# From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz

Stuart Hadfield[*], Zhihui Wang[+,**], Bryan O'Gorman[+,†,‡], Eleanor G. Rieffel[+],
Davide Venturelli[+,**], Rupak Biswas[+]

[*] Department of Computer Science, Columbia University, New York, NY

[+] Quantum Artificial Intelligence Lab., NASA Ames Research Center, Moffett Field, CA

[**] USRA Research Institute for Advanced Computer Science (RIACS), Mountain View, CA

[†] Stinger Ghaffarian Technologies, Inc., Greenbelt, MD

[‡] Berkeley Quantum Information and Computation Center and Departments of Chemistry and Computer Science,
University of California, Berkeley, CA

September 12, 2017

The next few years will be exciting as prototype universal quantum processors emerge, enabling implementation of a wider variety of algorithms. Of particular interest are quantum heuristics, which require experimentation on quantum hardware for their evaluation, and which have the potential to significantly expand the breadth of applications for which quantum computers have an established advantage. A leading candidate is Farhi et al.'s Quantum Approximate Optimization Algorithm, which alternates between applying a cost-function-based Hamiltonian and a mixing Hamiltonian. Here, we extend this framework to allow alternation between more general families of operators. The essence of this extension, the Quantum Alternating Operator Ansatz, is the consideration of general parameterized families of unitaries rather than only those corresponding to the time-evolution under a fixed local Hamiltonian for a time specified by the parameter. This ansatz supports the representation of a larger, and potentially more useful, set of states than the original formulation, with potential long-term impact on a broad array of application areas.

For cases that call for mixing only within a desired subspace, refocusing on unitaries rather than Hamiltonians enables more efficiently implementable mixers than was possible in the original framework. Such mixers are particularly useful for optimization problems with hard constraints that must always be satisfied, defining a feasible subspace, and soft constraints whose violation we wish to minimize. More efficient implementation enables earlier experimental exploration of an alternating operator approach, in the spirit of the Quantum Approximate Optimization Algorithm, to a wide variety of approximate optimization, exact optimization, and sampling problems. In addition to introducing the Quantum Alternating Operator Ansatz, we lay out design criteria for mixing operators, detail mappings for eight problems, and provide a compendium with brief descriptions of mappings for a diverse array of problems.

# 1  Introduction

Over the last few decades, researchers have discovered several stunning instances of quantum algorithms that provably outperform the best existing classical algorithms and, in some cases, the best possible classical algorithm [64]. For most problems, however, it is currently unknown whether quantum computing can provide an advantage, and if so, how to design quantum algorithms that realize such advantages. Today, challenging computational problems arising in the practical world are frequently tackled by heuristic algorithms, which by definition have not been analytically proven to be the best approach, or even proven analytically to outperform the best approach of the previous year. Rather, these algorithms are empirically shown to be effective, by running them on characteristic sets of problems. As prototype quantum hardware emerges, we can begin to apply this heuristic approach to quantum heuristic algorithms.

For several years now, special-purpose quantum hardware has been used to explore one quantum heuristic algorithm, quantum annealing. Emerging gate-model processors will enable investigation of a much broader array of quantum heuristics beyond quantum annealing. Within the last year, IBM has made available publicly through the cloud a gate-model chip with five superconducting qubits [40], and announced recently an upgrade to a 16-qubit chip. Likewise, Google [10] and Rigetti Computing [68] anticipate providing processors with 40–100 superconducting qubits within a year or two [55]. Many academic groups, including at TU Delft and at UC Berkeley, have similar efforts. Beside superconducting architectures, ion [17] and neutral atom based [66] devices are also reaching the scale at which intermediate-size experiments would be feasible [56]. Gate-model quantum computing expands the empirical evaluation of quantum heuristics applications beyond optimization of classical functions, as well as enabling a broader array of approaches to optimization [77].

While limited exploration of quantum heuristics beyond quantum annealing has been possible through small-scale classical simulation, the exponential overhead in such simulations has limited their usefulness. The next decade will see a blossoming of quantum heuristics as a broader and more flexible array of quantum computational hardware becomes available. The immediate question is: what experiments should we prioritize that will give us insight into quantum heuristics? One leading candidate is the Quantum Approximate Optimization Algorithm (QAOA), for which a number of tantalizing related results have been obtained [21, 23, 41, 71, 73–75] since Farhi et al.'s initial paper [22]. In QAOA, a phase-separation operator, usually the problem Hamiltonian that encodes the cost function of the optimization problem, and a mixing Hamiltonian are applied in alternation. The class $QAOA_p$ consists of level-$p$ QAOA circuits, in which there are $p$ iterations of applying a classical Hamiltonian (derived from the cost function) and a mixing Hamiltonian. The $2p$ parameters of the algorithm specify the durations for which each of these two Hamiltonians are applied.

Prior work suggests the power and flexibility of QAOA circuits. Farhi et al. [23] exhibited a $QAOA_1$ algorithm that beat the existing best approximation bound for efficient classical algorithms for the problem E3Lin2, only to inspire a better classical algorithm [5]. Jiang et al. [41] demonstrated that the class of QAOA circuits is powerful enough to obtain the $\Theta(\sqrt{2^n})$ query complexity on Grover's problem, and also provides the first algorithm within the QAOA framework to show a quantum advantage for a finite number of iterations greater than two. Farhi and Harrow [21] proved that, under reasonable complexity assumptions, the output distribution of even $QAOA_1$ circuits cannot be efficiently sampled classically. Yang et al. [75] proved that for evolution under a Hamiltonian that is

the weighted sum of Hamiltonian terms, with the weights allowed to vary in time, the optimal control is (essentially always) bang-bang, i.e. constant magnitude, of either the maximum or minimum allowed weight, for each of the terms in the Hamiltonian at any given time. Their work implies that QAOA circuits with the right parameters are optimal among Hamiltonians of the form $H(s) = (1 - f(s))H_B + f(s)H_C$, where $f(s)$ is a real function in the range $[0, 1]$. It remains an open question whether QAOA provides a quantum advantage over classical algorithms for approximate optimization.

QAOA circuits were first proposed by Farhi et al. [22] as the basis for a quantum approximate optimization algorithm (QAOA). Since Farhi et al.'s original work, QAOA circuits have also been applied for both exact optimization [41, 74] and sampling [21]. Here, we formally describe a Quantum Alternating Operator Ansatz (QAOA), extending the approach of Farhi et al. [22] to allow alternation between more general families of operators. The essence of this extension is the consideration of general parameterized families of unitaries rather than only those corresponding to the time-evolution of a fixed local Hamiltonian for a time specified by the parameter. This ansatz supports the representation of a larger, and potentially more useful, set of states than the original formulation. For cases that call for mixing only within a desired subspace, refocusing on unitaries rather than Hamiltonians enables more efficiently implementable mixers than was possible in the original framework. Such mixers are particularly useful for optimization problems with hard constraints that must always be satisfied, defining a feasible subspace, and soft constraints whose violation we wish to minimize. More efficient implementation enables earlier experimental exploration of an alternating operator approach, in the spirit of the Quantum Approximate Optimization Algorithm, to a wide variety of approximate optimization, exact optimization, and sampling problems.

We carefully construct a framework for this ansatz, laying out design criteria for families of mixing operators. We then detail QAOA mappings of several optimization problems, and provide a compendium of mappings for a diverse array of problems. These mapping range from the relatively simple, which could be implemented on near term devices, to complex mappings with significant resource requirements. This paper is meant as a starting point for a research program. Improved mappings and compilations, especially for some of the more complex problems, are a promising area for future work. Architectural codesign could be used to enable experimentation of QAOA approaches to some problems earlier than would be possible otherwise.

We reworked the original acronym so that "QAOA" continues to apply to both prior work and future work to be done in this more general framework. More generally, the reworked acronym refers to a set of states representable in a certain form, and so can be used without confusion in contexts other than approximate optimization, e.g. exact optimization and sampling. (Incidentally, this reworking also removes the redundancy from the now commonly used phrase "QAOA algorithm".)

Here, after describing the framework for the ansatz, we map a number of problems, designing phase-separation and mixing operators appropriate for each problem. We comment here on the relation between these mappings and those for quantum annealing (QA). Because current quantum annealers have a fixed driver (the mixing Hamiltonian in the QA setting), all problem dependence must be captured in the cost Hamiltonian on such devices. The general strategy is to incorporate the hard constraints as penalty terms in the cost function, and then convert the cost function to a cost Hamiltonian [9, 52, 65]. But this approach means that the algorithm must search a much larger space than were the evolution confined to feasible configurations, making the search less efficient than were it possible to constrain the evolution. This issue, and other drawbacks, led Hen & Spedalieri [39] and

Hen & Sarandy [38] to suggest a different approach for adiabatic quantum optimization (AQO) in which the standard driver is replaced by an alternative driver that confines the evolution to the feasible subspace. Their approach resembles a restricted class, H-QAOA (defined below), of QAOA algorithms. Our H-QAOA mappings of graph coloring, graph partitioning, and not-all-equal 3SAT are close to those in [38, 39]. While QAOA mappings are different from quantum annealing mappings, with most of the design effort going into the mixing operator rather than the cost-function-based phase separator, QAOA algorithms, like QA and AQO, but unlike most other quantum algorithms, are relatively easy for people familiar with classical computer science but not quantum computing to design, as we illustrate in this paper.

