, (\bar{v}, v) \in E_P\}$.
Let $Z_{uv} := \{z_0, z_1, z_2, ..., z_{L_{uv}}\}$ be the clean shortest path from node u to the chain $\phi(v)$.
Let Z be the set of all possible clean shortest paths Z_{uv} with $u \in V_{H'}, v \in A$.
Initialize ϕ' with $\phi'(v) := \emptyset$ for $\forall v \in V_P$.
if $\nexists u \in V_{H'}$ *such that* $\exists Z_{uv} \in Z$ *for* $\forall v \in A$ **then**
  return ϕ'
$\bar{u} := \text{argmin}_{u \in V_{H'}, \phi^{-1}(u) = -1} \sum_{v \in A} |Z_{uv}|$
$\phi'(\bar{v}) \leftarrow \phi'(\bar{v}) \cup \{\bar{u}\}$
for $v \in A$ **do**
  Find the biggest index i_v *from 1 to* $L_{\bar{u}v} - 1$, *such that there exists* $v' \neq v, v' \in A, z_i \in Z_{\bar{v}v'}$.
  $\delta := \lfloor \frac{\gamma_{\bar{v}}}{\gamma_{\bar{v}} + \gamma_v} (L_{\bar{u}v} - 1 - i) \rfloor$
  $\phi'(\bar{v}) \leftarrow \phi'(\bar{v}) \cup \{z_i | 1 \leq i \leq i_v + \delta\}$
  $\phi'(v) \leftarrow \phi'(v) \cup \{z_i | i_v + \delta + 1 \leq i \leq L_{\bar{u}v}\}$
return ϕ'

Algorithm 3: TOPOLOGY ADAPTING

Input: Hardware graph $H = (V_H, E_H, T(n, m, c))$, current embedding ϕ
Output: An expanded embedding $\phi^\dagger : V_P \to \mathcal{P}(V_H)$, and the adaptive topology T after expanding

Initialize $\phi^\dagger$ such that $\phi^\dagger(v) := \emptyset$ for $\forall v \in V_P$.
for $v \in V_P$ **do**
  for $u \in \phi(v)$ **do**
    Let (x, y, z) be the coordinate of u.
    Let $u' \in H$ with coordinate of $(2x, 2y, z)$.
    $\phi^\dagger(v) \leftarrow \phi^\dagger(v) \cup \{u'\}$
    if $z \leq c - 1$ **then**
      Let $u'' \in H$ with coordinate $(2x + 1, 2y, z)$
      $\phi^\dagger(v) \leftarrow \phi^\dagger(v) \cup \{u''\}$
    else
      Let $u'' \in H$ with coordinate $(2x, 2y + 1, z)$
      $\phi^\dagger(v) \leftarrow \phi^\dagger(v) \cup \{u''\}$
return $\phi^\dagger$, T(2n, 2m, c)

ATOM outperforms two others in running time for all cases. On average, ATOM is up to 20 times faster than Minorminer, and up to 66 times faster than OCT-based.

As a result, with the timeout is 14000 seconds, ATOM is able to find feasible embeddings for logical graphs with size of 1800, and 1500 for $d = 10$, and $d = 20$ respectively, while the corresponding numbers for Minorminer is 1500, and 1000, and for OCT-based is 500, and 400. The number of qubits needed for embedding resulted from ATOM is comparable with OCT-based, and Minorminer, and even better in some cases. Therefore, ATOM is able to provide a feasible embedding for QA much faster than Minorminer, and OCT-based can, but do not require much more additional resources. That leads to more economic advantages since with ATOM, people can solve more optimization problems in QA at the same time.

V. Conclusion

In this work, we introduce a novel notion of adaptive topology, and propose a minor embedding algorithm, namely Adaptive TOpology eMbedding (ATOM). ATOM is able to provide a feasible embedding without fixing the hardware graph size. Our experimental results show that ATOM requires much less running time to find a feasible embedding without demanding additional resources, compared with two state-of-the-art methods: OCT-based, and Minorminer. In the future, along with the rapid expansion in QPU size, and increasing demand on solving large optimization problems quickly, ATOM will be an efficient choice for minor embedding in QA.

References

[1] Arute and et al, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, pp. 505–510, Oct 2019.
[2] A. A. Abd El-Latif and et al, "Quantum-inspired blockchain-based cybersecurity: Securing smart edge utilities in iot-based smart cities," *Information Processing & Management*, vol. 58, no. 4, p. 102549, 2021.
[3] Jerbi and et al, "Parametrized quantum policies for reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 34, pp. 28362–28375, 2021.
[4] Guillaume and et al, "Deep space network scheduling using quantum annealing," *IEEE Transactions on Quantum Engineering*, vol. 3, pp. 1–13, 2022.
[5] Ding and et al, "Implementation of a hybrid classical-quantum annealing algorithm for logistic network design," *SN Computer Science*, vol. 2, p. 68, Feb 2021.
[6] Neukart and et al, "Traffic flow optimization using a quantum annealer," *Frontiers in ICT*, vol. 4, 2017.
[7] F. Ishizaki, "Computational method using quantum annealing for tdma scheduling problem in wireless sensor networks," in *2019 13th International Conference on Signal Processing and Communication Systems (ICSPCS)*, pp. 1–9, 2019.
[8] V. Choi, "Minor-embedding in adiabatic quantum computation: II. minor-universal graph design," *Quantum Information Processing*, vol. 10, pp. 343–353, Jun 2011.
[9] C. Klymko, B. Sullivan, and T. Humble, "Adiabatic quantum programming: Minor embedding with hard faults," *Quantum Information Processing*, vol. 13, 10 2012.
[10] T. Boothby, A. D. King, and A. Roy, "Fast clique minor generation in chimera qubit connectivity graphs," *Quantum Information Processing*, vol. 15, pp. 495–508, Jan 2016.
[11] T. D. Goodrich, B. D. Sullivan, and T. S. Humble, "Optimizing adiabatic quantum program compilation using a graph-theoretic framework," *Quantum Information Processing*, vol. 17, p. 1–26, may 2018.
[12] Bernal and et al, "Integer programming techniques for minor-embedding in quantum annealers," in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 112–129, 2020.
[13] J. Cai, W. G. Macready, and A. Roy, "A practical heuristic for finding graph minors," 2014.
[14] J. P. Pinilla and S. J. E. Wilton, "Layout-aware embedding for quantum annealing processors," in *High Performance Computing*, pp. 121–139, Springer International Publishing, 2019.
[15] K. Boothby, P. Bunyk, J. Raymond, and A. Roy, "Next-generation topology of d-wave quantum processors," 2020.