Multilevel Combinatorial Optimization across Quantum Architectures

HAYATO USHIJIMA-MWESIGWA*†, Fujitsu Laboratories of America, Inc.
RUSLAN SHAYDULIN*, School of Computing, Clemson University
CHRISTIAN F. A. NEGRE, Theoretical Division, Los Alamos National Laboratory
SUSAN M. MNISZEWSKI, Computer, Computational, & Statistical Sciences Division, Los Alamos National Laboratory
YURI ALEXEEV, Computational Science Division, Argonne National Laboratory
ILYA SAFRO†, School of Computing, Clemson University

Emerging quantum processors provide an opportunity to explore new approaches for solving traditional problems in the post Moore's law supercomputing era. However, the limited number of qubits makes it infeasible to tackle massive real-world datasets directly in the near future, leading to new challenges in utilizing these quantum processors for practical purposes. Hybrid quantum-classical algorithms that leverage both quantum and classical types of devices are considered as one of the main strategies to apply quantum computing to large-scale problems. In this article, we advocate the use of multilevel frameworks for combinatorial optimization as a promising general paradigm for designing hybrid quantum-classical algorithms. To demonstrate this approach, we apply this method to two well-known combinatorial optimization problems, namely, the Graph Partitioning Problem, and the Community Detection Problem. We develop hybrid multilevel solvers with quantum local search on D-Wave's quantum annealer and IBM's gate-model based quantum processor. We carry out experiments on graphs that are orders of magnitude larger than the current quantum

*Both authors contributed equally to this work.
†Work on this article was in part performed while the author was affiliated with Clemson University.

This work was supported in part with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. This work was supported in part by NSF award #1522751. High-performance computing resources at Clemson University were supported by NSF award MRI #1725573. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. This research also used the resources of the Argonne Leadership Computing Facility, which is DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. Yuri Alexeev and Ruslan Shaydulin were supported by the DOE Office of Science. The authors would also like to acknowledge the NNSA's Advanced Simulation and Computing (ASC) program at Los Alamos National Laboratory (LANL) for use of their Ising D-Wave 2000Q quantum computing resource. LANL is operated by Triad National Security, LLC, for the National Nuclear Security Administration of U.S. Department of Energy (Contract No. 89233218NCA000001). Susan Mniszewski and Christian Negre were supported by the ASC program at LANL. Assigned: Los Alamos Unclassified Report 19-30113.

Authors' addresses: H. Ushijima-Mwesigwa (corresponding author), Fujitsu Laboratories of America, Inc. Sunnyvale, CA, 94085; email: hayato@us.fujitsu.com; R. Shaydulin (corresponding author), School of Computing, Clemson University, Clemson, SC, 29634; email: rshaydu@g.clemson.edu; C. F. A. Negre, Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM, 87545; email: cnegre@lanl.gov; S. M. Mniszewski, Computer, Computational, & Statistical Sciences Division, Los Alamos National Laboratory, Los Alamos, NM, 87545; email: smm@lanl.gov; Y. Alexeev, Computational Science Division, Argonne National Laboratory, Argonne, IL, 60439; I. Safro, School of Computing, Clemson University, Clemson, SC, 29634; email: isafro@clemson.edu.

2643-6817/2021/02-ART1 $15.00
https://doi.org/10.1145/3425607

hardware size, and we observe results comparable to state-of-the-art solvers in terms of quality of the solution. **Reproducibility**: Our code and data are available at Reference [1].

CCS Concepts: • **Mathematics of computing → Graph algorithms**; *Combinatorial optimization*; • **Hardware → Quantum computation**;

Additional Key Words and Phrases: NISQ, quantum annealing, graph partitioning, modularity, community detection

ACM Reference format:
Hayato Ushijima-Mwesigwa, Ruslan Shaydulin, Christian F. A. Negre, Susan M. Mniszewski, Yuri Alexeev, and Ilya Safro. 2021. Multilevel Combinatorial Optimization across Quantum Architectures. *ACM Trans. Quantum Comput.* 2, 1, Article 1 (February 2021), 29 pages.
https://doi.org/10.1145/3425607

1 INTRODUCTION

Across different domains, computational optimization problems that model large-scale complex systems often introduce a major obstacle to solvers even if tackled with high-performance computing systems. The reasons include a large number of variables and even larger number of interactions, dimensionality required to describe each variable or interaction, and time slices. The combinatorial and mixed-integer optimization problems introduce additional layers of complexity, with integer variables often making the problem NP-hard (e.g., in cases of nonlinearity and nonconvexity). A common practical approach for solving these problems is to use iterative methods. The iterative methods, while being composed with completely different algorithmic principles, typically share a common property: Several fast improvement iterations are followed by a long tail of slow improvement iterations [46, 100]. Usually, in such iterative algorithms, solving a large-scale system by using first-order optimization (e.g., gradient descent) or limited observable information (e.g., local search) methods per iteration leads to a local optimum. In other words, local methods tend to converge to a local optimum, which often corresponds to a solution of much lower quality than the true global optimum [37]. Moreover, in some cases, another problem may exist within each iteration: The algorithms used to solve them are not necessarily exact. To accelerate the solvers at each iteration, various heuristics, parallelism-friendly methods, and ad hoc tricks are employed, which often reduce the quality of the solution.

In this article, we take steps towards building more robust solvers for mid- to large-scale combinatorial optimization problems by fusing two areas whose simultaneous application is only beginning to be explored, namely, quantum computing and multiscale methods. Recent advances in quantum computing provide a new approach for algorithm development for many combinatorial optimization problems. However, Noisy Intermediate Scale Quantum (NISQ) devices are widely expected to be limited to a few hundred, and for certain sparse architectures up to a few thousands qubits. The current state of quantum computing theory and engineering suggests moderately optimistic expectations. In particular, it is believed that in the near future, we will witness relatively robust small-scale architectures with much less noise. This would allow algorithms like the Quantum Approximate Optimization Algorithm (QAOA) and Quantum Annealing (QA) to be run on hardware with minimal error correction efforts. Given the realistic level of precision and, in the case of QAOA, ansatz depth, these algorithms are widely considered to be prime candidates for demonstrating quantum advantage, which is solving a computationally hard problem (such as NP-hard) faster than classical state-of-the-art algorithms. Such algorithms are our first building block.

The multiscale optimization method is our second building block. These methods have been developed to cope with large-scale problems by introducing an approach to avoid entering false local

attraction basins (local optima), a complementary method to stochastic and multistart strategies that help escape it if trapped. Because of historical reasons, on graph problems, they have been termed *multilevel* (rather than multiscale), which we will use here. The multilevel (or multiscale) methods have a long history of breakthrough results in many different optimization problems [14, 18, 29, 31, 33, 43, 53, 57, 71, 72, 74, 75, 77, 78, 83, 84] and have been implemented on a variety of hardware architectures. The success of multilevel methods for optimization problems supports our optimism about the ideas proposed in this article.

No unique prescription exists for how to design multilevel algorithms, but the main idea behind them is to *think globally while acting locally* on a hierarchy of coarse representations of the original large-scale optimization problem. A multilevel algorithm therefore begins by constructing such a hierarchy of progressively smaller (coarser) representations of the original problem. The goal of the next coarser level in this hierarchy is to approximate the current level problem with a coarser one that has fewer degrees of freedom and thus can be solved more effectively. When the coarse problem is solved, its solution is projected back to the finer level and further refined, a stage that is called uncoarsening. As a result of such a strategy, the multilevel framework is often able to significantly improve the running time and solution quality of optimization methods. The quality of multilevel algorithms in large part depends on that of the optimization solvers applied at all stages of the multilevel framework. In many cases, these locally acting optimization solvers are either heuristics that get stuck in a local optimum or exact solvers applied on a small number of variables (i.e., on subproblems). In both cases, the quality of a global solution can significantly suffer, depending on the quality of the solution from the local solver. The optimization algorithms running on the NISQ devices that may replace such local solvers are expected to be a critical missing component to achieve a game-changing breakthrough in multilevel methods for combinatorial optimization. Although the performance of these NISQ-era optimization algorithms is not fully understood (see Section 2.3 for an overview), in this work, we do not attempt to rigorously benchmark them. Rather, we focus on the problems arising when integrating these optimization algorithms into a multilevel framework. The proposed synergy between multilevel algorithms and future quantum devices is intended to bridge the gap between slow exact solvers (that are unrealistic to cope with relatively large instances in a reasonable time neither for classical nor NISQ devices) and fast suboptimal heuristics (that are practically employed everywhere) that may never achieve the optimal solution no matter how much time they are given.

In this article, we introduce Multilevel Quantum Local Search (ML-QLS), which uses an iterative refinement scheme on NISQ devices within a multilevel framework. ML-QLS extends the Quantum Local Search (QLS) [87, 88] approach to solve larger problems. This work builds on early results using a multilevel framework and the D-Wave quantum annealer for the graph partitioning [96]. We demonstrate the general approach of solving combinatorial optimization problems with NISQ devices in a multilevel framework on two well-known problems. In particular, we solve the Graph Partitioning Problem and the Community Detection Problem on graphs up to approximately 29,000 nodes using subproblem sizes of 20 and 64 that map onto NISQ devices such as IBM Q Poughkeepsie (20 qubits) and D-Wave 2000Q (~2,048 qubits). Such graphs are orders of magnitude larger than those solved by state-of-the-art hybrid quantum-classical methods. To implement this approach, we develop a novel efficient subproblem formulation method.

In contrast, some of the authors of this article have previously developed quantum and quantum-classical algorithms for the Graph Partitioning Problem and the Community Detection Problem for multiple parts (>2) [61, 95]. These did not use a multilevel approach, instead an *all at once* or concurrent approach was employed.

The rest of this article is organized as follows: In Section 2, we discuss the relevant background on quantum optimization and multilevel methods, and define the problems. In Sections 3 and 4, we

discuss the hybrid quantum-classical multilevel algorithm and computational results, respectively. A discussion of the outlook and important open problems that represent major future research directions are presented in Section 5.

2 BACKGROUND

The methods proposed and implemented in this work aim to solve large graph problems by integrating NISQ optimization algorithms to a multilevel scheme. In this section, we provide a brief introduction into all three components: target graph problems (Section 2.1), quantum optimization (Section 2.2), and multilevel methods (Section 2.4).

Many optimization problems discussed in this work are posed in Ising form. The Ising model is a common mathematical abstraction used to represent the energy of n discrete spin variables $\sigma_i \in \{-1, 1\}$, $1 \leq i \leq n$, and interactions J_{ij} between σ_i and σ_j. For each spin variable σ_i, a local field h_i is specified. The energy of a configuration σ is given by the Hamiltonian function:

$$H(\sigma) = \sum_{i,j} J_{ij}\sigma_i\sigma_j + \sum_i h_i\sigma_i, \quad \sigma_i \in \{-1, 1\}. \tag{1}$$

An equivalent mathematical formulation is the Quadratic Unconstrained Binary Optimization (QUBO) problem. The objective of a QUBO problem is to minimize (or maximize) the following function:

$$H(x) = \sum_{i<j} Q_{ij}x_ix_j + \sum_i Q_{ii}x_i, \quad x \in \{0, 1\}.$$

2.1 Problem Definitions

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. We denote by n and m the numbers of nodes and edges, respectively. For each node i, define $\mathbb{v}_i \in \mathbb{R}$ as the volume of node i and $A_{ij} \in \mathbb{R}$ as the positive weight of edge (i, j). For a fixed integer k, the *Graph Partitioning Problem* is to find a partition $V_1, \ldots, V_k$ of the vertex set V into k parts with equal total node volume such that the total weight of *cut edges* is minimized. A *cut edge* is defined as an edge whose end points are in different partitions. A requirement of equal total sizes of V_i for all i is sometimes referred as *perfectly balanced* graph partitioning, otherwise an imbalancing parameter is usually introduced to allow imbalanced partitions [16]. However, in this work, we deal with perfect balancing constraints and limit the number of parts to $k = 2$. In this case, we can write the GP problem as the following quadratic program:

$$\begin{aligned} \max \quad & \mathbf{s}^T A\mathbf{s} \\ \text{s.t.} \quad & \sum_{i=1}^{n} \mathbb{v}_i s_i = 0 \\ & s_i \in \{-1, 1\},\ i = 1, \ldots, n, \end{aligned} \tag{2}$$

which, as shown in Reference [95], can be reformulated into the following Ising model:

$$\begin{aligned} \max \quad & \mathbf{s}^T(\beta A - \alpha \mathbb{v}\mathbb{v}^T)\mathbf{s} \\ \text{s.t.} \quad & s_i \in \{-1, 1\},\ i = 1, \ldots, n, \end{aligned} \tag{3}$$

for some constants $\alpha, \beta > 0$, where $\mathbb{v}$ is a column vector of volumes such that $(\mathbb{v})_i = \mathbb{v}_i$.