In Sec. 2, we carefully construct a framework for this ansatz, laying out design criteria for families of mixing operators. Sec. 3 and Sec. 4 detail QAOA mappings and compilations for several optimization problems, illustrating design techniques and a variety of mixers. Sec. 3 considers four problems in which the configuration space of interest is strings: MaxIndependentSet, Max-$k$-ColorableSubgraph, Max-$k$-ColorableInducedSubgraph, and MinGraphColoring. Sec. 4 considers four problems in which the configuration space of interest is orderings: MinTravelingSalesperson, and three versions of single machine scheduling (SMS), also called job sequencing. App. A provide a compendium of mappings and compilations for a diverse array of problems, and provides resource estimates for their implementation. Sec. 5 concludes with a discussion of many open questions and directions for future work.

## 2 The Quantum Alternating Operator Ansatz (QAOA)

Here, we formally describe the Quantum Alternating Operator Ansatz (QAOA), extending the approach of Farhi et al. [22]. QAOA, in our sense, encompasses a more general class of quantum states that may be algorithmically accessible and useful. We focus here on the application of QAOA to approximate optimization, though it may also be used in exact optimization [41, 74] and sampling [21].

An instance of an *optimization problem* is a pair $(F, f)$, where $F$ is the *domain* (set of feasible points) and $f : F \to \mathbb{R}$ is the *objective function* to be optimized (minimized or maximized). Let $\mathcal{F}$ be the Hilbert space of dimension $|F|$, whose standard basis we take to be $\{|\mathbf{x}\rangle : \mathbf{x} \in F\}$. Generalizing Ref. [22], a QAOA circuit is characterized by two parameterized families of operators on $\mathcal{F}$:

- a family of *phase-separation operators* $U_\mathrm{P}(\gamma)$ that depends on the *objective function* $f$ and

- a family of *mixing operators* $U_\mathrm{M}(\beta)$ that depends on the domain and its structure,

where $\beta$ and $\gamma$ are real parameters. Specifically, a QAOA$_p$ circuit consists of $p$ alternating applications of operators from these two families:

$$Q_p(\boldsymbol{\beta}, \boldsymbol{\gamma}) = U_\mathrm{M}(\beta_p)U_\mathrm{P}(\gamma_p) \cdots U_\mathrm{M}(\beta_1)U_\mathrm{P}(\gamma_1). \tag{1}$$

This Quantum Alternating Operator Ansatz (QAOA) consists of the states representable as the application of such a circuit to a suitably simple initial state $|s\rangle$:

$$|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle = Q_p(\boldsymbol{\beta}, \boldsymbol{\gamma}) |s\rangle. \tag{2}$$

For a given optimization problem, a *QAOA mapping* of a problem consists of a family of phase-separation operators, a family of mixing operators, and a starting state. The

circuits for the quantum approximate optimization algorithm fit within this paradigm, with unitaries of the form $e^{-i\gamma H_P}$ and $e^{-i\beta H_M}$, with parameters $\gamma$ and $\beta$ indicating the time for which a fixed Hamiltonian is applied.

The domain usually will be expressed as the feasible subset of a larger *configuration space*, specified by a set of problem *constraints*. For implementation on expected near-term quantum hardware, each configuration space will need to be *encoded* into a subspace of a Hilbert space of a multiqubit system, with the domain corresponding to as a *feasible subspace* of the configuration space. For each domain, there are many possible mixing operators. As we will see, using more general one-parameter families of unitaries enables more efficiently implementable mixers that preserve the feasible subspace. Given a domain, an encoding of its configuration space, a phase separator, and a mixer, there are a variety of *compilations* of the phase separator and mixer to circuits that act on qubits.

For any function $f$, not just an objective function, we define $H_f$ to be the quantum Hamiltonian that acts as $f$ on the basis states:

$$H_f \ket{\mathbf{x}} = f(\mathbf{x}) \ket{\mathbf{x}}. \tag{3}$$

In prior work, the domain $F$ is the set of all $n$-bit strings, $U_P(\gamma) = e^{-i\gamma H_f}$, and $U_M(\beta) = e^{-i\gamma H_B}$. Furthermore, with just one exception, the mixing Hamiltonian was $H_B = \sum_{j=1}^{n} X_j$. We use the notation $X_j$, $Y_j$, $Z_j$ to indicate the Pauli matrices $X$, $Y$, $Z$ acting on the $j$th qubit. The corresponding parameterized unitaries are denoted by $X_j(\theta) = e^{-i\theta X_j}$ and similarly for $Y_j$ and $Z_j$. The one exception is Sec. VIII of [21], which discusses a variant for the problem of maximum independent set, in which $F$ is the set of bitstrings corresponding to independent sets, the phase separator depends on the cost function as above, and the mixing operator is $U_M(\beta) = e^{-i\gamma H_B}$, where $H_B$ is such that

$$\bra{\mathbf{x}} H_B \ket{\mathbf{y}} = \begin{cases} 1, & \mathbf{x}, \mathbf{y} \in F \text{ and } \mathrm{Ham}(\mathbf{x}, \mathbf{y}) = 1, \\ 0, & \text{otherwise,} \end{cases} \tag{4}$$

which connects feasible states with unit Hamming distance Ham. Sec. VIII of [21] does not discuss the implementability of $U_M(\beta)$.

We extend and formalize the approach of Sec. VIII of [21] with an eye to implementability, both in the short and long terms. We also build on a theory developed for adiabatic quantum optimization (AQO) by Hen and Spedalieri [39] and Hen and Sarandy [38], though the gate-model setting of QAOA leads to different implementation considerations than those for AQO. For example, Hen et al. identified Hamiltonians of the form $H_M = \sum_{j,k} H_{j,k}$, where $H_{j,k} = X_j X_k + Y_j Y_k$, as useful in the AQO setting for a variety of optimization problems with hard and soft constraints; such mixers restrict the mixing to the feasible subspace defined by the hard constraints. Analogously, the unitary $U_M = e^{-i\beta H_M}$ meets our criteria, discussed in Sec. 2.1, for good mixing for a variety of optimization problems including those considered in [38, 39]. Since $H_{j,k}$ and $H_{i,l}$ do not commute when $|\{j,k\} \cap \{i,l\}| = 1$, compiling $U_M$ to two-qubit gates is non-trivial. One could Trotterize, but it may be more efficient and just as effective to use an alternative mixing operator such as $U_M = e^{-i\beta H_{S_r}} \cdots e^{-i\beta H_{S_1}}$ where the pairs of qubits have been partitioned into $r$ subsets $\{S_i\}_i$ containing only disjoint pairs, motivating in part our more general ansatz.

We define as "Hamiltonian-based QAOA" (H-QAOA) the class of QAOA circuits in which both the phase separator family $U_P(\gamma) = e^{-i\gamma H_P}$ and the mixing operator family $U_M(\beta) = e^{-i\beta H_M}$ correspond to time-evolution under some Hamiltonians $H_P$ and $H_M$, respectively. (In this work, we consider only phase separators $U_P(\gamma) = \sum_{\mathbf{x}} e^{-i\gamma g(\mathbf{x})} \ket{\mathbf{x}}\bra{\mathbf{x}}$ that correspond to classical functions and thus also correspond to time-evolution under

some (potentially non-local) Hamiltonian, though more general types of phase separators may be considered.) We further define "Local-Hamiltonian-based QAOA" (LH-QAOA) as the subclass of H-QAOA in which the Hamiltonian $H_{\mathrm{M}}$ is a sum of (polynomially many) local terms.

Before discussing design criteria, we briefly mention that there are obvious generalizations in which $U_{\mathrm{P}}$ and $U_{\mathrm{M}}$ are taken from families parameterized by more than a single parameter. For example, in [24], a different parameter for every term in the Hamiltonian is considered. In this paper, we will only consider the case of one-dimensional families, given that it is a sufficiently rich area of study, with the task of finding good parameters $\gamma_1, \ldots, \gamma_p$ and $\beta_1, \ldots, \beta_p$ already challenging enough due to the curse of dimensionality [73]. A larger parameter space may support more effective circuits, but increases the difficulty of finding such circuits by opening up the design space and making the parameter setting more difficult.

## 2.1 Design Criteria

Here, we briefly specify design criteria for the three components of a QAOA mapping of a problem. We expect that as exploration of QAOA proceeds, these design criteria will be strengthened, and will depend on the context in which the ansatz is used. For example, when the aim is a polynomial-time quantum circuit, the component should have more stringent bounds on their complexity; without such bounds, the ansatz is not useful as a model for a strict subset of states producible via polynomially-sized quantum circuits. On the other hand, when the computation is expected to grow exponentially, a simple polynomial bound on the depth of these operators might be reasonable. One example might be for exact optimization of the problems considered here; for these problems the worst case algorithmic complexity is exponential, but it is worth exploring whether QAOA might out-perform classical heuristics in expanding the tractable range for some problems.

**Initial State.** We require that the initial state $|s\rangle$ be *trivial* to implement, by which we mean that it can be created by a constant-depth (in the size of the problem) quantum circuit from the $|0 \ldots 0\rangle$ state. Here, we often take as our initial state a single feasible solution, usually implementable by a depth-1 circuit consisting of single-qubit bit-flip operations $X$. Because in such a case the initial phase operator only applies a global phase, we may want to consider the algorithm as starting with a single-mixing operator $U_{\mathrm{M}}(\beta_0)$ to the initial state as a first step. In the quantum approximate optimization algorithm, the standard starting state $|+ \cdots +\rangle$ is obtained by a depth-1 circuit apply Hadamards $H$ to each of the qubits in the $|0 \ldots 0\rangle$ state.