Maximization of modularity is a famous problem in network science where the goal is to find communities in a network through node clustering (also known as modularity clustering) [62]. For the graph G, the problem of Modularity Maximization is to find a partitioning of the vertex

set into one or more parts (communities) that maximizes the modularity metric. The modularity matrix is a symmetric matrix given by

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2|E|}, \tag{4}$$

where k_i is the weighted degree of node i, namely, $k_i = \sum_j A_{ij}$. Whereas the modularity is typically defined on unweighted graphs, within the multilevel framework, due to the coarsening of nodes, we primarily work with weighted graphs. It can equivalently be written in matrix-vector notation as

$$B = A - \frac{1}{2|E|}\mathbb{k}\mathbb{k}^T, \tag{5}$$

where $\mathbb{k}$ is a vector of weighted degrees of the nodes in the graph. For two communities, the *Modularity Maximization Problem*, also referred to as the *Community Detection Problem*, can be written in Ising form as follows:

$$\begin{aligned} \max \quad & \frac{1}{4|E|}\mathbf{s}^T\left(A - \frac{1}{2|E|}\mathbb{k}\mathbb{k}^T\right)\mathbf{s} \\ \text{s.t.} \quad & s_i \in \{-1, 1\},\ i = 1, \ldots, n, \end{aligned} \tag{6}$$

where the objective value of Equation (6), for a given assignment of resulting communities, is referred to as the *modularity*. For more than two communities, the Ising formulation of the Community Detection Problem is given in Reference [61].

Note that the above formulation of Modularity Maximization can be viewed as the Graph Partitioning Problem in the Ising model given in Equation (3) where the volume of a node is defined as the weighted degree and the penalty constants $\beta = 1, \alpha = \frac{1}{2|E|}$. We exploit this deep duality between the two problems in our implementation.

2.2 Optimization on NISQ Devices

In recent years, we have seen a number of advances in quantum optimization algorithms that can be run on NISQ devices. The two most prominent ones are the Quantum Approximate Optimization Algorithm (QAOA) and Quantum Annealing (QA), which are inspired by the adiabatic theorem. There are many formulations of the adiabatic theorem (see Reference [6] for a comprehensive review), but all of them stem from the adiabatic approximation formulated by Kato in 1950 [44]. Adiabatic approximation states, roughly, that a system prepared in an eigenstate (e.g., a ground state) of some time-dependent Hamiltonian $H(t)$ will remain in the corresponding eigenstate[1] provided that $H(t)$ is varied "slowly enough." The requirement on the evolution time scales as $O(1/\Delta^2)$ in the worst case [25], where Δ is the minimum gap between the ground and first excited state of $H(t)$.

QA is a special case of AQC limited to stochastic Hamiltonians. The transverse field Hamiltonian

$$H_M = \sum_i \sigma_i^x \tag{7}$$

is used as the initial Hamiltonian. The final Hamiltonian is a classical Ising model Hamiltonian with the ground state encoding the solution of the original problem:

$$H_C = \sum_{ij} J_{ij} s_i s_j + \sum_i h_i s_i, \quad s_i \in \{-1, +1\}.$$

[1]A note on terminology: a Hamiltonian H is a Hermitian operator. The spectrum of H corresponds to the potential outcomes if one was to measure the energy of the system described by H. $|\psi\rangle$ is an eigenstate of a system described by Hamiltonian H with energy $\lambda \in \mathbb{R}$ if $H|\psi\rangle = \lambda|\psi\rangle$. In other words, $|\psi\rangle$ is an eigenvector of H with real eigenvalue λ.

The evolution of the system starts in the ground state of H_M and is described by a time-dependent Hamiltonian

$$H(t) = \frac{t}{T}H_C + \left(1 - \frac{t}{T}\right)H_M,\ t \in (0, T). \tag{8}$$

QAOA extends the logic of AQC to gate-model quantum computers and can be interpreted as a discrete approximation of the continuous QA schedule, performed by applying two alternating operators:

$$W(\beta_k) = e^{-i\beta_k H_M} \text{ and } V(\gamma_k) = e^{-i\gamma_k H_C},$$

where $W(\beta_k)$ corresponds to evolving the system with Hamiltonian H_M for a period of time β_k, and $V(\gamma_k)$ corresponds to evolving H_C for time γ_k. Similarly to QA, the evolution begins in the ground state of H_M, namely, $|+\rangle^{\otimes n}$. Alternating operators are applied to produce the state:

$$|\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle = e^{-i\beta_p H_M} e^{-i\gamma_p H_C} \ldots e^{-i\beta_1 H_M} e^{-i\gamma_1 H_C} |+\rangle^{\otimes n} = U(\boldsymbol{\beta}, \boldsymbol{\gamma}) |+\rangle^{\otimes n}. \tag{9}$$

An alternative implementation was proposed, inspired by the success of the Variational Quantum Eigensolver (VQE) [69, 104]. A variational implementation of QAOA combines an ansatz $U(\boldsymbol{\beta}, \boldsymbol{\gamma})$ (that can be different from the alternating operator described above) and a classical optimizer. A commonly used ansatz is a hardware-efficient ansatz [40], consisting of alternating layers of entangling and rotation gates. The algorithm starts by preparing a trial state by applying the parameterized gates to some initial state: $|\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle = U(\boldsymbol{\beta}, \boldsymbol{\gamma}) |+\rangle^{\otimes n}$. In the next step, the state $|\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle$ is measured, and the classical optimization algorithm uses the result of the measurement to choose the next set of parameters $\boldsymbol{\beta}, \boldsymbol{\gamma}$. The goal of the classical optimization is to find the parameters $\boldsymbol{\beta}, \boldsymbol{\gamma}$ corresponding to the optimal QAOA "schedule," that is, the schedule that produces the ground state of the problem Hamiltonian H_C:

$$\boldsymbol{\beta}_*, \boldsymbol{\gamma}_* = \arg\min_{\boldsymbol{\beta}, \boldsymbol{\gamma}} \langle\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})| H_C |\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle. \tag{10}$$

Both QA and QAOA have been successfully implemented in hardware by a number of companies, universities, and national laboratories [5, 20, 63, 65, 66, 70].

2.3 On the Scalability of Quantum Optimization Heuristics

The question of asymptotic scaling is the central question in the analysis of algorithms. Unfortunately, for many of the most promising quantum optimization algorithms, rigorous analysis (such as provable approximation ratios) beyond the most simple problems is unavailable. Therefore, researchers have to resort to experimental evaluations and back-of-the-envelope projections. Such approaches give rise to the second major complication, namely, the fact that empirical results on small problem instances are almost totally uninformative about the overall scaling behavior. Famously, the adiabatic quantum algorithm initially appeared to be practically useful for solving NP-complete problems in polynomial time based on numerical simulations of problems of up to tens of variables [26, 27]. Later analysis, however, has shown that for many problems, both synthetic ones that are classically easy and hard problems like 3-SAT [97], eigengaps diminish exponentially, leading to exponential worst-case running time for the adiabatic quantum algorithm [6]. This provides a cautionary example for researchers trying to analyze modern quantum optimization approaches. The exact number of variables needed for the separation between polynomial and exponential scaling to become apparent varies from problem to problem, and the separation might not be clear for small problem instances (i.e., the scaling that is in fact exponential looks polynomial for small instance sizes). However, the increased size of the simulations as well as the hardware (in the case of quantum annealers, reaching thousands of qubits) provides increasing confidence in the potential of quantum optimization heuristics.

Performance on the D-Wave quantum annealer depends on the input, the solver, and the hardware platform [23, 45, 49, 56]. Changes to the solver include modifying the anneal time or schedule. In this case the platform is the D-Wave 2000Q. Preprocessing strategies such as variable reduction, error mitigation, and improved embeddings applied to the input contribute to more optimized performance. The solver tuning strategies include finding an optimal anneal time dependent on the problem, as well as modifying the anneal schedule. Longer anneal times may be required for larger problems. Performance scaling for quantum applications is usually assessed by measuring the dependence of the time to solution (TTS) (for optimized run parameters) on the problem size, as often done for classical applications.

Considering the ground-state success probability for the Sherrington-Kirkpatrick (SK) model and MAX-CUT problems of increasing size was shown to be helpful in understanding the scaling [34]. When evaluating the TTS, one should optimize the run parameters as much as possible, in particular the optimal annealing time.

Work focused on determining the optimal anneal time on a quantum annealer over classical simulated annealing (SA) for logical-planted instances demonstrated a scaling advantage over SA on the D-Wave quantum annealer [7]. A scaling advantage in both system size and inverse temperature was demonstrated for simulation of frustrated magnetism in quantum condensed matter for the D-Wave quantum annealer over classical path-integral Monte Carlo [48]. An approach to benchmarking quantum annealers using specially crafted problems with varying hardness has been proposed [45]. Using a specially crafted problem class, a scaling advantage of algorithms simulating quantum annealing over simulated annealing has been demonstrated numerically for problems too large to be represented on available quantum hardware [23]. This class of problems has been extended and used to show that D-Wave quantum annealer can significantly outperform non-tailored classical state-of-the-art solvers [49].

A D-Wave API is available for collecting timing information on the details of the Quantum Processing Unit (QPU) access time [21]. The QPU access time consists of programming time, sampling time, postprocessing time, and a small amount of time spent for the QPU to regain its temperature per sample. Programming time measures how long it takes to initialize the problem on the QPU. The sampling time is further broken down into per sample times for the anneal, readout, and delay. With this API, runtime scaling for the quantum isomer search problem on the D-Wave QA was shown to grow linearly with problem size [94]. In this case the problem size is defined by the number of carbon atoms n in an alkane, which translates to $4(n-2)$ variables; and the number of isomer solutions increases with the number of carbon atoms.

The scaling of Quantum Approximate Optimization Algorithm (QAOA) is particularly hard due to two factors, namely the presence of outer-loop optimization and the lack of understanding of the scaling of the depth of QAOA circuit required to achieve a good solution. The first factor complicating the analysis of QAOA is the outer-loop optimization, i.e., the need to optimize parameters $\boldsymbol{\beta}, \boldsymbol{\gamma}$ in Equation (9). As the landscape is known to be highly non-convex [82, 85, 105], this optimization becomes a daunting task. Little is known about the structure of this landscape, making it hard to provide an upper bound on the computational hardness of the problem. At the time of writing the best known upper bound is that the problem of optimizing $\boldsymbol{\beta}, \boldsymbol{\gamma}$ is as hard as finding a global minimizer to a nonconvex problem, where even verifying a feasible point is a local minimizer is NP-hard [58]. However, in practice a number of techniques have been developed to successfully solve this problem. While the structure is hard to analyze, there have been successful attempts to leverage it using machine learning techniques [28, 47, 98] or accelerate search with multi-start optimization [85]. These results make us hopeful that with the help of a pre-trained model, high-quality QAOA parameters can be found in a small number of objective evaluations. There have been promising results showing that in higher-depth regime for some problems it is possible to

avoid optimization altogether and use a smoothly extrapolated set of parameters reminiscent of an adiabatic schedule [105].

The second factor in the lack of analytical and empirical results on QAOA behavior in low- to medium-depth regime (e.g., $5 \leq p \leq 100$). Analytically, QAOA appears to be hard to analyze beyond $p = \{1, 2\}$ for non-trivial problems [13, 92, 102]. At the same time, even for very simple problems and small instance sizes it is clear that achieving a good solution requires going beyond at least $p = 5$ [19, 82, 92]. Therefore, we have to rely on empirical results to answer the question of exactly how large the p needs to be to achieve a good solution. This empirical evaluation is impeded by the complexity of simulating QAOA in medium depths. On one end, traditional state-vector-based simulators have running time exponential in number of *qubits*, limiting the problem sizes to tens of variables. On the other end, tensor network based simulators have running time that is exponential in the number of *gates*, limiting the depth of the QAOA circuit that can be simulated. At the time of the writing, the state-of-the-art simulators were limited to simulating a thousand qubits to depth $p = 5$ [38]. These two constraints (on number of qubits for one simulation approach and on the depth for the other) make it challenging to numerically analyze QAOA performance in the crucial zone of medium-sized problems (hundreds to thousands of variables) and medium-depth circuit ($p > 10$). The high levels of noise and small number of available qubits make this analysis impossible on the currently available quantum hardware. All of the aforementioned complications contribute to the lack of the results showing how QAOA depth p scales with the size of the problem. At the same time, results presented in References [19, 105] indicate that at least for the problem sizes small enough to fit on near-term quantum computers, $10 \leq p \leq 30$ is sufficient to achieve high-quality solutions.

Due to the limitations of the available hardware, in this article, we do not run full QAOA ansatz on the IBM Q hardware. Instead, we use a shallow-depth hardware-efficient ansatz (HEA) [40]. Much less is known about the potential of such ansatzes to produce quantum speedup. Numerical experiments on small problem instances do not show that quantum entanglement provides an advantage in optimization [59]. At the same time, recent analytical results show that HEAs are efficiently simulatable classically in constant depth [60] and suffer from exponentially vanishing gradients in polynomial depth [55] (here depth is a function of number of qubits). A recent result shows that in logarithmic depth regime gradient vanishes only polynomially, making HEAs trainable in this regime [17]. This result indicates a potential for quantum advantage using HEAs in logarithmic depth, as they are both hard to simulate classically and do not suffer from exponentially vanishing gradients. Evaluating the potential for quantum advantage from using logarithmic-depth HEAs in QAOA setting for optimization remains an open problem.

2.4 Multilevel Combinatorial Optimization

The goal of the multilevel approach for optimization problems on graphs is to create a hierarchy of coarsened graphs $G_0 = G, G_1, \ldots, G_k$ in such a way that the next coarser graph G_{i+1} "approximates" some properties of G_i (that are directly relevant to the optimization problem of interest) with fewer degrees of freedom. After constructing such a hierarchy, the coarsening is followed by solving the problem on G_k as best as we can (preferably exactly) and then the uncoarsening projects the solution back to G_0 through gradual refinement at all levels of the hierarchy, with a refined solution at level $i + 1$ serving as the initial solution at level i. The entire coarsening-uncoarsening process is called a V-cycle. Other variations of hierarchical coarsening-uncoarsening schemes include W- and F-cycles [15]. Figure 1 presents an outline of a V-cycle.