This criterion could be relaxed to logarithmic depth if needed. It should not be relaxed too much: relaxing the criterion to polynomial depth would obviate the usefulness of the ansatz as a model for a strict subset of states producible via polynomially-sized quantum circuits. Algorithms with more complex initial states should be considered hybrid algorithms, with an initialization part and a QAOA part. Such algorithms are of interest in cases when one expect the computation to grow exponentially, such as is the case for exact optimization for many of the problems here, but might still out-perform classical heuristics in expanding the tractable range.

**Phase-separation unitaries.** We require the family of phase separation operators to be diagonal in the computational basis. In almost all cases, we take $U_{\mathrm{P}}(\gamma) = e^{-i\gamma H_f}$.

**Mixing unitaries (or "mixers").** We require the family of mixing operators $U_{\mathrm{M}}(\beta)$ to

- preserve the feasible subspace: for all values of the parameter $\beta$ the resulting unitary takes feasible states to feasible states, and

- provide transitions between all pairs of states corresponding to feasible points. More concretely, for any pair of feasible computational-basis states $\mathbf{x}, \mathbf{y} \in F$, there is some parameter value $\beta^*$ and some positive integer $r$ such that the corresponding mixer connects those two states: $|\langle \mathbf{x} \,|\, U_{\mathrm{M}}^r(\beta^*) \,|\, \mathbf{y}\rangle| > 0$.

In some cases we may want to relax some of these criteria. For example, if a QAOA circuit is being used as a subroutine within a hybrid quantum-classical algorithm, or in a broader quantum algorithm, we may use starting states informed by previous runs and thus allow mixing operators that mix less.

## 3 QAOA Mappings: Strings

This section describes mappings to QAOA for four problems in which the underlying configuration space is strings with letters taken from some alphabet. We introduce some basic families of mixers, and discuss compilations thereof, illustrating their use with MaxColorableSubgraph as an example. We then build on these basic mixers to design families of more complicated mixers, such as controlled versions of these mixers, and illustrate their use in mappings and circuits for the problems MaxIndependentSet, MaxColorableInducedSubgraph and MinGraphColoring as examples. The mixers we develop in this section, and close variants, are applicable to a wide variety of problems, as we will see in Sec. A.

### 3.1 Example: Max-$\kappa$-ColorableSubgraph

**Problem.** Given a graph $G = (V, E)$ with $n$ vertices and $m$ edges, and $\kappa$ colors, maximize the size (number of edges) of a properly vertex-colored subgraph.

The domain $F$ is the set of colorings $\mathbf{x}$ of $G$, an assignment of a color to each vertex. (Note that here and throughout, the term "colorings" includes *improper* colorings.) The domain $F$ can be represented as the set of length $n$ strings over an alphabet of $\kappa$ characters, $\mathbf{x} = x_1 x_2 \ldots x_n$, where $x_i \in [\kappa]$. The objective function $f : [\kappa]^n \to \mathbb{N}$ counts the number of properly colored edges in a coloring:

$$f(\mathbf{x}) = \sum_{\{u,v\} \in E} \mathsf{NEQ}(x_u, x_v). \tag{5}$$

We now build up machinery to define mixing operators for a QAOA approach to this problem. Since some mixing operators are more naturally expressed in one encoding rather than another, we find it useful to describe different mixing operators in different encodings, though we emphasize that doing so is merely for convenience; all mixing operators are encoding independent, so the descriptions may be translated from one encoding to another. The domain $F$ is naturally expressed as strings of $d$-dits, a $d$-valued generalization of bits. For the present problem, $d = \kappa$. In addition to discussing colorings as strings of dits, we will use a "one-hot" encoding into $n\kappa$ bits, with $x_{i,c}$ indicating whether or not vertex $i$ is assigned color $c$.

### 3.1.1 Single Qudit Mixing Operators

We focus initially on designing *partial mixers*, component operators that will be used to construct full mixing operators. For this mapping of the maxColorableSubgraph problem, the partial mixers are operators acting on a single qudit with dimension $d = \kappa$ that mix the colors associated to one vertex $v$. As we will see in subsequent sections, this is a particularly simple case of partial mixers which are often more complicated multiqubit controlled operators. Once we have defined these single-qudit partial mixing operators, we put them together to create a full mixer for the problem.

We begin by considering the following family of single-qudit mixing operators expressed in terms of qudits, and qudit operators, and then consider encodings and compilations to qubit-based architectures, which will inspire us to consider other families of single-qubit mixing operators. See App. C.2 and [6, 29] for a review of qudit operators, including the generalized Pauli operators $\check{X}$ and $\check{Z}$.

**$r$-nearby-values single-qudit mixer.** Let $U_{r\text{-NV}}(\beta) = e^{-i\beta H_{r\text{-NV}}}$, where $H_{r\text{-NV}} = \sum_{i=1}^{r} \left( \check{X}^i + (\check{X}^\dagger)^i \right)$, which acts on a single qudit, with $\check{X} = \sum_{a=1}^{d} |a+1\rangle \langle a|$. We identify two special cases by name: the "single-qudit ring mixer" for $r = 1$, $H_{\text{ring}} = H_{1\text{-NV}}$; and the "fully-connected" mixer for $r = d - 1$, $H_{\text{FC}} = H_{(d-1)\text{-NV}}$. Whenever we introduce a Hamiltonian, we also implicitly introduce its corresponding family of unitaries, as with $H_{r\text{-NV}}$ and $U_{r\text{-NV}}(\beta)$.

The single-qudit ring mixer will be our bread and butter. We concentrate on qubit encodings thereof, given the various projections of hardware with at least 40 qubits that will be available in the next year or two [55, 68], though it could alternatively be implemented directly using a qudit-based architecture. We explore two natural encodings of a qudit into qubits: (1) the *one-hot* encoding using $d$ qubits, in which the qudit state $|a\rangle$ is encoded as $|0\rangle^{\otimes a} \otimes |1\rangle \otimes |0\rangle^{\otimes d-1-a}$; and (2) the *binary* encoding using $\lceil \log_2 d \rceil$ qubits, in which the qudit state $|a\rangle$ is encoded as the qubit basis state corresponding to the binary representation of the integer $a$. The one-hot encoding uses more qubits but supports simpler compilation of the single-qudit ring mixer for general $d$, both in the sense of it being much easier to write down a compilation, and in the sense that it uses many fewer 2-qubit gates; in the one-hot encoding, 2-qubit mixing operators suffice, whereas the binary encoding requires $\lceil \log_2 d \rceil$-local Hamiltonian terms, with the corresponding unitary requiring further compilation to 2-qubit gates.

In the one-hot encoding, the single-qudit ring mixer is encoded as a qubit unitary $U_{\text{ring}}^{(\text{enc})}$ corresponding to the qubit Hamiltonian

$$H_{\text{ring}}^{(\text{enc})} = \sum_{a}^{d} \left( X_a X_{a+1} + Y_a Y_{a+1} \right), \tag{6}$$

which acts on the whole Hilbert space in a way that preserves the Hamming weight of computational basis states, and acts as $U_{\text{ring}}$ on the encoded subspace spanned by unit Hamming weight computational basis states.

Although $H_{\text{ring}}^{(\text{enc})}$ is 2-local, its terms do not mutually commute. There are several implementation options. First, hardware that natively implements the multiqubit gate $U_{\text{ring}}^{(\text{enc})}$ directly may be plausible (much as quantum annealers already support the simultaneous application of a Hamiltonian to a large set of qubits), but most proposals for universal quantum processors are based on two-qubit gates, so a compilation to such gates is desirable both for applicability to hardware likely to be available in the near term and for error-correction and fault tolerance in the longer term. Second, we could use constructions

used in quantum many-body physics to compile $U_{\text{ring}}^{(\text{enc})}$ into a circuit of 2-local gates [72]. Third, the multiqubit gate $U_{\text{ring}}^{(\text{enc})}$ could be implemented approximately via Trotterization or other Hamiltonian simulation algorithms. A different approach, and the alternative we explore most extensively here, is to implement a different unitary rather than $U_{r\text{-NV}}$, one related to $U_{r\text{-NV}}$, and sharing its desirable mixing properties, as encapsulated in the design criteria of Sec. 2.1, but is easier to implement. The form of the circuit obtained by Trotterization is suggestive. We consider sequentially apply unitaries corresponding to sets of terms in the Hamiltonian, each set chosen in such a way that the unitary is readily implementable. This reasoning mirrors the relation of H-QAOA circuits to Trotterized AQO. We give a few examples of mixers obtained in this way.

**Parity single-qudit ring mixer.** Still in the one-hot encoding, we partition the $d$ terms $e^{-i\beta(X_a X_{a+1} + Y_a Y_{a+1})}$ by parity. Let

$$U_{\text{parity}}(\beta) = U_{\text{last}}(\beta) U_{\text{even}}(\beta) U_{\text{odd}}(\beta), \tag{7}$$

where

$$U_{\text{odd}}(\beta) = \prod_{a \text{ odd, } a \neq n} e^{-i\beta(X_a X_{a+1} + Y_a Y_{a+1})}, \quad U_{\text{even}}(\beta) = \prod_{a \text{ even}} e^{-i\beta(X_a X_{a+1} + Y_a Y_{a+1})}, \tag{8}$$

$$U_{\text{last}}(\beta) = \begin{cases} e^{-i\beta(X_d X_1 + Y_d Y_1)}, & d \text{ odd}, \\ I, & d \text{ even}. \end{cases} \tag{9}$$

Such "XY" gates are natively implemented on certain superconducting processors [15].

It is easy to see that the parity single-qudit ring mixer meets the first criterion of keeping the evolution within the feasible subspace of a graph consisting of a single vertex. To see that it also meets the second, providing transitions between all feasible computational basis states, it is useful to consider the quantum swap gate, which behaves in exactly the same way as the XY gate on the subspace spanned by $\{|0, 1\rangle, |1, 0\rangle\}$. The swap gate $\text{SWAP}_{ij} = \frac{1}{2}(I + X_i X_j + Y_i Y_j + Z_i Z_j)$ is both unitary and Hermitian, and thus

$$e^{i\theta \text{SWAP}_{ij}} = \cos(\theta) I + i \sin(\theta) \text{SWAP}_{ij}. \tag{10}$$

For $0 < \beta < \pi/2$, each term in the parity single-qudit ring mixer is a superposition of a swap gate and the identity. A single application of $U_{\text{parity}}(\beta)$ will have non-zero transition amplitudes only between pairs of colors with indices no more than two apart. Nevertheless, this mixer meets the second criteria because all possible orderings of $d$ swap gates appear in $\lceil \frac{d}{2} \rceil$ repeats of the parity operator for any $0 < \beta < \pi/2$, thus providing non-zero amplitude transitions between all feasible computational basis states for a single-vertex graph.

When $d$ is an integer power of two, there is also a straightforward, though more resource-intensive, compilation of the parity single-qudit ring mixer using the binary encoding. Applying a Pauli $X$ gate to the least-significant qubit acts as $H_{\text{even}}$ on the encoded subspace. Incrementing the register by one, applying the Pauli gate to the least-significant qubit, and finally decrementing the register by one overall acts as $H_{\text{odd}}$. Therefore we can implement $U_{\text{parity}}$ by incrementing the register by one, applying an $X(\beta) = e^{-i\beta X}$ to the least-significant qubit, decrementing the register by one, and then again applying an $X(\beta)$ gate to the least-significant qubit. For an $l$-bit register, each incrementing and decrementing operation can be written as a series of $l$ multiply-controlled $X$ gates, with the numbers of control ranging from 0 to $l - 1$.

**Repeated parity single-qudit ring mixers.** As we mention above, a single application of $U_{\text{parity}}(\beta)$ will have non-zero transition amplitudes only between pairs of colors with indices no more than two apart, which suggests that it may be useful to repeat the parity mixer within one mixing step.

**Partition single-qudit ring mixers.** We now generalize the above construction for the parity single-qudit ring mixer to more general partition mixers. For a given ordered partition $\mathcal{P} = (P_1, \ldots, P_p)$ of the terms of $H_{r\text{-NV}}$ such that all pairs of terms with a $P_i$ act on disjoint states of the qudit, let

$$U_{\mathcal{P}\text{-}r\text{-NV}}(\beta) = U_{P_p\text{-XY}}(\beta) \cdots U_{P_1\text{-XY}}(\beta), \tag{11}$$

where

$$U_{P\text{-XY}}(\beta) = \prod_{\{a,b\} \in P} e^{-i\beta(|a\rangle\langle b| + |b\rangle\langle a|)}. \tag{12}$$

By construction, in the one-hot encoding, the terms of $U_{P\text{-XY}}$ commute (because they act on disjoint pairs of qubits), and so the ordering does not matter; all can be implemented in parallel. We call $U_{\mathcal{P}\text{-}r\text{-NV}}$ the "partition $\mathcal{P}$" $r$-nearby-values single-qudit ring mixer, and $U_{r\text{-NV}}$ the "simultaneous" $r$-nearby-values single-qudit ring mixer to distinguish the latter from the former. The latter is member of H-QAOA, while the former is not.

Even more generally, for a set of single-qudit ring mixers a $\{H_\alpha\}$ indexed by some $\alpha$ and an ordered partition $\mathcal{P} = (P_1, \ldots, P_p)$ thereof in which the single-qudit mixers within each part mutually commute, we define a simultaneous version, $e^{-i\beta \sum_\alpha H_\alpha}$ and a $\mathcal{P}$-partitioned version, $\prod_{i=1}^{P} \left[ \prod_{\alpha \in P_i} e^{-i\beta H_\alpha} \right]$, where the order of the product over the elements of each part doesn't matter because they commute, and the order of the product over the parts of the partition is given by their ordering within the ordered partition $\mathcal{P}$.

**Binary single-qudit mixer for $d = 2^l$.** We now return briefly to the binary encoding, and describe a different single-qudit mixer. An alternative to the $r$-nearby values single-qudit mixer, which is easily implementable using the binary encoding when $d = 2^l$ is a power of two, is the "simple binary" single-qudit mixer:

$$H_{\text{binary}}^{(\text{enc})} = \sum_{i=1}^{l} X_i, \tag{13}$$

where $X_i$ acts on the $i$th qubit in the binary encoding of the qudit. Since the ordering of the colors was arbitrary to begin with, it doesn't much matter whether the Hamiltonian mixes nearby values in the ordering or mixes the colors in a different way, in this case to colors with indices whose binary representations have Hamming distance 1.

When $d$ is not a power of 2, a straightforward generalization of the binary single-qudit mixer has difficulty in meeting the first of the design criteria since swapping one of the bit values in the binary representation may take the evolution out of the feasible subspace. While requiring $d$ to be a power of 2 restricts its general applicability, the binary single-qudit mixer could be useful in some interesting cases, such as 4-coloring. For 2-coloring (a problem is equivalent to MaxCut), the full binary single-qudit mixer is simply the standard mixer $X$. We will use this encoding in Sec. 4.2 to handle slack variables in a single-machine scheduling problem, a case in which there is flexibility in the upper range of the integer to be encoded, allowing us to round up to the nearest power of two when needed.

### 3.1.2 Full QAOA mapping

Having introduced several partial mixers for single qudits, we now show a complete QAOA circuit for an instance of MaxColorableSubgraph with $n$ vertices, $m$ edges, and $\kappa$ colors, compiled to 2-local gates on qubits. Using the one-hot encoding, we require $n\kappa$ qubits.

**Mixing operator.** We use as the full mixer a parity ring mixer made up of parity single-qudit ring mixers, one for each of the qudits corresponding to each vertex,

$$U_\mathrm{M} = \prod_{v=1}^{n} U_{v,\mathrm{parity}}^{(\mathrm{enc})}. \tag{14}$$

The single qubit mixers act on different qubits and can be applied in parallel. The overall parity mixing operator requires a depth-2 or depth-3 circuit (for even and odd $\kappa$, respectively) of $n\kappa$ gates. Other single-qudit mixers we defined above, including $r$ repeats of the parity ring mixer, other partitioned mixers, or the binary mixer, could be used in place of the parity single-qudit mixer in this construction. All of these unitary mixers by construction meet our first criterion for a mixer: keeping the evolution in the feasible subspace. Further, each of these mixers, after at most $\lceil \frac{\kappa}{2} \rceil$ repeats, provides non-zero amplitude transitions between all colors at a given vertex, with the product providing transitions between any two feasible states.

**Phase-separation operator.** The objective function can be written in the classical one-hot encoding as

$$m - \sum_{\{u,v\}\in E} \sum_{a=1}^{k} x_{u,a} x_{v,a}, \tag{15}$$

where $x_{v,a} = 1$ indicating that vertex $v$ has been assigned color $a$. To obtain a phase-separation Hamiltonian, we substitute $(I - Z)/2$ for each binary variable, to obtain

$$H'_\mathrm{P} = \frac{4 - \kappa}{4} m I + \frac{1}{4} \sum_{\{u,v\}\in E} \sum_{a=1}^{\kappa} \left( Z_{u,a} + Z_{v,a} - Z_{u,a} Z_{v,a} \right). \tag{16}$$

The constant term effects only a physically-irrelevant global phase, and since we are only concerned about the feasible subspace, we can disregard each sum $\sum_{a=1}^{\kappa} Z_{u,a}$ of all $\kappa$ single $Z$ operators corresponding to a single qudit, since they multiply each of the $d$ Hamming weight 1 elements corresponding to the $d$ single qudit values by the same constant, resulting in a global phase. Removing those terms and rescaling, the phase separator now has the simpler form

$$H_\mathrm{P}^{(\mathrm{enc})} = \sum_{\{u,v\}\in E} \sum_{a=1}^{\kappa} Z_{u,a} Z_{v,a}, \tag{17}$$

where $Z_{v,a}$ acts on the $a$th qubit in the one-hot encoding of the $v$th qudit, corresponding to coloring vertex $v$ with color $a$. The phase separator requires a circuit containing $m\kappa$ gates with depth at most $D_G + 1$, where $D_G$ is the maximum degree over all vertices in the instance graph $G$.

Translated back to acting on qudits, Eq. (17) acts as $H_\mathrm{P} = H_g$ (as defined in Eq. (3)), where $g(\mathbf{x}) = \kappa m - 4f(\mathbf{x})$. We refer to this function $g$ as the "phase function", which will typically be an affine transformation of the objective function, which corresponds simply to a physically irrelevant global phase and a rescaling of the parameter. Defining $H_\mathrm{P}$ using such a phase function allows us to write a simpler encoded version $H_\mathrm{P}^{(\mathrm{enc})}$ that corresponds exactly to $H_\mathrm{P}$, without qualification, on the encoded subspace.

**Initial state.** Any encoded coloring can be generated by a depth-1 circuit of no more than $n$ single-qubit $X$ gates. A reasonable initial state is one in which all vertices are assigned the same color. Alternatively, we could start with any other feasible state, or the initial state could be obtained by applying one or more rounds of the mixer to a single feasible state, so that the algorithm begins with a superposition of feasible states.

The circuit depth and gate count for the full algorithm will increase when compiling to realistic near-term hardware with architectures that have nearest neighbor topological constraints limiting between which pairs of physical qubits two-qubit gates can be applied. See [71] for one approach for compiling to realistic hardware with such constraints.