Typically, when solving problems on graphs in which nodes represent the optimization variables (such as those in the partitioning and community detection), having fewer degrees of freedom

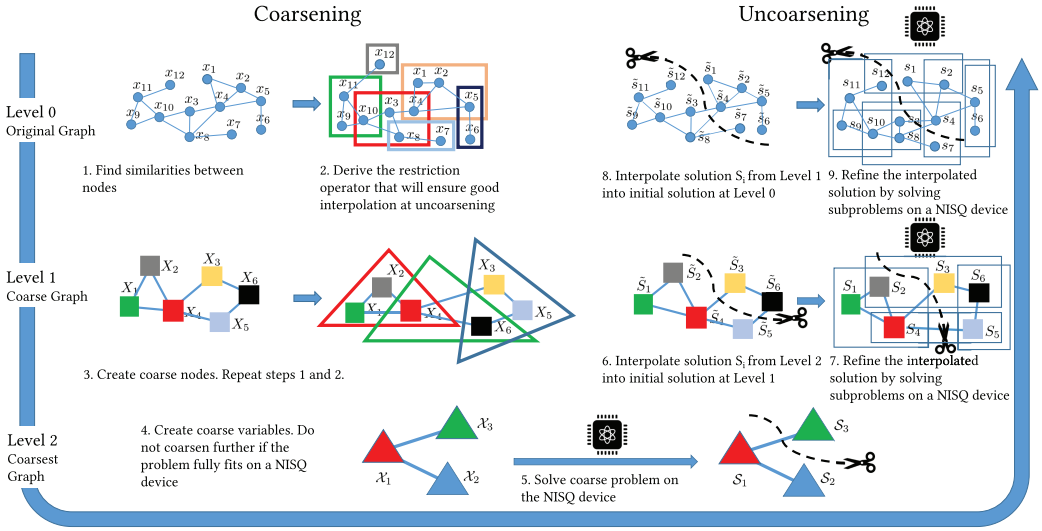


Fig. 1. V-cycle for a graph problem. First, the problem is iteratively coarsened (left). Second, the coarse problem is solved using a NISQ optimization solver (bottom). Finally, the problem is iteratively uncoarsened and the solution is refined using a NISQ solver (right).

implies a decreased number of nodes in each next coarser graph: $|V_0| > |V_1| > |V_2| > \cdots > |V_k|$.[2] With a smaller number of variables at progressively coarser levels, one can use more sophisticated but possibly slower refinement algorithms. However, it is still not sufficient to solve the whole problem until the coarsening reaches the coarsest level. As a result, at each level, the actual solution is produced by a refinement. Refinement is typically implemented with a decomposition method that uses a previous iteration or a coarser-level solution as an initialization. The multilevel algorithms rely heavily [101] on the quality of refinement solvers for small and local subproblems at all levels of coarseness. Thus, the most straightforward way to use NISQ devices in multilevel frameworks is to iteratively apply them as local solvers to refine a solution inherited from the coarse level. Because the refinement is executed at all levels of coarseness, *even a small improvement of a solution at the coarse level may cause a major improvement at the finest scale.* Typically, this is the most time-consuming stage of the multilevel solvers, which is expected to be fundamentally better if improved by NISQ devices.

Most refinement solvers in multilevel frameworks rely on fast but low-quality heuristics, rather than on the ability to compute an optimal solution. Moreover, in many existing solvers, the number of variables in such local subproblems is comparable with or smaller than the size of the problems that can be directly embedded on the NISQ devices (see examples in References [33, 53]), making them a perfect target for NISQ optimization algorithms. In most multilevel/multiscale/multigrid-based optimization solvers, a refinement consists of covering the domain (or all variables) with *small* subsets of variables (i.e., small subproblems) such that solving a small local problem on a subset improves the global solution for the current level.

Multilevel graph partitioning and community detection algorithms are examples of the most successful applications of multilevel algorithms for large graphs, achieving an excellent time/quality trade-off [16]. In this article, we deliberately use the simplest version of coarsening (to focus on the hybrid quantum-classical refinement) in which the edges of the fine-level graph

[2]Note that this does not necessarily imply $|E_0| > |E_1| > |E_2| > \cdots > |E_k|$.

are collapsed and create coarse-level vertices by merging the fine-level ones. There are several classes of refinement for both problems but in all of them, at each step a small subset of nodes (or even a singleton) is reassigned with partition (or cluster) that either better optimizes the objective or improves constraints. Some variants of stochastic extensions also exist.

2.5 On Scalability of Multilevel Methods

If we do not consider algorithmic frameworks with very limited space (such as streaming), the scalability of a correctly designed in-memory multilevel framework with its instance graphs is practically limited (or proportional) to the available memory size. Requirements to keep graphs in memory (not necessarily RAM) for multilevel partitioning and community detection are similar to those of matrices for multigrid [15], so the complexity is also comparable up to the factor of refinement. Theoretically, the multilevel and multigrid frameworks exhibit $O(|E|)$ or O(number of non-zeros in a matrix) complexity. However, for optimization on graphs, the refinement stage is typically computationally more expensive than that for the multigrid (e.g., compare Gauss-Seidel relaxation sweeps [54] and Kernighan-Lin or min-cut/max-flow refinement in graph partitioning [16]), because at each step of the refinement, an integer problem has to be solved. Refinement for the relaxed versions of integer problems (e.g., for the minimum 2-sum or bandwidth [77]) are usually faster but the quality suffers and in the end they should follow some rounding scheme for integer solution. However, even if the the refinement complexity is linear in the number of edges or corresponding matrix non-zeros, some overhead is typically introduced for the integer problems.

In this article, we deliberately use a simple coarsening that folds edges by merging pairs of nodes. The situation with the scalability of multilevel frameworks if high-order interpolation coarsening is involved (e.g., algebraic multigrid inspired weighted aggregation [76] when nodes can be split into several fractions, and different fractions form coarse nodes) is different. The high-order interpolation coarsening may result in increasing number of edges at several fine levels immediately implying increased running time. In such cases, the complexity can increase to $\max_{\text{level } i} O(E_i)$. Subsequently, a larger graph at each level requires more intensive refinement. In addition, the number of refinement calls required to achieve a very good solution strongly depends on the coarsening quality, which makes it difficult to get a complexity required for a nearly optimal solution. The only practical solution to that is artificially limiting the number of refinement calls (see all major solvers such as Metis, Jostle, Kahip, and Scotch reviewed in Reference [16]).

The criteria for limiting the number of refinement calls is never ideal. The refinement algorithms always heuristically decide what vertex or group of vertices should be optimized with respect to the current assignment of vertices to clusters (or parts) to make the current solution better. Typically, they take so called "boundary" nodes in all parts, i.e., those whose move from part to part can potentially improve the solution and optimize them or their groups. Therefore, the scalability of multilevel graph partitioning and clustering frameworks can be loosely described in terms of $|V|$ and $|E|$. Instead, it is better to describe it in terms of expected cut similar to the analysis in Reference [42]. To the best of our knowledge, there is no analysis that connects the size of the graph and performance of multilevel algorithms providing any practically useful theoretical bounds. Currently, the best way to describe the scalability of multilevel graph partitioning and clustering solvers is to use $O(|E|)$. However, the hidden constant in the linear complexity depends on the type of refinement. Thus, we anticipate that with the advent of reliable quantum hardware, one can expect a significant improvement in the running time and quality of the refinement in multilevel frameworks that will eliminate computationally expensive solvers to locally optimize groups of variables. We refer the reader to one of the most recent examples of multilevel scalability in Reference [22] (e.g., see Table 4) in which a graph with 3.8B edges was (suboptimally) solved in 255 seconds.

3 METHODS

An iterative improvement scheme is a common approach for solving large-scale problems with NISQ devices. Traditionally, this is done by formulating the entire problem in the Ising model or as a QUBO and then solving it using hybrid quantum-classical algorithms (see, for example, "qbsolv" from D-Wave systems [11]). These methods decompose the large QUBO into smaller sub-QUBOs or decrease the number of degrees of freedom to fit the subproblem on the hardware (for example, using a multilevel scheme) and iteratively improve the global solution by solving the small subproblems (sub-QUBOs). One of the main limitations of this approach is the size and density of the original QUBO. For example, in the graph partitioning formulation given by Equation (3), the term $\mathbb{v}\mathbb{v}^T$ leads to the formulation of a completely dense $n \times n$ QUBO matrix regardless of whether or not the original graph was sparse. Storing and processing this dense matrix can easily make this method prohibitively computationally expensive even for moderately sized problems. In our implementation of Quantum Local Search (QLS) [88], we circumvent this limitation by developing a novel subproblem formulation of the Graph Partitioning Problem and Modularity Maximization as a QUBO that does not require formulating the entire QUBO.

Another concern is the effectiveness of selection criteria of candidate variables (or nodes) to be included in each subproblem. A common metric used in selecting whether or not a variable is to be included in the subproblem is whether or not changing the variable value would reduce (increase) the objective value for a minimization (maximization) problem. Thus, since computing the change in objective value for a small change in the solution is performed multiple times, it is important to ensure that this computation is efficient. We derive a novel efficient way to compute the change in the objective value of the entire QUBO also without formulating the entire QUBO and thus provide an efficient refinement scheme using current NISQ devices.

We begin by introducing an efficient QUBO subproblem formulation for the Graph Partitioning Problem and the Community Detection Problem. Then, we present an efficient way to compute the gain and change in the objective of the entire QUBO. Finally, we put it all together and outline our algorithm.

3.1 QUBO Formulation for Subproblems

Let M be an $n \times n$ symmetric matrix that represents the QUBO for a large-scale problem such that it is prohibitively expensive to either generate or store M. However, for QLS, we need to generate constant-size sub-QUBOs of M, which in turn represent subproblems of the original problem. To generate a sub-QUBO, let k be the size of the desired sub-QUBO. In other words, the sub-QUBO will have k variables and $n - k$ *fixed variables* that remain invariant for this specific sub-QUBO. We refer to the k variables as *free variables*. Without loss of generality, let the first k variables of $\mathbf{s}$ be the free variables, then we write $\mathbf{s}$ as

$$\mathbf{s} = \begin{bmatrix} \mathbf{s}_v \\ \mathbf{s}_f \end{bmatrix},$$

where $\mathbf{s}_v$ represents the k free variable terms and $\mathbf{s}_f$ represents the $n - k$ fixed terms. In the next step, M can be represented using block form

$$M = \left[\begin{array}{c|c} M_{vv} & M_{vf} \\ \hline M_{vf}^T & M_{ff} \end{array}\right] \tag{11}$$

such that M_{vv} is a $k \times k$ matrix. Next, we can write $\mathbf{s}^T M\mathbf{s}$ as

$$\mathbf{s}^T M\mathbf{s} = \mathbf{s}_v^T M_{vv}\mathbf{s}_v + \mathbf{s}_v^T(2M_{vf}\mathbf{s}_f) + \mathbf{s}_f^T M_{ff}\mathbf{s}_f. \tag{12}$$

Since $\mathbf{s}_f$ are fixed values, we have $\mathbf{s}_f^T M_{ff}\mathbf{s}_f$ as a constant, thus

$$\min\ \mathbf{s}^T M\mathbf{s} = \min\ \mathbf{s}_v^T M_{vv}\mathbf{s}_v + \mathbf{s}_v^T(2M_{vf}\mathbf{s}_f). \tag{13}$$

From Equation (11), we have

$$\mathbb{v}\mathbb{v}^T = \left[\begin{array}{c|c} \mathbb{v}_v\mathbb{v}_v^T & \mathbb{v}_v\mathbb{v}_f^T \\ \hline \mathbb{v}_f\mathbb{v}_v^T & \mathbb{v}_f\mathbb{v}_f^T \end{array}\right]. \tag{14}$$

Therefore, from Equation (13), we have

$$\min\ \mathbf{s}^T\mathbb{v}\mathbb{v}^T\mathbf{s} = \min\ \mathbf{s}_v^T\mathbb{v}_v\mathbb{v}_v^T\mathbf{s}_v + 2\mathbf{s}_v^T\mathbb{v}_v\mathbb{v}_f^T\mathbf{s}_f. \tag{15}$$

The formulation in Equation (15) is particularly important, because it shows that the matrix $\mathbb{v}\mathbb{v}^T$ does not need to be explicitly created at each iteration during refinement. This is a crucial observation, because $\mathbb{v}\mathbb{v}^T$ is a completely dense matrix.