Further investigation is need to understand which mixers and initial states, for a given resource budget, result in a more or less effective altoritlms, and whether some have an advatage with respect to finding good parameters $\gamma_1, \ldots, \gamma_p$ and $\beta_1, \ldots, \beta_p$ or being robust to error.

## 3.2   Example: MaxIndependentSet

**Problem.** Given a graph $G = (V, E)$, with $|V| = n$ and $|E| = m$, find the largest subset $V' \subset V$ of mutually non-adjacent vertices.

This problem was discussed in Sec. VII of [22] as a "variant" of the quantum approximate optimization algorithm introduced in that paper. To handle this problem, Farhi et al. suggested restricting the evolution to what we are calling the feasible subspace of the overall Hilbert space, the subspace spanned by computational basis elements corresponding to independent sets, through modification of what we are calling the mixing operator. We now make the construction of the H-QAOA mixer Farhi et al. define more explicit, and introduce partitioned mixers that have implementation advantages over the H-QAOA, or simultaneous, mixer defined in Farhi et al..

The configuration space is the set of $n$-bit strings, representing subsets $V' \subset V$ of vertices, where $i \in V'$ if and only if $x_i = 1$. The domain $F$ is represented by the subset of all $n$-bit strings corresponding to independent sets of $G$. In contrast to the domain for MaxColorableSubgraph, this domain is dependent on the problem instance, not just on the size of the problem. Because the configuration is already bit-based, some aspects of mapping this problem to QAOA are simpler, but the partitioned mixing operators are more complicated in that they require controlled operation.

To support the discussion of controlled operators, we use the notation $\Lambda_{\mathbf{y}}(Q)$ to indicate a unitary *target operator* $Q$ applied to a set of target qubits controlled by the state $\mathbf{y}$ of a set of control qubits:

$$\Lambda_{\mathbf{y}}(Q) = \sum_{\mathbf{y}' \neq \mathbf{y}} |\mathbf{y}'\rangle \langle \mathbf{y}'| \otimes I + |\mathbf{y}\rangle \langle \mathbf{y}| \otimes Q. \tag{18}$$

More generally, we use $\Lambda_{\chi}(Q)$ when the operation is controlled on a predicate $\chi$:

$$\Lambda_{\chi}(Q) = \sum_{\mathbf{y}: \neg \chi(\mathbf{y})} |\mathbf{y}\rangle \langle \mathbf{y}| \otimes I + \sum_{\mathbf{y}: \chi(\mathbf{y})} |\mathbf{y}\rangle \langle \mathbf{y}| \otimes Q. \tag{19}$$

Whether the subscript of $\Lambda$ is a string or predicate will be clear from context. For a Hamiltonian $H_Q$ such that $Q = e^{-iH_Q}$, we can write the controlled unitary Eq. (19) as

$$\Lambda_{\chi}(Q) = e^{-iH_{\chi} \otimes H_Q}. \tag{20}$$

We refer to $H_{\chi} \otimes H_Q$ as the *controlled Hamiltonian*, $\chi$-controlled-$H_Q$. Note that Hamiltonian $H_{\chi}$ that acts as the predicate $\chi$ on computational basis states, in the sense of Eq. (3),

is precisely the projector $H_\chi = \sum_{\mathbf{x}:\chi(\mathbf{x})=1} |\mathbf{x}\rangle\langle\mathbf{x}|$ that projects onto the subspace on which the predicate is 1. We will use this relation to connect corresponding controlled Hamiltonians and controlled unitaries. In particular, when we want to apply a phase only on the part of the Hilbert space picked out by a predicate $\chi$, we can write

$$\Lambda_\chi\left(e^{-i\theta}\right) = e^{-i\theta H_\chi}, \tag{21}$$

where we have adapted the control notation of Eq. (20) to mean applying the operator $Q = e^{-i\theta}$ to zero target qubits. We will often compile controlled unitaries (both phase separators and mixers) by using ancilla qubits to intermediate the control, e.g. for a single ancilla qubit (initialized at $|0\rangle$ and returned thereto):

$$(\Lambda_\chi(Q))^{(\text{comp})} = \Lambda_\chi(X_{\text{anc}})\,\Lambda_{x_{\text{anc}}}(Q)\,\Lambda_\chi(X_{\text{anc}}) \tag{22}$$

### 3.2.1 Partial mixing operator at each vertex

Given an independent set $V'$, we can add a vertex $w \notin V'$ to $V'$ while maintaining feasibility only if none of its neighboring vertices $\text{nbhd}(w)$ are already in $V'$. On the other hand, we can always remove any vertex $w \in V'$ without affecting feasibility. Hence, a bit-flip operation at a vertex, controlled by its neighbors (adjacent vertices), suffices both to remove and add vertices while maintaining the independence property. These classical moves inspire the controlled-bit-flip partial mixing operators.

In general, for a string $\mathbf{y}$ and a set of indices $V$, let $\mathbf{y}_V = (y_{v_i})_{i=1}^{|V|}$ be the substring of $\mathbf{y}$ in lexicographical order of the indices. In particular, let $\mathbf{x}_{\text{nbhd}(v)} = (x_w)_{w\in\text{nbhd}(v)}$. (The ordering of the characters within the substring is arbitrary, because we only use this as the argument to predicates that are symmetric under permutation of the arguments.) For each vertex, we define the partial mixer as a multiply-controlled $X$ operator

$$\begin{aligned} H_{\text{CX},v} &= X_v H_{\text{NOR}(x_{\text{nbhd}(v)})} \\ &= 2^{-D_v} X_v \prod_{w\in\text{nbhd}(v)} (I + Z_w), \end{aligned} \tag{23}$$

with corresponding partial mixing unitary, a multiply-controlled-$X(\beta)$ single-qubit rotation,

$$\begin{aligned} U_{\text{CX},v}(\beta) &= e^{-i\beta H_{\text{CX},v}} \\ &= \Lambda_{\text{NOR}(x_{\text{nbhd}(v)})}\left(e^{-i\beta X_v}\right) \\ &= \Lambda_{\text{NOR}(x_{\text{nbhd}(v)})}\left(X_v(\beta)\right), \end{aligned} \tag{24}$$

where $X_v(\beta)$ is the single-qubit operator $X_v(\beta) = e^{-i\beta X_v}$. Since $X_v$ is both Hermitian and unitary, $e^{-i\beta X_v}$ is a combination of the identity and $X_v$ for $0 < \beta < \pi/2$.

### 3.2.2 Full QAOA mapping

**Mixing operators.** Let $H_{\text{CX}} = \sum_{i=1}^n H_{\text{CX},v_i}$. We define two distinct types of mixers:

- the *simultaneous* controlled-$X$ mixer, $U_{\text{sim-CX}}(\beta) = e^{-i\beta H_{\text{CX}}}$; and

- a class of *partitioned* controlled-$X(\beta)$ mixers, $U_{\mathcal{P}\text{-CX}}(\beta) = \prod_{i=1}^{|\mathcal{P}|} \prod_{v\in P_i} U_{\text{CX},v}$,

where $\mathcal{P}$ is an ordered partition of the partial mixers, in which each part contains mutually commuting partial mixers. Since the partial mixers often do not commute, different ordered partitions often result in dfferent mixers. By design, the partitioned mixers restricts evolution to the feasible subspace. With respect to the second design criterion, there is non-zero transition amplitude from the $|0\rangle^{\otimes n}$ state corresponding to the empty set to all other independent sets; for $0 < \beta < \pi/2$, we get terms corresponding to products of the individual control-bit-flip operators for all subsets of vertices, including those corresponding to independent sets. (For those subsets $S$ not corresponding to independent sets, the product will result in a independent set $V' \in S$ that does not include vertices in $S$ which have neighbors whose controlled-bit-flip preceded them in the partition order, thus different ordered partition affects the amount of non-zero amplitude in the states corresponding to independent sets.) Two applications of any partitioned mixer results in non-zero amplitude between any two feasible states. An interesting question is how different ordered partitions affect the ease with which good parameters can be found and the quality of the solutions obtained.

Partitioned mixers are generally easier to compile than the simultaneous mixer, since the partitioned mixer is a product of multiqubit-controlled-NOT operators, generalized Toffoli gates on at most $D_G + 1$ qubits. Altogether, this construction uses $n$ partial mixers, which can then be compiled into single- and two-qubit gates. For many graphs, partitions in which each set contains multiple commuting partial mixers exist, reducing the depth.

**Phase-separation operator.** The objective function is the size of the independent set, or $f(\mathbf{x}) = \sum_{i=1}^{n} x_i$, which we could translate into a phase-separating Hamiltonian via substitution of $(I - Z)/2$ for each binary variable. Instead, we use affine transformation of the objective function $g(\mathbf{x}) = n - 2f(\mathbf{x})$, which when translated yields a phase separation operator of a simpler form:

$$U_{\mathrm{P}}(\gamma) = e^{-i\gamma H_g} = \prod_{i=1}^{n} e^{-i\gamma Z_i}, \tag{25}$$

which is simply a depth-1 circuit of $n$ single-qubit Z-rotations.

**Initial state.** A reasonable initial state is the trivial state $|s\rangle = |0\rangle^{\otimes n}$ corresponding to the empty set.

## 3.3 Example: MaxColorableInducedSubgraph

**Problem.** Given $\kappa$ colors, and a graph $G = (V, E)$ with $n$ vertices and $m$ edges, find the largest induced subgraph that can be properly $\kappa$-colored.