As described in Section 2.1, the Community Detection Problem is given by

$$\max\ \frac{1}{4|E|}\mathbf{s}^T\left(A - \frac{1}{2|E|}\mathbb{k}\mathbb{k}^T\right)\mathbf{s} \tag{16}$$

or

$$\min\ \mathbf{s}^T\left(\frac{1}{2|E|}\mathbb{k}\mathbb{k}^T - A\right)\mathbf{s} \tag{17}$$

and the Graph Partitioning Problem is given by

$$\min\ \mathbf{s}^T\left(\alpha\mathbb{v}\mathbb{v}^T - \beta A\right)\mathbf{s}. \tag{18}$$

In the above formulation, modularity clustering can be viewed as the Graph Partitioning Problem in a QUBO model, where the volume of a node is defined as the weighted degree and the penalty constant is $\frac{1}{|E|}$ Therefore, in both cases, we can perform a refinement while defining fixed values as

$$\min\ \mathbf{s}^T\left(\frac{1}{2|E|}\mathbb{k}\mathbb{k}^T - A\right)\mathbf{s} = \min\ \mathbf{s}_v^T\left(\frac{1}{2|E|}\mathbb{k}_v\mathbb{k}_v^T\right)\mathbf{s}_v + \mathbf{s}_v^T\left(\frac{1}{|E|}\mathbb{k}_v\mathbb{k}_f^T\right)\mathbf{s}_f - \mathbf{s}^T A\mathbf{s} \tag{19}$$

and

$$\min\ \mathbf{s}^T\left(\alpha\mathbb{v}\mathbb{v}^T - \beta A\right)\mathbf{s} = \min\ \mathbf{s}_v^T\left(\alpha\mathbb{v}_v\mathbb{v}_v^T\right)\mathbf{s}_v + \mathbf{s}_v^T\left(2\alpha\mathbb{v}_v\mathbb{v}_f^T\right)\mathbf{s}_f - \beta\mathbf{s}^T A\mathbf{s} \tag{20}$$

with

$$\min\ -\beta\mathbf{s}^T A\mathbf{s} = \min\ -\beta\mathbf{s}_v^T A_{vv}\mathbf{s}_v - \mathbf{s}_v^T(2\beta A_{vf}\mathbf{s}_f). \tag{21}$$

The formulation in Equations (19) and (20) are particularly important during the refinement step, because this implies that the complete dense (and therefore prohibitively large) QUBO or Ising model does not need to be created at each iteration. These formulations also demonstrate a close relationship between the Graph Partitioning Problem and the Community Detection Problem.

3.2 Efficient Evaluation of the Objective

To select the free variables for the subproblem, we need to be able to efficiently compute the change of the objective function by moving one node from one part to another. In other words, for each vertex v, we need to efficiently compute the *gain*, which is the decrease (or increase) in the edge-cut together with penalty if v is moved to the other part.

For a symmetric matrix M, the change in the value $Q = \mathbf{s}^T M \mathbf{s}$ by flipping a single variable s_i corresponding to the node i is given by

$$\Delta Q(i) = 2\left(\sum_{j \in C_1} M_{ij} - \sum_{j \in C_2} M_{ij}\right), \tag{22}$$

where C_1 and C_2 correspond to all variables with $s_i = -1$ and $s_i = 1$, respectively. Next, we define

$$deg(v, C) := \sum_{j \in C} A_{vj}; \quad Deg(C) := \sum_{i \in C} k_i; \quad Vol(C) := \sum_{i \in C} \mathbb{v}_i$$

then

$$2\left(\sum_{j \in C_1} A_{ij} - \sum_{j \in C_2} A_{ij}\right) = 2deg(v_i, C_1) - 2deg(v_i, C_2)$$

and finally

$$\begin{aligned} 2\left(\sum_{j \in C_1} (\mathbb{v}\mathbb{v}^T)_{ij} - \sum_{j \in C_2} (\mathbb{v}\mathbb{v}^T)_{ij}\right) &= 2\left(\mathbb{v}_i \sum_{j \in C_1, i \neq j} \mathbb{v}_j - \mathbb{v}_i \sum_{j \in C_2} \mathbb{v}_j\right) \\ &= 2\mathbb{v}_i(Vol(C_1 \backslash i) - Vol(C_2)), \end{aligned}$$

where we assume that $i \in C_1$. This expression can be computed in $O(1)$ time.

In the same way,

$$\begin{aligned} 2\left(\sum_{j \in C_1} (\mathbb{k}\mathbb{k}^T)_{ij} - \sum_{j \in C_2} (\mathbb{k}\mathbb{k}^T)_{ij}\right) &= 2\left(k_i \sum_{j \in C_1, i \neq j} k_j - k_i \sum_{j \in C_2} k_j\right) \\ &= 2k_i(Deg(C_1 \backslash i) - Deg(C_2)) \end{aligned}$$

can also be computed in $O(1)$ time given $Deg(C_1)$ and $Deg(C_2)$, where $Deg(C_i)$ represents the sum of weighted degrees of nodes in community i.

Therefore, the change in modularity is given by

$$\Delta Q(i) = \frac{k_i}{|E|}(Deg(C_1 \backslash i) - Deg(C_2)) - 2\Big(deg(v_i, C_1) - deg(v_i, C_2)\Big) \tag{23}$$

and change in edge-cut together with penalty value is given by

$$\Delta Q(i) = 2\alpha\mathbb{v}_i(Vol(C_1 \backslash i) - Vol(C_2))) - 2\beta\Big(deg(v_i, C_1) - deg(v_i, C_2)\Big). \tag{24}$$

For each node i, both expressions (23) and (24) can be computed in $O(k_i)$ time, where k_i is the unweighted degree of i.

At no point during the algorithm should the complete QUBO matrix be formulated. This also applies to the process of evaluating a given solution. In other words, evaluating the modularity for the Community Detection Problem or edge-cut together with penalty term for the Graph Partitioning Problem should be done in $O(1)$ time and space. The term is

$$\mathbf{s}^T \mathbb{v}\mathbb{v}^T \mathbf{s} = (Vol(C_1) - Vol(C_2))^2$$

where as

$$\mathbf{s}^T A\mathbf{s} = 2(|E| - 2cut).$$

Therefore,

$$\mathbf{s}^T(\alpha \mathbb{v}\mathbb{v}^T - \beta A)\mathbf{s} = \alpha(Vol(C_1) - Vol(C_2))^2 - 2\beta(|E| - 2cut) \tag{25}$$

and

$$\mathbf{s}^T\left(\frac{1}{2|E|}\mathbb{k}\mathbb{k}^T - A\right)\mathbf{s} = \frac{1}{2|E|}(Deg(C_1) - Deg(C_2))^2 - 2(|E| - 2cut), \tag{26}$$

where Equations (25) and (26) give the formulations for computing the modularity and edge-cut with corresponding penalty value, respectively, without creating the QUBO matrix.

3.3 Algorithm Overview

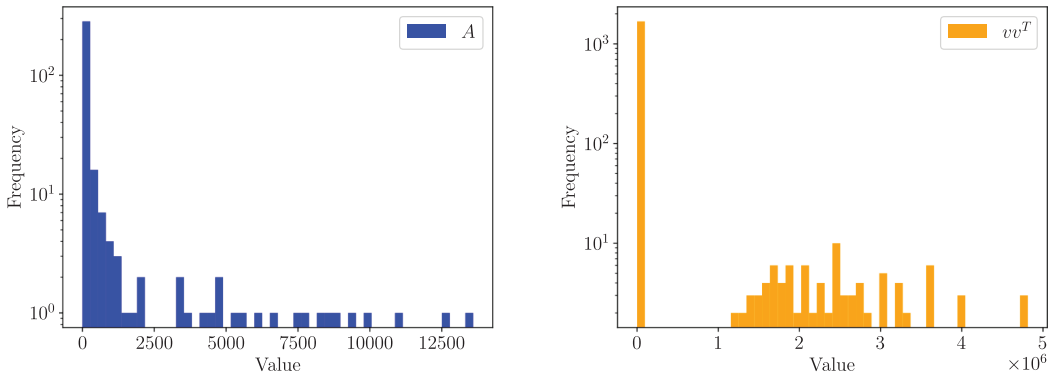
Now, we can combine the building blocks described in the previous two subsections. Let $G = (V, E)$ be the problem graph. ML-QLS begins by coarsening the problem graph. During the coarsening stage, for some integer k, a hierarchy of coarsened graphs $G = G_0, G_1, \ldots, G_k$ is constructed. In this work, we used the coarsening tools implemented in KaHIP Graph Partitioning package [80]. We used the coarsening implementation that is performed using maximum weight matching with "expansion^{*2}" metric as described in Reference [36]. The maximum edge matching is found using the Global Path Algorithm [36]. In the next step, a QUBO is formulated for the smallest graph G_k and solved on the quantum device. If $|V_k|$ is greater than the hardware size,[3] then QLS [88] with a random initialization is used to solve for G_k. Then, the solution is iteratively projected onto finer levels and refined using QLS. The algorithm overview is presented in Algorithm 1.

For the Graph Partitioning Problem, the initial weight of each node is one by definition, therefore coarsening of the nodes keeps the total node volume constant at each coarsening level. For the Community Detection Problem, the initial weight of each node is set to the degree of the node. This ensures that the size of the graph (total number of weighted edges) is also kept constant at each level. Note that Graph Partitioning is defined with respect to total node volume ($|V|$), while modularity is defined with respect to the size ($|E|$, the total number of weighted edges) of the graph.

3.4 Addressing the Limited Precision of the Hardware

One of the subproblem solvers we used in this work is Quantum Annealing, which we ran on the LANL D-Wave 2000Q machine. The D-Wave 2000Q is an analog quantum annealer with limited precision. In this work, we used a simple coarsening that constructs coarser graphs by aggregating nodes at a finer level to become a single node at the coarser level (i.e., many nodes on the finer level are merged into one node at the coarser level, with the volume of the new node set to be the sum of the volumes of the nodes on the coarser level). This causes the precision required to describe the node volumes and edge weights for coarser graphs to increase dramatically, especially for the large-scale problems. Thus, a QUBO describing the coarsest graph could require significantly more precision to represent compared to the finest graph. For example, in Graph Partitioning where the QUBO problem to be minimized is $A - \alpha \mathbb{v}\mathbb{v}^T$, the range of values in the matrix A increases at a different rate than the range of values in the matrix $\mathbb{v}\mathbb{v}^T$ during the coarsening process, increasing the precision required to describe the overall QUBO formed at each level (see an example in Figure 2(a)). Thus, if the QUBO $A - \alpha \mathbb{v}\mathbb{v}^T$ is directly scaled to accommodate the limited precision of the device, the quality of the results can suffer. In our experiments, we observe that directly scaling the QUBO returned feasible, but low-quality solutions. To overcome this challenge, for the problems solved on the D-Wave device, we first scaled the matrices A and $\alpha \mathbb{v}\mathbb{v}^T$ separately,

[3] More specifically, greater than the maximum number of variables in a problem that can be embedded on the device.



(a) Edge weights of the coarsest graph SSS12.

(b) Entries of the matrix $\mathbb{v}\mathbb{v}^T$.

Fig. 2. In Figure 2(a), the maximum value is approximately 13×10^3. In Figure 2(b), the maximum value is approximately 5×10^6 and minimum value 1. A naive scaling of QUBO matrix $A - \mathbb{v}\mathbb{v}^T$ can result in values that are too large to be handled by the quantum annealer due to its limited precision. Such values of A are ignored, leading to random balanced partitions.

ALGORITHM 1: Multilevel Quantum Local Search

```
function ML-QLS(G, problem_type)
    if problem_type is modularity then
        G = UpdateWeights(G)
    G_0, G_1, ..., G_k = KaHIPCoarsen(G)
    if |V_k| ≤HardwareSize then
        // solve directly
        QUBO = FormulateQUBO(G_k)
        solution = SolveSubproblem(QUBO)
    else
        // use QLS
        initial_solution = RandomSolution(G_k)
        solution = RefineSolution(G_k, initial_solution)
    for G_i in G_{k-1}, G_{k-2}, ..., G_0 do
        projected_solution = ProjectSolution(solution, G_i, G_{i+1})
        solution = RefineSolution(G_i, projected_solution)
    return solution
function RefineSolution(G_i, projected_solution)
    solution = projected_solution
    while not converged do
        ΔQ = ComputeGains(G_i, solution)
        X = HighestGainNodes(ΔQ)
        QUBO = FormulateQUBO(X)
        // using IBM UQC or D-Wave QA
        candidate = SolveSubproblem(QUBO)
        if candidate > solution then
            solution = candidate
    return solution
```

and then formed the QUBO to be optimized. This approach then resulted in achieving results with high-quality solutions on the D-Wave device.

4 EXPERIMENTS AND RESULTS

Implementation. The general framework for ML-QLS is implemented in Python 3.7 with NetworkX [32] for network operations. We have used the coarsening algorithms available in the KaHIP Graph Partitioning package [80] that are implemented in C++. The code for the general ML-QLS framework is available on GitHub [1].

Systems. The refinement algorithms presented in this work require access to NISQ devices capable of solving problems formulated in the Ising model. To this end, we have used the D-Wave 2000Q quantum annealer located at Los Alamos National Laboratory, as well as IBM's Poughkeepsie 20 qubit quantum computer available on the Oak Ridge National Laboratory IBM Q hub network together with the high-performance simulator, IBM Qiskit Aer Simulator [8]. However, our framework is modular and can easily be extended to utilize other novel quantum computing architectures as they become available.

The D-Wave 2000Q is the state-of-the-art quantum annealer at this time. It has up to 2,048 qubits that are laid out in a special graph structure known as a Chimera graph. The Chimera graph is sparse, thus the device has sparse connectivity. Fully connected graphs as dense problems need to be embedded onto the device, which leads to the maximum size of 64 variables. We have used the embedding algorithm described in Reference [12] to calculate a complete embedding of the 64-variable problem. We found this embedding only once and reused it during our experiments. We utilized D-Wave's Solver API (SAPI), which is implemented in Python 2.7, to interact with the system. The D-Wave system is intrinsically a stochastic system, where solutions are sampled from a distribution corresponding to the lowest energy state. For each subproblem, the best solution out of 10,000 samples is returned. The annealing time for each call to the D-Wave system was set to 20 microseconds.

To solve problems formulated in the Ising model on IBM's Poughkeepsie quantum computer and simulator, we implemented QAOA using the SBPLX [73] optimizer to find the optimal variational parameters. We allowed 2,000 iterations for SBPLX to find optimal parameters for QAOA. At each iteration, the circuit is executed 5,000 times (5,000 "shots") to obtain the statistic on the objective function. After the optimal parameters are found, the solution corresponding to best of 5,000 samples produced by running the ansatz with optimal parameters is returned. Due to the limitations of NISQ devices available in IBM Q hub network [86], we used the RYRZ variational form [3] (also known as a hardware-efficient ansatz) as the ansatz for our QAOA implementation. For the experiments run on IBM quantum device Poughkeepsie, we perform the variational parameter optimization on the simulator locally and run QAOA on the device via the IBM Q Experience cloud API. This is done due to the job queue limitations provided via the IBM Q Experience. However, we expect to be able to run QAOA variational parameter optimization fully on a device, as more devices are becoming available on the cloud. We have used GNU Parallel [93] for the large-scale numerical experiments performed on the quantum simulator.

Considering the fact that solutions from the NISQ devices and simulator do not provide optimality guarantees, we have also solved various subproblems formulated in the Ising model using the solver Gurobi [64] together with modeling package Pyomo [35]. The results using Gurobi as a solver for each subproblem are denoted as "Optimal" in our plots. Note that while each subproblem was solved and proven to be optimal for subproblem size 20, the same is not always true for subproblem size 64. For subproblem size 64, we occasionally observe non-zero gaps.

Table 1. Properties of the Networks Used to Evaluate ML-QLS

Network name	$\|V\|$	$\|E\|$	d_{avg}	d_{max}
SSS12	21,996	1,221,028	111.02	722
4elt	15,606	45,878	5.88	10
bcsstk30	28,924	1,007,284	69.65	218
cti	16,840	48,232	5.73	6
data	2,851	15,093	10.59	17
roadNet-PA-20k	20,000	26,935	2.69	7
opsahl-powergrid	4,941	6,594	2.67	19
msc23052	5,722	103,391	36.14	125
finan512-10k	10,000	28,098	5.62	54

d_{avg} is average degree, d_{max} is maximum degree.

Table 2. Properties of Synthetic Networks Used in the Modularity Evaluation

Network name	$\|V\|$	$\|E\|$	d_{avg}	d_{max}	γ	β	μ
GirvanNewman	10,000	75,000	15.0	15	1	1	0.1
lancichinetti1	10,000	76,133	15.22	50	2	1	0.1

d_{avg} is average degree, d_{max} is maximum degree, γ is the exponent for the degree distribution, β is the exponent for the community size distribution, and μ is the mixing parameter. For a detailed discussion of the parameters the reader is referred to Reference [52].

Instances. A summary of the graphs used in the experiments together with their properties is presented in Table 1. For the Graph Partitioning Problem, we evaluate ML-QLS on five graphs, four of which are drawn from the Graph Partitioning Archive [90] (4elt, bcsstk30, cti and data) and one from the set of hard to partition graphs (vsp_msc10848_300sep_100in_1Kout, denoted in figures as SSS12) [78]. For the Modularity Maximization Problem, we evaluate ML-QLS on six graphs. The graphs roadNet-PA-20k and opsahl-powergrid are real-world networks from the KONECT dataset [51]. Graphs msc23052 and finan512-10k are taken from the graph archive presented in Reference [79]. The graphs finan512-10k and roadNet-PA-20k are reduced to 10,000 and 20,000 nodes, respectively, by performing a breadth-first search from the median degree node. Note that due to the high diameter of these networks and their structure (portfolio optimization problem and road network), this preserves their structural properties. GirvanNewman is a synthetic graph generated using the model introduced by Girvan and Newman (GN) [30]. The graph lancichinetti1 is a synthetic graph generated using a generalization of the GN model that allows for heterogeneity in the distributions of node degree and community size, introduced by Lancichinetti et al. [52]. Table 2 shows the parameters used to generate the synthetic graphs.

Experimental Setup. Our experiments are performed to compare the solutions from ML-QLS with those of high-quality classical solvers and the best known results, if available. For the Graph Partitioning Problem, the results are compared to those produced by KaHIP [80], which is a state-of-the-art multilevel Graph Partitioning solver. The best known results are taken at the Graph Partitioning Archive [90] where applicable. To make our approach more comparable to KaHIP, we follow the user guide [4] and use the kaffpaE version of the solver with the option --mh_enable_kabapE for high quality refinement for perfectly balanced parts. We use the option --preconfiguration=fast to ensure results are compared with a single V-cycle. Our results (cut values) are normalized with either the best known value when applicable or by the smallest cut value found by any of the solvers used.

Table 3. Highest Modularity Value Found by All Methods for a Given Problem

Network name	Best modularity
finan512-10k	0.499
GirvanNewman	0.459
lancichinetti1	0.452
msc23052	0.499
opsahl-powergrid	0.497
roadNet-PA-20k	0.499

The highest possible modularity value for at most two communities is 0.5.

For the Modularity Maximization Problem, we compare our solutions using ML-QLS with two classical clustering methods, Asynchronous Fluid Communities [67] (implemented in NetworkX [32]) and Spectral Clustering [89, 99] (implemented in Scikit-learn [68]). Note that even though these methods solve the same problem (namely, Community Detection or clustering), they do not explicitly maximize modularity. Therefore, it is unfair to directly compare the modularity of the solution produced by them to ML-QLS, which is explicitly maximizing modularity. However, they provide a useful baseline. Moreover, since the maximum possible modularity for at most 2 communities is 0.5, the best solutions found by all methods are no more than 1%–10% away from the optimal (see Table 3).

The experimental results are presented in Figure 3. We have made all raw result data available on Github [2]. For each problem and method (except for QAOA on IBM Q Poughkeepsie quantum computer, labeled "QAOA (IBMQ Poughkeepsie)" in Figure 3), we perform 10 runs of a single V-cycle with different seeds. For "QAOA (IBMQ Poughkeepsie)," we perform just one run per each problem due to the limited access to quantum hardware.

Observations. We observe that ML-QLS is capable of achieving results close to the best ones found by other solvers for all problems. For Graph Partitioning, Figure 3 shows significant variability in the quality of the solution across different solvers and problem instances. This effect is also observed for the state-of-the-art Graph Partitioning solver KaHIP when run for a single V-cycle. This is partially due to the fact that we normalize the objectives to make them directly comparable. For example, for the graph `4elt` the best known cut value presented in the Graph Partitioning Archive [90] is 139. Therefore, an *absolute* difference of 28 edges in cut obtained by a solver translates into a 20% *relative* difference presented in Figure 3. However, the same *absolute* difference of 28 edges would translate into $\approx 0.44\%$ for the graph `bcsstk30` (best known cut 6394). The graph `SSS12` is specifically designed to be hard for traditional Graph Partitioning frameworks [78]. This explains the high variation in the performance of KaHIP on it.

It is worth noting that QAOA on the IBM quantum computers (see "QAOA (IBM Q Poughkeepsie)" in Figure 3) takes more iterations to converge to a solution compared to D-Wave. This is partially due to the fact that we perform the QAOA variational parameter optimization on the simulator and only run once with the optimized parameters on the device. As a result, the learned variational parameters do not include the noise profile of the device, limiting the quality of subproblem solutions. As devices become more easily available, we expect to be able to run full variational parameter optimization on the quantum hardware.

To project the performance improvements for future hardware, we simulate the performance of ML-QLS as a function of hardware (subproblem) size shown in Figure 4. As the subproblem size

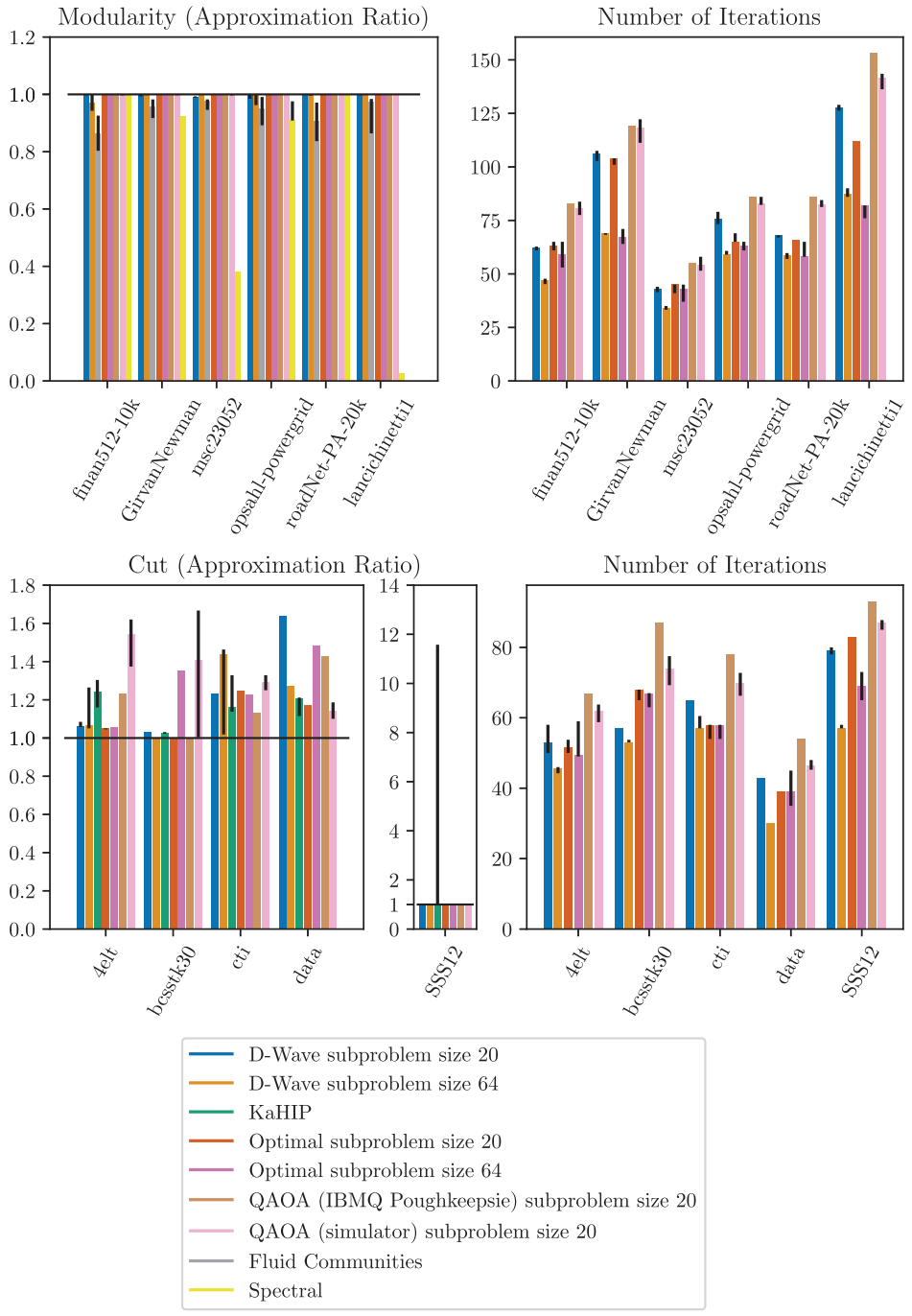
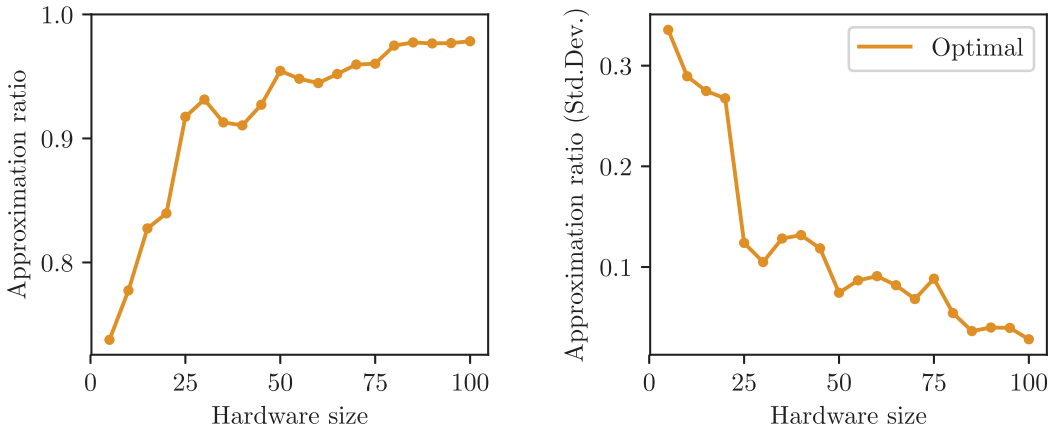


Fig. 3. Quality of the solution and the number of iterations for all problems and solvers. The height of the bars is the median over 10 seeds. Error bars (black) are 25th and 75th percentiles. For the objective function (Cut or Modularity) all results are normalized by the best solution found by any solver (for Graph Partitioning this includes the best known cuts from the Graph Partitioning Archive [90]). Number of iterations is the number of calls to the subproblem solver (ML-QLS only).



(a) Mean

(b) Standard deviation

Fig. 4. Modularity (Approximation ratio) as the function of the size of the subproblem (hardware size). The performance is projected using Gurobi as the subproblem solver. Figure 4(a) presents the mean approximation ratio averaged over the entire benchmark. Figure 4(b) presents the standard deviation. As the hardware size increases, the quality of the solution found by ML-QLS improves.

increases, the average quality of the solution found by ML-QLS improves and variation in results decreases. This shows that performance of ML-QLS can be improved as larger size quantum devices and better quantum optimization routines are developed. Note that this makes the assumption that the subproblem is solved to optimality or to a solution that is close to optimal. Evaluating the scaling of quantum optimization algorithms used for solving the subproblems falls outside of the scope of this article (we provide an overview of relevant recent results in Section 2.3).