The *induced subgraph* of a graph $G = (V, E)$ for a subset of vertices $W \subset V$ is the graph $H = (W, E_W)$, where the $E_W = \{\{v, w\} \in E : v, w \in W\}$. The configuration space is the set $[\kappa + 1]^n$ of $(\kappa + 1)$-dit strings of length $n$, corresponding to partial $\kappa$-colorings of the graph: $x_v = 0$ indicates that vertex $v$ is uncolored, and $x_v = c > 0$ indicates that the vertex has color $c$. The induced subgraph is defined by the colored vertices. The domain is the set of *proper* partial colorings, those in which two colored vertices that are adjacent in $G$ have different colors. The objective function $f : [\kappa + 1]^n \to \mathbb{N}$ is the number of vertices that are colored,

$$f(\mathbf{x}) = \sum_v \mathsf{NEQ}(x_v, 0). \tag{26}$$

### 3.3.1 Controlled null swap mixer at a vertex

The controlled-null-swap partial mixer we define has elements of the mixers we have seen for the previous two problems, combining the control by vertex neighbors from MaxIndependentSet and the color swap from MaxColorableSubgraph. Here, however, we make substantial use of the uncolored state, and only consider swapping a color with uncolored status at each vertex. An uncolored vertex can be assigned color $c$, maintaining feasibility as long as none of its neighbors have been assigned color $c$, suggesting a swap between a color and uncolored controlled on neighboring vertices; uncoloring a vertex always preserves feasibility. This reasoning in terms of classical moves inspires, for problems containing NEQ constraints, the controlled-null-swap partial mixing Hamiltonian

$$
\begin{aligned}
H_{\text{NS},v,a} &= (|a\rangle\langle 0|_v + |0\rangle\langle a|_v)\, H_{\text{NONE}(\mathbf{x}_{\text{nbhd}(v)},a)} \\
&= (|a\rangle\langle 0|_v + |0\rangle\langle a|_v) \prod_{w\in\text{nbhd}(v)} (I_w - |a\rangle\langle a|_w),
\end{aligned}
\tag{27}
$$

with corresponding controlled null-swap-rotation mixing unitary

$$
U_{\text{NS},v,a}(\beta) = \Lambda_{\text{NONE}(\mathbf{x}_{\text{nbhd}(v)},a)}\left(e^{-i\beta\left(|a\rangle\langle 0|_v + |0\rangle\langle a|_v\right)} + \sum_{b\notin\{0,a\}} |b\rangle\langle b|_v\right),
\tag{28}
$$

where

$$
\text{NONE}(\mathbf{y}, A) = \bigwedge_{\substack{y\in\mathbf{y} \\ a\in A}} \text{NEQ}(y,a) = \bigwedge_{a\in A}\bigwedge_{i=1}^{|\mathbf{y}|}\text{NEQ}(y,a) = \bigwedge_{a\in A}\bigwedge_{i\in[|\mathbf{y}|]}\text{NEQ}(y,a)
\tag{29}
$$

is shorthand for none of the variables in $\mathbf{y}$ having value any of the values in $A$; when $A = \{a\}$ is a singleton set, we write simply $\text{NONE}(\mathbf{y},a) = \text{NONE}(\mathbf{y},\{a\})$.

### 3.3.2 Full QAOA mapping

**Mixing operators.** Define

$$
H_{\text{NS}} = \sum_{i=1}^{n}\sum_{a=1}^{\kappa} H_{\text{NS},i,a}.
\tag{30}
$$

We define two distinct types of mixers:

- the *simultaneous* controlled null-swap mixer, $U_{\text{sim-NS}}(\beta) = e^{-i\beta H_{\text{NS}}}$; and

- a family of *partitioned* controlled null-swap mixers, $U_{\mathcal{P}\text{-NS}}(\beta) = \prod_{a=1}^{\kappa}\prod_{i=1}^{|\mathcal{P}|}\prod_{v\in P_i} U_{\text{NS},v,a}$.

Again, we have a variety of partitioned mixers, each specified by an ordered partition $\mathcal{P}$ of the vertices such that for each color the terms corresponding to the vertices in the partition commute. We have segregated the colors into separate stages, but other orderings are possible.

We use the one-hot encoding of Sec. 3.1, but with additional variables $x_{v,0}$ for the uncolored states: the binary variables for each vertex $v$ are $x_{v,0}, x_{v,1}, \ldots, x_{v,k}$. This encoding uses $n(\kappa+1)$ computational qubits. In this encoding, a single partial mixer has the form

$$
U_{\text{NS},v,a}^{(\text{comp})}(\beta) = \Lambda_{\text{NOR}(x_{\text{nbhd}(v)},a)}\left(e^{-i\beta(X_{v,0}X_{v,a}+Y_{v,a}Y_{v,0})}\right),
\tag{31}
$$

where $\bar{\mathbf{x}}_{\text{nbhd}(v),a} = (x_{w,a})_{w \in \text{nbhd}(v)}$. Reasoning similar to that we used for the mixers discussed for the MaxIndependentSet and MaxColorableSubgraph problems shows that this mixer has non-zero transition amplitude between any feasible computational-basis state and the trivial state corresponding to the empty set as the induced subgraph. Two applications of this mixer give non-zero transition amplitudes between any two feasible computational-basis states.

To ease compilation, each partial mixer can be implemented as

$$U_{\text{NS},v,a}^{(\text{comp})}(\beta) = \Lambda_{\text{NOR}(x_{\text{nbhd}(v),a})} (X_{\text{anc}}) \Lambda_{x_{\text{anc}}} \left( e^{-i\beta(X_{v,0}X_{v,a} + Y_{v,a}Y_{v,0})} \right) \Lambda_{\text{NOR}(x_{\text{nbhd}(v),a})} (X_{\text{anc}}),$$

$$(32)$$

where the control is intermediated by an ancilla qubit, which is initialized and return to the zero state. Altogether, this construction uses $\kappa n$ partial mixers, which can then be compiled into single- and two-qubit gates. For many graphs, partitions in which each set contains multiple commuting partial mixers exist, reducing the depth.

**Phase-separation operators.** We can translate the objective function to a Hamiltonian as usual, or translate a linear modification of the objective function to obtain a simpler form.

The phase separator function $g(\mathbf{x}) = n - 2f(\mathbf{x})$ corresponding to the simple compiled phase separator

$$H_{\text{P}}^{(\text{comp})} = \sum_v Z_{v,0}, \qquad (33)$$

which can be implemented using a depth-1 circuit of $n$ single-qubit $Z$-rotations.

**Initial state.** A reasonable initial state is $|s\rangle = \left( |1\rangle \otimes |0\rangle^{\otimes \kappa} \right)^{\otimes n}$, corresponding to all vertices uncolored.

## 3.4 Example: MinGraphColoring

**Problem.** Given a graph $G = (V, E)$, find the minimal number of colors $k^*$ required to properly color it.

A graph that can be $\kappa$-colored but not $(\kappa - 1)$-colored is said to have *chromatic number* $\kappa$. We take as our configuration space the set of $\kappa$-dit strings of length $n$, where $\kappa = D_G + 2$. The domain $F$ is the set of *proper* colorings, many of which will use fewer than $\kappa$ colors. With $D_G + 2$ colors, as we explain next, it is possible to get from any proper coloring to any other by local moves while staying in the feasible space, a property we will make use of in designing mixing operators. We comment that it may be advantageous to take use a larger number of colors since that may promote mixing, but the tradeoffs there would need to be determined in a future investigation.

It is easy to see that any graph can be colored with $D_G + 1$ colors. To see that $\kappa = D_G + 2$ suffices to get between any two $D_G + 1$ colorings, first recognize that given a $D_G + 2$ coloring, one can always obtain a $D_G + 1$ coloring by simply choosing a color and recoloring each vertex currently colored in that color with one of the other colors, since at least one of those colors will not be used by its neighbors. This move is local, in that it depends only on the neighborhood of the vertex. Now, given two $D_G + 1$ colorings $C$ and $C'$, we will iterate through colors $c$ to tranform between the two colorings via local moves while staying in the feasible space. Let $S' \subset V$ be the set of vertices colored $c$ in $C'$, and let $S \subset S'$ be the set of vertices in $S'$ that are not colored $c$ in $C$. Consider all neighbors of vertices in $S$. For any neighbor colored $c$, color it with the unused color. We are now

free to color all vertices in $S$ with color $c$. Iterating through the $\kappa$ colors, provides a means of getting from one $D_G + 1$ coloring to another by local moves that remain in the feasible space.

### 3.4.1 Partial mixer at a vertex

We use a controlled version of the mixer in Sec. 3.1 that allows a vertex to change colors only when doing so would not result in an improper coloring; we may swap colors $a$ and $b$ at vertex $v$ only if none of its neighbors are colored $a$ or $b$. The partial mixer we define has a similar form to the controlled-null-swap partial mixer defined in Eqs. (27) and (28), but supports color changes between any two colors at a vertex, rather than only between a color and uncolored. Define the controlled-swap partial mixing Hamiltonian:

$$
\begin{aligned}
H_{\mathrm{CS},v,\{a,b\}} &= \left( |a\rangle\langle b|_v + |b\rangle\langle a|_v \right) H_{\mathsf{NONE}(\mathbf{x}_{\mathrm{nbhd}(v)},\{a,b\})} \\
&= \left( |a\rangle\langle b|_v + |b\rangle\langle a|_v \right) \prod_{w \in \mathrm{nbhd}(v)} \left( I_w - |a\rangle\langle a|_w - |b\rangle\langle b|_w \right),
\end{aligned}
\tag{34}
$$

with corresponding controled-swap-rotation mixing unitary

$$
U_{\mathrm{CS},v,\{a,b\}} = \Lambda_{\mathsf{NONE}(\mathbf{x}_{\mathrm{nbhd}(v)},\{a,b\})} \left( e^{-i\beta\left(|a\rangle\langle b|_v + |b\rangle\langle a|_v\right)} + \sum_{c \notin \{a,b\}} |c\rangle\langle c|_v \right),
\tag{35}
$$

where $\mathsf{NONE}(\mathbf{x}, A)$ was defined in Eq. (29). These mixers are controlled versions of the single qudit fully-connected mixer of Sec. 3.1, rather than the single qudit ring mixer, which makes sure that every possible state is reachable.

### 3.4.2 Full QAOA Mapping

**Mixing Operator.** Let

$$
H_{\mathrm{CS}} = \sum_v \sum_{a,b} H_{\mathrm{CS},v,\{a,b\}} \,.
\tag{36}
$$

We define two types of mixers:

- the *simultaneous* controlled-swap mixer,

$$
U_{\mathrm{sim\text{-}NS}}(\beta) = e^{-i\beta H_{\mathrm{CS}}}; \text{ and}
\tag{37}
$$

- a family of *partitioned* controlled-swap mixers,

$$
U_{\mathcal{P}\text{-}\mathrm{NS}}(\beta) = \prod_{a,b} \prod_{i=1}^{|\mathcal{P}|} \prod_{v \in P_i} U_{\mathrm{CS},v,\{a,b\}}(\beta) \,.
\tag{38}
$$

As before, each partitioned mixer is specified by an ordered partition $\mathcal{P}$ of the vertices such that, for each color, the partial mixers for vertices in one set of the partition all commute with each other. Altogether, this construction uses $(\kappa - 1)\kappa n/2$ partial mixers, For many graphs, partitions in which each set contains multiple commuting partial mixers exist, allowing different partial mixers to be carried out in parallel, reducing the depth.

**Phase-separation operator.** The objective function, $f : [\kappa]^n \rightarrow \mathbf{Z}_+$, is

$$f(\mathbf{x}) = \sum_{a=1}^{\kappa} \mathsf{OR}(\mathsf{EQ}(x_1, a), \ldots, \mathsf{EQ}(x_n, a)) \,, \tag{39}$$

which counts the numbers of colors used. Let $g(\mathbf{x}) = \kappa - f(\mathbf{x})$ be the phase operator that counts the number of colors *not* used. Let $H_{\mathsf{NONE}(\mathbf{x},a)}$ be the projector onto the subspace of $\mathcal{H}$ spanned by the states corresponding to strings in $[\kappa]^n$ that do not contain the character $a$. We have $H_g = \sum_a H_{\mathsf{NONE}(\mathbf{x},a)}$, so

$$U_{\mathrm{P}}(\gamma) = e^{-i\gamma H_g} = \prod_{a=1}^{\kappa} e^{-i\gamma H_{\mathsf{NONE}(\mathbf{x},a)}}. \tag{40}$$

**Initial state.** For the initial state, we use an easily found $D_G + 1$ (or $D_G + 2$) coloring.

### 3.4.3 Compilation in one-hot encoding

We now give partial compilations of the elements of the mapping to qubits using the one-hot encoding.

   **Mixing operators.** In the one-hot encoding, the controlled-swap mixing Hamiltonian can be written

$$\begin{aligned}
H_{\mathrm{CS},v,\{a,b\}}^{\mathrm{(comp)}} &= (X_{v,a}X_{v,b} + Y_{v,a}Y_{v,b}) \, H_{\mathsf{NOR}(x_{\mathrm{nbhd}(v),\{a,b\}})} \\
&= (X_{v,a}X_{v,b} + Y_{v,a}Y_{v,b}) \prod_{w \in \mathrm{nbhd}(v)} \prod_{c \in \{a,b\}} (I_{w,c} + Z_{w,c})
\end{aligned} \tag{41}$$

with corresponding unitary written as

$$U_{\mathrm{CS},v,\{a,b\}}^{\mathrm{(comp)}}(\beta) = \Lambda_{\mathsf{NOR}(x_{\mathrm{nbhd}(v),\{a,b\}})} \left( e^{-i\beta\left(X_{v,a}X_{v,b} + Y_{v,a}Y_{v,b}\right)} \right), \tag{42}$$

where, for a string doubly indexed $\mathbf{y} = (y_{i,j})_{i,j}$, $\mathbf{y}_{A,B} = \left( (y_{i,j})_{i \in A} \right)_{j \in B}$ denotes the substring consisting the characters $y_{i,j}$ for which $i \in A$ and $j \in B$, in lexicographical ordering of the two indices. In particular, $\mathbf{x}_{\mathrm{nbhd}(v),\{a,b\}}$ indicates the bits corresponding to coloring the neighbors of $v$ either color $a$ or color $b$. The $\Lambda_{\mathsf{NOR}(x_{\mathrm{nbhd}(v),\{a,b\}})}$ dictates that none of the neighbors of $v$ take value $a$ or $b$ for the swap to be performed. Each $U_{\mathrm{CS},v,\{a,b\}}^{\mathrm{(comp)}}$ is a controlled gate with $2D_v$ control qubits and two target qubits. Altogether, the full mixing Hamiltonian can be implemented using $\kappa(\kappa-1)n/2$ controlled gates on no more than $D_G + 2$ qubits.

   **Phase separator.** Let $U_{\mathrm{P},a}(\gamma) = e^{-i\gamma H_{\mathsf{NONE}(\mathbf{x},a)}}$, so that the phase separator Eq. (40) can be written as $U_{\mathrm{P}}(\gamma) = \prod_{a=1}^{\kappa} U_{\mathrm{P},a}(\gamma)$. Each partial phase separator can alternatively be written as

$$U_{\mathrm{P},a} = \Lambda_{\mathsf{NONE}(\mathbf{x},a)} \left( e^{-i\gamma} \right). \tag{43}$$

**Initial state.** Any coloring can be prepared in depth 1 using $n$ single-qubit $X$ gates:

$$|\mathbf{x}\rangle = \left( \prod_{i=1}^{n} X_{i,x_i} \right) |0\rangle^{\otimes n\kappa}. \tag{44}$$

# 4 QAOA Mappings: Orderings and schedules

Many challenging computational problems have a configuration space that is fundamentally the set of orderings or schedules of some number of items. Here, we introduce the machinery for mapping such problems to QAOA, using the traveling salesperson and several single-machine scheduling problem as illustrative examples.

## 4.1 Example: Traveling Salesperson Problem (TSP)

**Problem.** Given a set of $n$ cities, and distances $d : [n]^2 \in \mathbb{R}_+$, find an ordering of the cities that minimizes the total distance traveled for the corresponding tour.
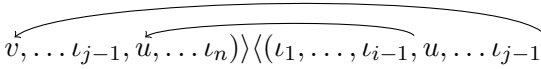
While for expository purposes, we call these numbers distances, the mapping works for any cost function on pairs of cities, whether or not it forms a metric or not; the distances are not required to be symmetric, or to satisfy the triangle inequality.

### 4.1.1 Mapping

The configuration space here is the set of all orderings of the cities. Labeling the cities by $[n]$, the ordering $\boldsymbol{\iota} = (\iota_1, \iota_2, \ldots, \iota_{n-1}, \iota_n)$ indicates traveling from city $\iota_1$ to city $\iota_2$, then on to city $\iota_3$, and so on until finally returning from city $\iota_n$ back to city $\iota_1$. The configuration space includes some degeneracy in solutions with respect to cyclic permutations; specifically, for any ordering $\boldsymbol{\iota}$, the configuration space includes both $(\iota_1, \iota_2, \ldots, \iota_{n-1}, \iota_n)$ and $(\iota_2, \iota_3, \ldots, \iota_n, \iota_1)$, even though the are essentially the same solution to TSP. We leave in this degeneracy to preserve symmetries which make it easier to construct mixers. As there are no problem constraints, the domain is the same as the configuration. The objective function is

$$f(\boldsymbol{\iota}) = \sum_{j=1}^{n} d_{\iota_j, \iota_{j+1}}. \tag{45}$$

**Ordering swap partial mixing Hamiltonians.** Our mixers for orderings will be built on the partial mixers we call "value-selective ordering swap mixing Hamiltonians." Consider $\{\iota_i, \iota_j\} = \{u, v\}$, indicating that city $u$ (resp. $v$) is visited at the $i$th (resp. $j$th) stop on the tour. There are $\binom{n}{2}^2$ value-selective ordering swap mixing Hamiltonians, $H_{\mathrm{PS}, \{i,j\}, \{u,v\}}$, which swap the $i$th and $j$th elements in the ordering if and only if those elements are the cities $u$ and $v$:

$$
\begin{aligned}
&H_{\mathrm{PS}, \{i,j\}, \{u,v\}} \\
&= \sum_{\boldsymbol{\iota}:\{\iota_i, \iota_j\}=\{u,v\}} |(\iota_1, \ldots, \iota_{i-1}, \overset{\frown}{v, \ldots \iota_{j-1}, u}, \ldots \iota_n)\rangle\langle(\iota_1, \ldots, \iota_{i-1}, u, \ldots \iota_{j-1}, v, \ldots \iota_n)| \, .
\end{aligned}
\tag{46}
$$

We make extensive use of a special case, the *adjacent* ordering swap mixing Hamiltonians,

$$H_{\mathrm{PS}, i, \{u,v\}} = H_{\mathrm{PS}, \{i,i+1\}, \{u,v\}} \, . \tag{47}$$

To swap the $i$th and $j$th elements of the ordering regardless of which cities those are, we use the value-independent ordering swap partial mixing Hamiltonian

$$H_{\mathrm{PS}, \{i,j\}} = \sum_{\{u,v\} \in \binom{[n]}{2}} H_{\mathrm{PS}, \{i,j\}, \{u,v\}}. \tag{48}$$

Of these $\binom{n}{2}$ partial mixing Hamiltonians, $n$ are adjacent value-independent ordering swap partial mixing Hamiltonians,

$$H_{\text{PS},i} = \sum_{\iota} |(\iota_1, \ldots \iota_{i-1}, \iota_{i+1}, \iota_i, \iota_{i+2}, \ldots, \iota_n)\rangle \langle (\iota_1, \ldots, \iota_n)|, \tag{49}$$

which swap the $i$th element with the subsequent one regardless of which cities those are.