4.1 Scaling and Running Time Estimates

The proposed ML-QLS approach is based on the traditional multilevel methods for graph partitioning and graph clustering and therefore a lot of scaling and running time considerations are shared between the two family of methods. Concretely, in our implementation, we use the coarsening available in KaHIP graph partitioning package [80], making the running time and the scaling of the *coarsening* stage of our method and KaHIP equal. The running time of solving the problem on the coarsest level does not scale with problem size, as the size of the coarsest level is fixed to be equal to the hardware size. Therefore, in this section, we will focus on the analysis of the *refinement* stage.

To evaluate how the proposed approach scales with the problem size, we construct a series of graphs with the number of nodes ranging from 1,000 to 128,000. The graphs are constructed in the same way as the graph `roadNet-PA-20k`, namely, by performing a breadth-first search from the median degree node of `roadNet-PA` [50] and including nodes until the desired size is reached. We fix the subproblem size to 20 for Gurobi and to 64 for D-Wave. The results are presented in Figure 5. We note that the number of iterations scales roughly logarithmically with the problem size, as we observe roughly constant number of iterations per uncoarsening level. As we do not constrain the number of refinement iterations, this indicates that the solution projected from coarser levels is of high quality and does not need to be significantly changed at the refinement stage, indicating that the coarsening has successfully constructed a multilevel hierarchy that preserves problem structure. The success of coarsening in this example might be due to the simplicity of the problem

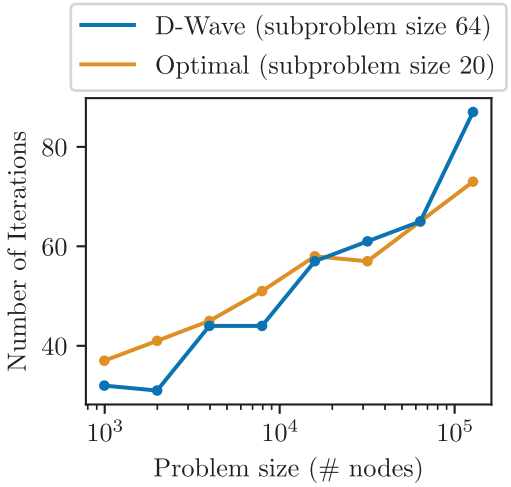


Fig. 5. The number of iterations (calls to optimizer) to solution for the modularity maximization problem as a function of problem size. The performance is projected using Gurobi as the subproblem solver with subproblem size 20 and allowing it to solve each subproblem to optimality, as well as using D-Wave as the subproblem solver with subproblem size 64. For Gurobi, the subproblem size was limited to 20 to guarantee that each subproblem is proven to be solved optimally.

structure (planar graph of a road network). However, in all major multilevel solvers the number of refinement calls is artificially limited, preserving the scaling (see Section 2.5 for an in-depth discussion).

Due to the high overhead of queuing system and remote execution on cloud quantum computers, the actual observed running time in our experiments is not representative of the algorithm performance. To give the reader a sense of the running time of our approach, we use the number of iterations to solution presented in Figures 3 and 5 to compute rough estimates of the running time of the algorithm. To do that, we briefly discuss the running time of QAOA and quantum annealing for our problems. We observe that as the running time of coarsening is approximately 1 second for the problems in our benchmark, the overall running time is dominated by refinement. Same is true of this class of multilevel algorithms in general (see Section 2.5).

To understand the running time of QAOA, we need to understand two of its components: the "training time" (time to find good QAOA parameters) and the "execution time" (time to sample from QPU with optimal parameters). In this work, we train our variational quantum optimizer (which can be considered a version of QAOA as described in Section 2.2) purely classically in simulation. This results in running time of $\approx$ 300 seconds for 20 qubits and 2,000 iterations of outer-loop optimization using quantum simulator Qiskit Aer and these runtimes can be improved using other more efficient state-vector simulators [103]. However, this is approach becomes unrealistic as problem size grows. While novel tensor-network based approaches allow purely classical training of certain class of QAOA instances [91], they are not applicable to the problems considered in this article due to the full connectivity of the problem. Therefore, there are currently no viable approaches to classically training QAOA for larger modularity and graph partitioning problems. Running time for larger problems can be estimated by taking the "execution time" and multiplying it by number of iterations of outer-loop optimization. At the time of writing, IBM Quantum Experience did not allow us to directly measure the execution time of the quantum circuit with optimal parameters. Therefore, we use the running time reports available in the literature, from which we

estimate the walltime of $\approx 1s$ for 5,000 shots [9, 41]. Combined with the purely classical training time reported above, for a problem that takes 60 iterations to solve (e.g., modularity maximization for `4elt`), this gives projected running time of ≈ 30 minutes. The actual running time observed in experiment has been larger due to the additional overhead introduced by the queue system (up to hours per iteration, depending on queue state).

For Quantum Annealing on the D-Wave system, the overall running time spent on the device is directly proportional to the number of iterations of the algorithm as reported in Figure 5. For a fixed annealing time, and number of samples requested, the running time on the device is approximately constant per iteration. For an annealing time of 20 microseconds, together with 1,000 samples, in our experiments, we observed an overall running time of approximately 5 seconds for the 15 iterations when applied to the graph of 1,000, while taking about 15 seconds when applied to the graph of approximately 127,000 nodes.

5 OPEN PROBLEMS AND DISCUSSION

Revising (un)coarsening operators in anticipation of the new class of high-quality refinement solvers is the first major open problem. The majority of multilevel algorithms for combinatorial optimization problems are inspired by the idea of "thinking globally while acting locally." However, there is a crucial difference between these algorithms for combinatorial problems and such methods as multigrid for continuous problems or multiscale PDE-based optimization. In multigrid (e.g., for systems of linear equations), a relaxation at the uncoarsening stage is convergent [14], and in most cases assumes an optimal solution (up to some tolerance) for a subset of relaxed variables given other variables are invariant (i.e., a fixed solution for those variables that are not in the optimized subset). Examples include easily parallelizable Jacobi relaxation, as well as hard-to-parallelize Gauss-Seidel relaxation, in which most variables are typically optimized sequentially, and many more. Both coarsening and uncoarsening operators (also known as the restriction and prolongation in multigrid) assume this convergence, which in the end provides guarantees for the entire multilevel framework. However, for the *combinatorial* multilevel solvers, the integer variables make this assumption practically impossible, even for subproblems containing tens of variables optimized simultaneously. With the development of less noisy quantum devices, we can assume that in our hands will be extremely fast heuristics to produce nearly (if hypothetically not completely) optimal solutions for combinatorial optimization problems of up to several hundreds of variables. To use the multilevel paradigm correctly, there will be a critical need to revise (un)coarsening operators that take this feature into account, because (to the best of our knowledge) all existing versions of coarsening operators do not consider optimality of the refinement. Moreover, most existing multilevel frameworks exhibit more emphasis on computational speedup rather than on the quality of the solution to better approximate the fine problem.

The second problem is not unique to multilevel methods but to most decomposition-based approaches. Even if quantum devices become fully developed and become more accessible for the broad scientific community, they will still remain more expensive than regular CPU-based devices. The decomposition approaches split the problem into many small local subproblems, while multilevel methods may need even more of them, because solving subproblems is required at all levels of coarseness. Thus, there is a critical need in developing an extremely fast routing classifier for a subproblem that will decide whether solving a particular subproblem on the NISQ device will be beneficial in comparison to the CPU.

The third consideration is the sparsity of the problem. The methods outlined in this work are evaluated on sparse problems, and we expect them to perform well only under the assumption on sparsity of the problem. We make this assumption because the standard benchmarks for the problems we consider are sparse (see, for example, the Graph Partitioning Archive [90] and DI-

MACS Graph Partitioning and Graph Clustering Challenge [10, 81]). Development of decomposition methods specifically targeting dense problems is an interesting future direction. Typically, in existing multilevel solvers, the fact of density at some level serves as a stopping criterion, i.e., when the problem is originally dense, the multilevel hierarchy construction is terminated, which is clearly an insufficient thing to do for many distributions of accumulated edge weights. In other words, this problem exists not only for multilevel quantum but for classical solvers as well. In multilevel algorithms, the problem density requires a different approach such as sparsification [39].

6 CONCLUSION

Current Noisy Intermediate-Scale Quantum (NISQ) devices are limited in the number of qubits and can therefore only be used to directly solve combinatorial optimization problems that exhibit a limited number of variables. To overcome this limitation, in this work, we have proposed the multilevel computational framework for solving large-scale combinatorial problems on NISQ devices. We demonstrate this approach on two well-known combinatorial optimization problems, the Graph Partitioning Problem, and the Community Detection Problem, and perform experiments on the 20-qubit IBM gate-model quantum computer, and the 2,048-qubit D-Wave 2000Q quantum annealer. To implement an efficient iterative refinement scheme using the NISQ devices, we have developed novel techniques for efficiently formulating and evaluating sub-QUBOs without explicitly constructing the entire QUBO of the large-scale problem, which in many cases can be a dense matrix that makes it computationally expensive to store and process. In our experiments, for the Graph Partitioning Problem, five graphs were chosen such that the smallest graph had 2,851 nodes while the largest had 28,924 nodes, while for the Community Detection Problem, the smallest graph had 4,941 nodes and largest had 10,000 nodes. For both problems, for comparison purposes, we ran one V-cycle of the multilevel framework with the different NISQ devices multiple times and compared the results to the state-of-art methods. Our experimental results give comparable results to the state-of-the-art methods and, for some cases, we were able to get the best-known results. This work therefore provides an important stepping stone to demonstrating practical Quantum Advantage. As the capabilities of NISQ devices increase, we are hopeful that similar methods can provide a path to adoption of quantum computers for a variety of business [24] and scientific applications.

APPENDIX

A D-WAVE TIMING RESULTS

For Quantum Annealing on D-Wave, each iteration of the refinement process requires the execution of a single Quantum Machine Instruction (QMI), which includes the QUBO parameters and annealing-cycle parameters sent to the D-Wave system for processing. The total time accessing the QPU (QPU Access time) for a single QMI can be broken down into four parts. The first is the one-time initialization step to program the QPU performed at every independent QPU call. Then for each sample requested, we have the following times: The annealing time, followed by the time needed to wait for the QPU to regain its initial temperature, referred to as the QPU Delay Time, and last the time needed to read the sample from the QPU referred to as the QPU Readout Time. In other words,

$$qpu\_access\_time = qpu\_programming\_time + N * (annealing\_time + delay\_time + readout\_time), \tag{27}$$

where N is the number of samples requested in a single QMI. The timing $N * (annealing\_time + delay\_time + readout\_time)$ is referred to as the QPU Sampling time. In our experiments, we request 1,000 samples for each QMI and fixed the annealing time to $20\mu s$ per sample. The annealing time as a user parameter can be either increased or reduced. However significantly reducing this parameter could greatly affect the QPU's ability to find lower energy states, thus affecting the

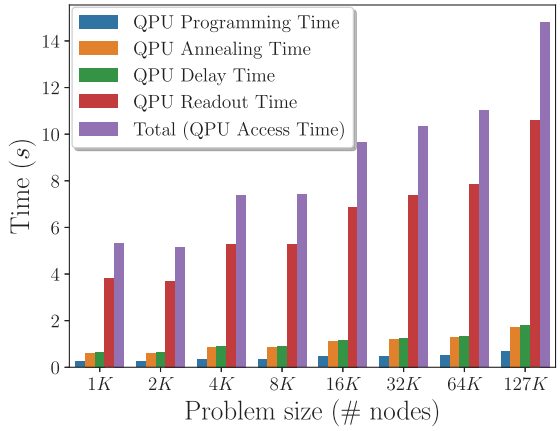


Fig. 6. D-Wave timing results showing a breakdown of the total time accessing the QPU as the problem size respresented by the number of nodes of the graph increases.

number of iterations of refinement algorithm and subsequently the entire outcome of the method. To test and demonstrate scalability of the proposed approach on the D-Wave system, similar to the experiments presented in Figure 5, we varied the number of nodes for a road network graph from approximately 10^3 nodes up to 10^5 nodes and computed the maximum modularity of each graph. Similar to the case in Figure 5 where each QUBO was solved up to optimality, we observe a logarithmic scaling in the number of calls to the D-Wave system (number of iterations) as shown in Figure 5. Figure 6 gives a breakdown of the total time (Total QPU Access time) for all iterations to compute the modularity for a given input graph where the smallest graph with approximately 1,000 nodes required a QPU Access time of 5 seconds, while the largest graph with approximately 127,000 nodes required a QPU Access time of approximately 15 seconds. However, the majority of the QPU Access time was actually in the QPU Readout time. Requiring less than 2 seconds of total annealing time for 1,000 samples is an impressive result for a graph of approximately 127,000 nodes. This time could be significantly trivially reduced by requesting a smaller number of samples per iteration, however, this may affect the solution quality. Note that the QPU Access time is a linear function in the number of iterations of the refinement method. Due to the heuristic nature of the proposed approach, the total number of iterations by itself is not a strictly increasing function in the number of nodes of the input graph. Thus, for example, we observe in Figure 5, the number of iterations is not a strictly increasing function and the graphs with $2K$ and $8K$ nodes required a smaller number of iterations than the graphs with $1K$ and $4K$ nodes, respectively. This is subsequently observed in the running times as shown in Figure 6.

ACKNOWLEDGMENTS

The authors thank anonymous referees whose valuable comments helped to improve this work.

REFERENCES

[1] [n.d.]. Retrieved from https://github.com/rsln-s/ml_qls.
[2] [n.d.]. Retrieved from https://github.com/rsln-s/ml_qls/tree/bc376276ba684460aeccaa371b4fc38003139e34/multilevel/data/results_csv.
[3] [n.d.]. IBM QISKit Aqua: Variational forms. Retrieved from https://github.com/Qiskit/qiskit-aqua/blob/master/qiskit/aqua/components/variational_forms/ryrz.py.
[4] [n.d.]. KaHIP v2.10 – Karlsruhe High Quality Partitioning User Guide. Retrieved from http://algo2.iti.kit.edu/schulz/software_releases/kahipv2.10.pdf.

[5] [n.d.]. Quantum Enhanced Optimization (QEO). Retrieved from https://www.iarpa.gov/index.php/research-programs/qeo.
[6] Tameem Albash and Daniel A. Lidar. 2018. Adiabatic quantum computation. *Rev. Mod. Phy.* 90, 1 (2018), 015002.
[7] Tameem Albash and Daniel A. Lidar. 2018. Demonstration of a scaling advantage for a quantum annealer over simulated annealing. *Phys. Rev. X* 8, 3 (July 2018). DOI: https://doi.org/10.1103/physrevx.8.031016
[8] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, Francisco Jose Cabrera-Hernádez, Jorge Carballo-Franquis, Adrian Chen, Chun-Fu Chen, Jerry M. Chow, Antonio D. Córcoles-Gonzales, Abigail J. Cross, Andrew Cross, Juan Cruz-Benito, Chris Culver, Salvador De La Puente González, Enrique De La Torre, Delton Ding, Eugene Dumitrescu, Ivan Duran, Pieter Eendebak, Mark Everitt, Ismael Faro Sertage, Albert Frisch, Andreas Fuhrer, Jay Gambetta, Borja Godoy Gago, Juan Gomez-Mosquera, Donny Greenberg, Ikko Hamamura, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Hiroshi Horii, Shaohan Hu, Takashi Imamichi, Toshinari Itoko, Ali Javadi-Abhari, Naoki Kanazawa, Anton Karazeev, Kevin Krsulich, Peng Liu, Yang Luh, Yunho Maeng, Manoel Marques, Francisco Jose Martín-Fernández, Douglas T. McClure, David McKay, Srujan Meesala, Antonio Mezzacapo, Nikolaj Moll, Diego Moreda Rodríguez, Giacomo Nannicini, Paul Nation, Pauline Ollitrault, Lee James O'Riordan, Hanhee Paik, Jesús Pérez, Anna Phan, Marco Pistoia, Viktor Prutyanov, Max Reuter, Julia Rice, Abdón Rodríguez Davila, Raymond Harry Putra Rudy, Mingi Ryu, Ninad Sathaye, Chris Schnabel, Eddie Schoute, Kanav Setia, Yunong Shi, Adenilton Silva, Yukio Siraichi, Seyon Sivarajah, John A. Smolin, Mathias Soeken, Hitomi Takahashi, Ivano Tavernelli, Charles Taylor, Pete Taylour, Kenso Trabing, Matthew Treinish, Wes Turner, Desiree Vogt-Lee, Christophe Vuillot, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, Christopher Wood, Stephen Wood, Stefan Wörner, Ismail Yunus Akhalwaya, and Christa Zoufal. 2019. Qiskit: An Open-source Framework for Quantum Computing. DOI: https://doi.org/10.5281/zenodo.2562110
[9] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (Oct. 2019), 505–510. DOI: https://doi.org/10.1038/s41586-019-1666-5
[10] David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner. 2013. *Graph Partitioning and Graph Clustering*. Vol. 588. American Mathematical Society, Providence, RI.
[11] Michael Booth, Steven P. Reinhardt, and Aidan Roy. 2017. Partitioning optimization problems for hybrid classical. *Quantum Execution. Technical Report.* 14-1006A-A D-Wave Technical Report Series. D-Wave Systems Inc. https://www.dwavesys.com/sites/default/files/partitioning_QUBOs_for_quantum_acceleration-2.pdf.
[12] Tomas Boothby, Andrew D. King, and Aidan Roy. 2016. Fast clique minor generation in Chimera qubit connectivity graphs. *Quant. Inf. Proc.* 15, 1 (2016), 495–508.
[13] Fernando G. S. L. Brandao, Michael Broughton, Edward Farhi, Sam Gutmann, and Hartmut Neven. 2018. For fixed control parameters the quantum approximate optimization algorithm's objective function value concentrates for typical instances. *arXiv:1812.04170* (2018).
[14] A. Brandt. 2002. Multiscale scientific computation: Review 2001. In *Multiscale and Multiresolution Methods. LNCSE*, Vol. 20. Springer, 3–95. Retrieved from http://dx.doi.org/10.1007/978-3-642-56205-1_1.
[15] A. Brandt and D. Ron. 2003. Multigrid solvers and multilevel optimization strategies. In *Multilevel Optimization and VLSICAD*, J. Cong and J. R. Shinnerl (Eds.). Kluwer.
[16] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. 2016. Recent advances in graph partitioning. In *Algorithm Engineering: Selected Results and Surveys. LNCS*, Vol. 9220. Springer-Verlag, 117–158.
[17] M. Cerezo, Akira Sone, Tyler Volkoff, Lukasz Cincio, and Patrick J. Coles. 2020. Cost-Function-Dependent Barren Plateaus in Shallow Quantum Neural Networks. arXiv:2001.00550
[18] J. Cong and J. R. Shinnerl (Eds.). 2003. *Multilevel Optimization and VLSICAD*. Kluwer.
[19] Gavin E. Crooks. 2018. Performance of the quantum approximate optimization algorithm on the maximum cut problem. *arXiv:1811.08419* (2018).
[20] D-Wave Systems Inc.2018. Introduction to the D-wave quantum hardware. Retrieved from www.dwavesys.com/tutorials/background-reading-series/introduction-d-wave-quantum-hardware.

[21] D-Wave Systems Inc. 2019. Measuring computation time on D-wave systems. *D-Wave User Manual 09-1107A-M* (2019). Retrieved from https://docs.dwavesys.com/docs/latest/doc_timing.html.

[22] Timothy A. Davis, William W. Hager, Scott P. Kolodziej, and S. Nuri Yeralan. 2019. Algorithm 1003: Mongoose, a graph coarsening and partitioning library. *ACM Trans. Math. Software* 46, 1 (2019).

[23] Vasil S. Denchev, Sergio Boixo, Sergei V. Isakov, Nan Ding, Ryan Babbush, Vadim Smelyanskiy, John Martinis, and Hartmut Neven. 2016. What is the computational value of finite-range tunneling? *Phys. Rev. X* 6, 3 (Aug. 2016). DOI: https://doi.org/10.1103/physrevx.6.031015

[24] Yongcheng Ding, Lucas Lamata, José D. Martín-Guerrero, Enrique Lizaso, Samuel Mugel, Xi Chen, Román Orús, Enrique Solano, and Mikel Sanz. 2019. Towards Prediction of Financial Crashes with a D-Wave Quantum Computer. arXiv:1904.05808

[25] Alexander Elgart and George A. Hagedorn. 2012. A note on the switching adiabatic theorem. *J. Math. Phys.* 53, 10 (Oct. 2012), 102202. DOI: https://doi.org/10.1063/1.4748968

[26] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, Joshua Lapan, Andrew Lundgren, and Daniel Preda. 2001. A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem. *Science* 292, 5516 (2001), 472–475.

[27] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. 2000. Quantum Computation by Adiabatic Evolution. arXiv:quant-ph/0001106

[28] Artur Garcia-Saez and Jordi Riu. 2019. Quantum observables for continuous control of the quantum approximate optimization algorithm via reinforcement learning. *arXiv preprint arXiv:1911.09682* (2019).

[29] E. Gelman and J. Mandel. 1990. On multilevel iterative methods for optimization problems. *Math. Prog.* 48, 1–3 (1990), 1–17.

[30] M. Girvan and M. E. J. Newman. 2002. Community structure in social and biological networks. *Proc. Nat. Acad. Sci.* 99, 12 (June 2002), 7821–7826. DOI: https://doi.org/10.1073/pnas.122653799

[31] Serge Gratton, Annick Sartenaer, and Philippe L. Toint. 2008. Recursive trust-region methods for multiscale nonlinear optimization. *SIAM J. Optim.* 19, 1 (2008), 414–444.

[32] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy'08)*, Gaël Varoquaux, Travis Vaught, and Jarrod Millman (Eds.). Pasadena, CA, 11–15.

[33] William W. Hager, James T. Hungerford, and Ilya Safro. 2018. A multilevel bilinear programming algorithm for the vertex separator problem. *Comput. Optim. Applic.* 69, 1 (2018), 189–223.

[34] Ryan Hamerly, Takahiro Inagaki, Peter L. McMahon, Davide Venturelli, Alireza Marandi, Tatsuhiro Onodera, Edwin Ng, Carsten Langrock, Kensuke Inaba, Toshimori Honjo, Koji Enbutsu, Takeshi Umeki, Ryoichi Kasahara, Shoko Utsunomiya, Satishi Kako, Ken-ichi Kawarabayashi, Robert L. Byer, Martin M. Fejer, Hideo Mabuchi, Dirk Englund, Eleanor Rieffel, Hiroki Takesue, and Yoshihisa Yamamoto. 2019. Experimental investigation of performance differences between coherent Ising machines and a quantum annealer. *Sci. Adv.* 5, 5 (2019), eaau0823.

[35] William E. Hart, Carl D. Laird, Jean-Paul Watson, David L. Woodruff, Gabriel A. Hackebeil, Bethany L. Nicholson, and John D. Siirola. 2017. *Pyomo-optimization Modeling in Python*. Vol. 67. Springer.

[36] Manuel Holtgrewe, Peter Sanders, and Christian Schulz. 2010. Engineering a scalable high quality graph partitioner. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS'10)*. IEEE. DOI: https://doi.org/10.1109/ipdps.2010.5470485

[37] Reiner Horst, Panos M. Pardalos, and Nguyen Van Thoai. 2000. *Introduction to Global Optimization*. Springer Science & Business Media.

[38] Cupjin Huang, Mario Szegedy, Fang Zhang, Xun Gao, Jianxin Chen, and Yaoyun Shi. 2019. Alibaba Cloud Quantum Development Platform: Applications to Quantum Algorithm Design. arXiv:arXiv:1909.02559

[39] Emmanuel John and Ilya Safro. 2016. Single-and multi-level network sparsification by algebraic distance. *J. Complex Netw.* 5, 3 (2016), 352–388.

[40] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M. Chow, and Jay M. Gambetta. 2017. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature* 549, 7671 (2017), 242.

[41] Peter J. Karalekas, Nikolas A. Tezak, Eric C. Peterson, Colm A. Ryan, Marcus P. da Silva, and Robert S. Smith. 2020. A quantum-classical cloud platform optimized for variational hybrid algorithms. *Quant. Sci. Technol.* 5, 2 (Apr. 2020), 024003. DOI: https://doi.org/10.1088/2058-9565/ab7559

[42] G. Karypis and V. Kumar. 1995. *Analysis of Multilevel Graph Partitioning*. Technical Report TR-95-037. Computer Science Dept., Univ. of Minnesota, Minneapolis, MN.

[43] G. Karypis and V. Kumar. 1999. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* 20, 1 (1999).

[44] Tosio Kato. 1950. On the adiabatic theorem of quantum mechanics. *J. Phys. Soc. Japan* 5, 6 (Nov. 1950), 435–439. DOI: https://doi.org/10.1143/jpsj.5.435

[45] Helmut G. Katzgraber, Firas Hamze, Zheng Zhu, Andrew J. Ochoa, and H. Munoz-Bauza. 2015. Seeking quantum speedup through spin glasses: The good, the bad, and the ugly. *Phys. Rev. X* 5, 3 (Sept. 2015). DOI: https://doi.org/10.1103/physrevx.5.031026

[46] Carl T. Kelley. 1999. *Iterative Methods for Optimization*. SIAM.

[47] Sami Khairy, Ruslan Shaydulin, Lukasz Cincio, Yuri Alexeev, and Prasanna Balaprakash. 2019. Learning to optimize variational quantum circuits to solve combinatorial problems. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI'19)*.

[48] Andrew D. King, Jack Raymond, Trevor Lanting, Sergei V. Isakov, Masoud Mohseni, Gabriel Poulin-Lamarre, Sara Ejtemaee, William Bernoudy, Isil Ozfidan, Anatoly Y. Smirnov, Mauricio Reis, Fabio Altomare, Michael Babcock, Catia Baron, Andrew J. Berkley, Kelly Boothby, Paul I. Bunyk, Holly Christiani, Colin Enderud, Bram Evert, Richard Harris, Emile Hoskinson, Shuiyuan Huang, Kais Jooya, Ali Khodabandelou, Nicolas Ladizinsky, Ryan Li, P. Aaron Lott, Allison J. R. MacDonald, Danica Marsden, Gaelen Marsden, Teresa Medina, Reza Molavi, Richard Neufeld, Mana Norouzpour, Travis Oh, Igor Pavlov, Ilya Perminov, Thomas Prescott, Chris Rich, Yuki Sato, Benjamin Sheldan, George Sterling, Loren J. Swenson, Nicholas Tsai, Mark H. Volkmann, Jed D. Whittaker, Warren Wilkinson, Jason Yao, Hartmut Neven, Jeremy P. Hilton, Eric Ladizinsky, Mark W. Johnson, and Mohammad H. Amin. 2019. Scaling advantage in quantum simulation of geometrically frustrated magnets. *arXiv preprint arXiv:1911.03446* (2019).

[49] James King, Sheir Yarkoni, Jack Raymond, Isil Ozfidan, Andrew D. King, Mayssam Mohammadi Nevisi, Jeremy P. Hilton, and Catherine C. McGeoch. 2019. Quantum annealing amid local ruggedness and global frustration. *J. Phys. Soc. Japan* 88, 6 (June 2019), 061007. DOI: https://doi.org/10.7566/jpsj.88.061007

[50] Jérôme Kunegis. 2013. KONECT—The Koblenz network collection. In *Proceedings of the Web Observatory Workshop*. 1343–1350.

[51] Jérôme Kunegis. 2013. KONECT: The Koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*. ACM, 1343–1350.

[52] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. 2008. Benchmark graphs for testing community detection algorithms. *Phys. Rev. E* 78, 4 (Oct. 2008). DOI: https://doi.org/10.1103/physreve.78.046110

[53] Sven Leyffer and Ilya Safro. 2013. Fast response to infection spread and cyber attacks on large-scale networks. *J. Complex Netw.* 1, 2 (2013), 183–199.

[54] Oren E. Livne and Achi Brandt. 2012. Lean algebraic multigrid (LAMG): Fast graph Laplacian linear solver. *SIAM J. Sci. Comput.* 34, 4 (2012), B499–B522.

[55] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. 2018. Barren plateaus in quantum neural network training landscapes. *Nat. Commun.* 9, 1 (Nov. 2018). DOI: https://doi.org/10.1038/s41467-018-07090-4

[56] Catherine C. McGeoch. 2019. Principles and guidelines for quantum performance analysis. In *Proceedings of the Conference on Quantum Technology and Optimization Problems. (QTOP'19). LNCS, Vol.* 411413, 36–47.

[57] Athanasios Migdalas, Panos M. Pardalos, and Peter Värbrand. 2013. *Multilevel Optimization: Algorithms and Applications*. Vol. 20. Springer Science & Business Media.

[58] K. G. Murty and S. N. Kabadi. 1987. Some NP-complete problems in quadratic and linear programming. *Math. Prog.* 39 (1987), 117–129.

[59] Giacomo Nannicini. 2019. Performance of hybrid quantum-classical variational heuristics for combinatorial optimization. *Phys. Rev. E* 99, 1 (Jan. 2019). DOI: https://doi.org/10.1103/physreve.99.013304

[60] John Napp, Rolando L. La Placa, Alexander M. Dalzell, Fernando G. S. L. Brandao, and Aram W. Harrow. 2019. Efficient classical simulation of random shallow 2D quantum circuits. arXiv:arXiv:2001.00021

[61] Christian F. A. Negre, Hayato Ushijima-Mwesigwa, and Susan M. Mniszewski. 2020. Detecting multiple communities using quantum annealing on the D-Wave system. *PLoS One* 15 (02 2020), 1–14. https://doi.org/10.1371/journal.pone.0227538.

[62] M. E. J. Newman. 2006. Modularity and community structure in networks. *Proc. Nat. Acad. Sci.* 103 (2006), 8577. https://doi.org/10.1073/pnas.0601602103.

[63] Sergey Novikov, Robert Hinkey, Steven Disseler, James I. Basham, Tameem Albash, Andrew Risinger, David Ferguson, Daniel A. Lidar, and Kenneth M. Zick. 2018. Exploring more-coherent quantum annealing. *arXiv:1809.04485* (2018).

[64] Gurobi Optimization. 2014. 'Gurobi optimizer reference manual. Retrieved from http://www.gurobi.com.

[65] J. S. Otterbach, R. Manenti, N. Alidoust, A. Bestwick, M. Block, B. Bloom, S. Caldwell, N. Didier, E. Schuyler Fried, S. Hong et al. 2017. Unsupervised Machine Learning on a Hybrid Quantum Computer. *arXiv preprint arXiv:1712.05771* (2017).

[66] G. Pagano, A. Bapat, P. Becker, K. S. Collins, A. De, P. W. Hess, H. B. Kaplan, A. Kyprianidis, W. L. Tan, C. Baldwin et al. 2019. Quantum approximate optimization with a trapped-ion quantum simulator. *arXiv:1906.02700* (2019).

[67] Ferran Parés, Dario Garcia Gasulla, Armand Vilalta, Jonatan Moreno, Eduard Ayguadé, Jesús Labarta, Ulises Cortés, and Toyotaro Suzumura. 2017. Fluid communities: A competitive, scalable and diverse community detection algorithm. In *Proceedings of the International Conference on Complex Networks and their Applications*. Springer, 229–240.

[68] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* 12 (2011), 2825–2830.

[69] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'brien. 2014. A variational eigenvalue solver on a photonic quantum processor. *Nat. Commun.* 5 (2014), 4213.

[70] Hannes Pichler, Sheng-Tao Wang, Leo Zhou, Soonwon Choi, and Mikhail D. Lukin. 2018. Quantum optimization for maximum independent set using Rydberg atom arrays. *preprint arXiv:1808.10816* (2018).

[71] Dorit Ron, Ilya Safro, and Achi Brandt. 2010. A fast multigrid algorithm for energy minimization under planar density constraints. *SIAM Multisc. Model. Simul.* 8, 5 (2010), 1599–1620.

[72] Dorit Ron, Ilya Safro, and Achi Brandt. 2011. Relaxation-based coarsening and multiscale graph organization. *Multisc. Model. Simul.* 9, 1 (2011), 407–423.

[73] Thomas Harvey Rowan. 1990. *Functional Stability Analysis of Numerical Algorithms*. Ph.D. Dissertation. University of Texas at Austin.

[74] Ehsan Sadrfaridpour, Talayeh Razzaghi, and Ilya Safro. 2019. Engineering fast multilevel support vector machines. *Mach. Learn.* (2019), 1–39.

[75] Ehsan Sadrfaridpour, Jeereddy Sandeep, Ken Kennedy, Andre Luckow, Talayeh Razzaghi, and Ilya Safro. 2017. Algebraic multigrid support vector machines. In *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN'17)*. Retrieved from https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2017-37.pdf.

[76] Ilya Safro, Dorit Ron, and Achi Brandt. 2006. Graph minimum linear arrangement by multilevel weighted edge contractions. *J. Algor.* 60, 1 (2006), 24–41.

[77] Ilya Safro, Dorit Ron, and Achi Brandt. 2008. Multilevel algorithms for linear ordering problems. *ACM J. Exper. Algor.* 13 (2008).

[78] E. Sadrfaridpour, T. Razzaghi, and I. Safro. 2019. Engineering fast multilevel support vector machines. *Mach. Learn.* 108 (2019), 1879–1917. https://doi.org/10.1007/s10994-019-05800-7

[79] Ilya Safro and Boris Temkin. 2011. Multiscale approach for the network compression-friendly ordering. *J. Disc. Algor.* 9, 2 (2011), 190–202.

[80] Peter Sanders and Christian Schulz. 2013. Think locally, act globally: Highly balanced graph partitioning. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*. LNCS, Vol. 7933. Springer, 164–175.

[81] D. A. Bader, A. Kappes, H. Meyerhenke, P. Sanders, C. Schulz, and D. Wagner. 2018. Benchmarking for graph clustering and partitioning. In *Encyclopedia of Social Network Analysis and Mining*, R. Alhajj and J. Rokne (Eds.). Springer, New York, NY. https://doi.org/10.1007/978-1-4939-7131-2_23

[82] Ruslan Shaydulin and Yuri Alexeev. 2019. Evaluating quantum approximate optimization algorithm: A case study. In *Proceedings of the 10th International Green and Sustainable Computing Conference (IGSC'19)*. IEEE. DOI: https://doi.org/10.1109/igsc48788.2019.8957201

[83] Ruslan Shaydulin, Jie Chen, and Ilya Safro. 2019. Relaxation-based coarsening for multilevel hypergraph partitioning. *SIAM Multisc. Model. Simul.* 17, 1 (2019), 482–506.

[84] Ruslan Shaydulin and Ilya Safro. 2018. Aggregative coarsening for multilevel hypergraph partitioning. In *Proceedings of the 17th International Symposium on Experimental Algorithms (SEA'18). (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 103, Gianlorenzo D'Angelo (Ed.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2:1–2:15. DOI: https://doi.org/10.4230/LIPIcs.SEA.2018.2

[85] Ruslan Shaydulin, Ilya Safro, and Jeffrey Larson. 2019. Multistart methods for quantum approximate optimization. In *Proceedings of the IEEE High Performance Extreme Computing Conference (HPEC'19)*. IEEE. DOI: https://doi.org/10.1109/hpec.2019.8916288

[86] Ruslan Shaydulin, Hayato Ushijima-Mwesigwa, Christian F. A. Negre, Ilya Safro, Susan M. Mniszewski, and Yuri Alexeev. 2019. A hybrid approach for solving optimization problems on small quantum computers. *Computer* 52, 6 (June 2019), 18–26. DOI: https://doi.org/10.1109/mc.2019.2908942

[87] Ruslan Shaydulin, Hayato Ushijima-Mwesigwa, Ilya Safro, Susan Mniszewski, and Yuri Alexeev. 2018. Community detection across emerging quantum architectures. In *Proceedings of the 3rd International Workshop on Post Moore's Era Supercomputing*.

[88] Ruslan Shaydulin, Hayato Ushijima-Mwesigwa, Ilya Safro, Susan Mniszewski, and Yuri Alexeev. 2019. Network community detection on small quantum computers. *Adv. Quant. Technol.* DOI: https://doi.org/10.1002/qute.201900029
[89] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, 8 (2000), 888–905. DOI: 10.1109/34.868688
[90] A. J. Soper, C. Walshaw, and M. Cross. 2004. A combined evolutionary search and multilevel optimisation approach to graph-partitioning. *J. Global Optim.* 29, 2 (June 2004), 225–241. DOI: https://doi.org/10.1023/b:jogo.0000042115.44455.f3
[91] Michael Streif and Martin Leib. 2020. Training the quantum approximate optimization algorithm without access to a quantum processing unit. *Quant. Sci. Technol.* 5, 3 (May 2020), 034008. DOI: https://doi.org/10.1088/2058-9565/ab8c2b
[92] Mario Szegedy. 2019. What do QAOA energies reveal about graphs? *arXiv:1912.12277*.
[93] Ole Tange. 2018. *GNU Parallel 2018*. DOI: https://doi.org/10.5281/zenodo.1146014
[94] Jason P. Terry, Prosper D. Akrobotu, Christian F. A. Negre, and Susan M. Mniszewski. 2020. Quantum isomer search. *PLoS One* 15, 1 (2020), e0226787.
[95] Hayato Ushijima-Mwesigwa, Christian F. A. Negre, and Susan M. Mniszewski. 2017. Graph partitioning using quantum annealing on the D-Wave system. In *Proceedings of the 2nd International Workshop on Post Moores Era Supercomputing*. ACM, 22–29.
[96] Hayato M. Ushijima-Mwesigwa, Christian F. A. Negre, Susan M. Mniszewski, and Ilya Safro. 2018. Multilevel quantum annealing for graph partitioning. In *Proceedings of the Qubits 2018 D-Wave Users Conference*.
[97] Wim van Dam, Michele Mosca, and Umesh Vazirani. 2002. How powerful is adiabatic quantum computation? In *Proceedings of the 42nd Symposium on Foundations of Computer Science*. 279–287. DOI: https://doi.org/10.1109/SFCS.2001.959902
[98] Guillaume Verdon, Michael Broughton, Jarrod R. McClean, Kevin J. Sung, Ryan Babbush, Zhang Jiang, Hartmut Neven, and Masoud Mohseni. 2019. Learning to learn with quantum neural networks via classical neural networks. *arXiv:1907.05415*.
[99] Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Statist. Comput.* 17, 4 (2007), 395–416.
[100] Stefan Voß, Silvano Martello, Ibrahim H. Osman, and Catherine Roucairol. 2012. *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Springer Science & Business Media.
[101] C. Walshaw. 2004. Multilevel refinement for combinatorial optimisation problems. *Ann. Oper. Res.* 131 (2004), 325–372.
[102] Zhihui Wang, Stuart Hadfield, Zhang Jiang, and Eleanor G. Rieffel. 2018. Quantum approximate optimization algorithm for MaxCut: A fermionic view. *Phys. Rev. A* 97, 2 (2018), 022304.
[103] Xin-Chuan Wu, Sheng Di, Emma Maitreyee Dasgupta, Franck Cappello, Hal Finkel, Yuri Alexeev, and Frederic T. Chong. 2019. Full-state quantum circuit simulation by using data compression. In *Proceedings of the High Performance Computing, Networking, Storage and Analysis International Conference (SC'19)*. IEEE Computer Society, Denver, CO.
[104] M.-H. Yung, J. Casanova, A. Mezzacapo, J. McClean, L. Lamata, A. Aspuru-Guzik, and E. Solano. 2014. From transistor to trapped-ion computers for quantum chemistry. *Sci. Rep.* 4, 1 (Jan. 2014). DOI: https://doi.org/10.1038/srep03589
[105] Leo Zhou, Sheng-Tao Wang, Soonwon Choi, Hannes Pichler, and Mikhail D. Lukin. 2018. Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices. *arXiv:1812.01041*.

Received February 2020; revised August 2020; accepted September 2020