These partial mixers can be combined in several ways to form full mixers, of which we explore two types.

**Simultaneous ordering swap mixer.** Defining $H_{\text{PS}} = \sum_{i=1}^{n} H_{\text{PS},i}$, we have the "simultaneous ordering swap mixer",

$$U_{\text{sim-PS}}(\beta) = e^{-i\beta H_{\text{PS}}}. \tag{50}$$

Different subsets and partitions of the partial mixers $U_{\text{PS},\{i,j\},\{u,v\}} = e^{-iH_{\text{PS},\{i,j\},\{u,v\}}}$, and different orderings of a partition, yield different partitioned ordering swap mixers. The color parity mixers we now define use the adjacent partial mixer. Other mixers, using more of the $\binom{n}{2}^2$ partial mixers, are possible, as are repeated versions of the following color-parity ordering swap mixer.

**Color-parity ordering swap mixer.** To define the ordered partition, we first define an ordered partition on the set of partial mixers $U_{\text{PS},i,\{u,v\}}$ for a fixed tour position $i$, where the parts of this partition contains mutually commuting partial mixers. We then partition the $i$ to obtain a full ordered partition. Two partial mixers $U_{\text{PS},i,\{u,v\}}$ and $U_{\text{PS},i,\{u',v'\}}$ commute as long as $\{u, v\} \cap \{u', v'\} = \emptyset$. Partitioning the $\binom{n}{2}$ pairs of cities into $\kappa$ parts such that each part contains only mutually disjoint pairs is equivalent to considering a $\kappa$-edge-coloring of the complete graph $K_n$ and assigning an ordering to the colors. For odd $n$, $\kappa = n$ suffices, and for even $n$, $\kappa = n - 1$ suffices [1]. (Using the geometrical construction based on regular polygons, we can define the canonical partition by placing the vertices at the vertices of the polygon in order, with the last one in the center for even $n$; the parts of the partition are then ordered by their lowest element under the lexicographical ordering of the pairs of cities $\{u, v\}$.) Let $\mathcal{P}_{\text{col}} = (P_1, \ldots, P_c, \ldots, P_\kappa)$ be the resulting ordered partition, which we call a "color partition" of the pairs of cities. For example, for $n = 4$, the partition is $\mathcal{P}_{\text{col}} = (\{\{1, 2\}, \{3, 4\}\}, \{\{1, 3\}, \{2, 4\}\}, \{\{1, 4\}, \{2, 3\}\})$. For different tour positions $i$, two partial unitaries $U_{\text{PS},i,\{u,v\}}$ and $U_{\text{PS},i',\{u',v'\}}$ commute if $i$ and $i'$ are not consecutive ($|i - i' \mod n| > 1$). Thus, for partitioning the positions, we may use the parity partition $\mathcal{P}_{\text{par}}$ defined in Sec. 3.1. We can thus define the "color-parity" ordered partition $\mathcal{P}_{\text{CP}} = \mathcal{P}_{\text{col}} \times \mathcal{P}_{\text{par}}$, with the induced lexicographical ordering of the parts. The part $P_{c,\text{odd}}$ contains all $U_{\text{PS},i,\{u,v\}}$ such that $i$ is odd and edge $\{u, v\}$ is colored $c$, i.e. in $P_c$, and defines the unitary

$$U_{c,\text{odd}}(\beta) = \prod_{(i,\{u,v\}) \in P_{c,\text{odd}}} U_{\text{PS},i,\{u,v\}}(\beta), \tag{51}$$

where the ordering of the products does not matter because each term commutes. Similarly for $P_{c,\text{even}}$ and $U_{c,\text{even}}$, and $P_{c,\text{last}}$ and $U_{c,\text{last}}$. Thus we have the full color-parity mixer

$$U_{\text{CP}}(\beta) = U_{\mathcal{P}_{\text{CP}}\text{-PS}}(\beta) = \prod_{P_{c,\pi} \in \mathcal{P}_{\text{CP}}} U_{c,\pi}, \tag{52}$$

where the unitaries $\{U_{c,\pi}\}$ are applied in the order they appear in $\mathcal{P}_{\text{CP}}$. The color-parity partition is optimal with respect to the number of parts in the partition (exactly so for

even $n$ and up to an additive factor of 2 for odd $n$). By construction, application of this mixer to any feasible state results in a feasible state, thus satisfying the first design criterion. With regard to the second criterion, while a single application of this mixer will have non-zero transitions only between orderings that swap cities in tour positions no more than two apart, repeating the mixer sufficiently many times results in non-zero transitions between any two states representing orderings. More specifically, since any ordering can be obtained from any other with no more than $\frac{n(n-1)}{2}$ adjacent swaps, alternating between odd and even swaps, $\frac{n(n-1)}{2}$ repeats suffices for any $0 < \beta < \pi/2$.

### 4.1.2 Compilation

**Encoding orderings.** We encode orderings in two stages: first into strings, then into bits making use of encodings from Sec. 3. Here, we focus on a "direct encoding", as opposed to the "absolute encoding" that will be introduced in Sec. 4.3. Other encodings of orderings are possible, such as the Lehmer code and inversion tables. In the direct encoding, an ordering $\boldsymbol{\iota} = (\iota_1, \ldots, \iota_n)$ is encoded directly as a string $[n]^n$ of integers. Once in the form of strings, any of the string encodings introduced in Sec. 3 can be applied. We apply the one-hot encoding with $n^2$ binary variables; the binary variable $x_{j,u}$ indicates whether or not $\iota_j = u$ in the ordering, in other words, whether city $u$ is visited at the $j$-th stop of the tour.

**Phase separator.** We use the phase function $g(\boldsymbol{\iota}) = 4f(\boldsymbol{\iota}) - (n-2)\sum_{u=1}^{n}\sum_{v=1}^{n}d(u,v)$, which translates to a phase separator encoded as

$$H_{\mathrm{P}}^{(\mathrm{enc})} = \sum_{i=1}^{n}\sum_{u=1}^{n}\sum_{v=1}^{n}d(u,v)Z_{u,i}Z_{v,i+1} . \tag{53}$$

The phase separating unitary corresponding to Eq. (53) imparts a phase determined by the sum of the distances between successive cities to a state corresponding to a tour. This unitary can be implemented using $n^2(n-1)$ 2-qubit gates, which mutually commute. Using the same color-parity partition of the terms as for the color-parity ordering swap mixer, this can be done in depth $2\kappa \leq 2n$.

**Mixer.** The individual value-selective ordering swap partial mixer, which swaps cities $u$, $v$ between tour positions $i$ and $j$, is expressed in the one-hot encoding as

$$U_{\mathrm{PS},\{i,j\},\{u,v\}}^{(\mathrm{enc})}(\beta) = e^{-i\beta H_{\mathrm{PS},\{i,j\},\{u,v\}}} , \tag{54}$$

$$H_{\mathrm{PS},\{i,j\},\{u,v\}}^{(\mathrm{enc})} = S_{u,i}^{+}S_{v,j}^{+}S_{u,j}^{-}S_{v,i}^{-} + S_{u,i}^{-}S_{v,j}^{-}S_{u,j}^{+}S_{v,i}^{+}, \tag{55}$$

where

$$S^{+} = X + iY = |1\rangle\langle 0| , \tag{56}$$

$$S^{-} = X - iY = |0\rangle\langle 1| . \tag{57}$$

The $i$th adjacent value-selective swap partial mixer (Eq. (47)) is the special case

$$H_{\mathrm{PS},i,\{u,v\}}^{(\mathrm{enc})} = S_{u,i}^{+}S_{v,i+1}^{+}S_{u,i+1}^{-}S_{v,i}^{-} + S_{u,i}^{-}S_{v,i+1}^{-}S_{u,i+1}^{+}S_{v,i}^{+} . \tag{58}$$

Each of the two term of the form $S^{+}S^{+}S^{-}S^{-}$ in Eq. (58), can be written as a sum of eight terms, each a product of 4 Pauli operators (e.g. $XXYY$). The color-parity partitioned ordering swap mixer of Eq. (52) can be implemented using $(n-1)\binom{n}{2}$ of these 4-qubit gates, implementable in depth $2\kappa \leq 2n$ in these gates. The 2-qubit gate circuit depth is at most $2\kappa$ times the depth of a compilation for such 4-qubit gates.

**Initial state.** The initial state, an arbitrary ordering, can be prepared from the zero state using at most $n$ single-qubit $X$ gates.

## 4.2 Example: Single Machine Scheduling, Minimizing Total Squared Tardiness

**Problem.** $(1|d_j|\sum w_j T_j^2)$ Given a set of $n$ jobs with processing times $\mathbf{p}$, deadlines $\mathbf{d} \in \mathbb{Z}_+$, and weights $\mathbf{w}$, find a schedule minimizing the total weighted squared tardiness $\sum_{j=1}^n w_j T_j^2$. The tardiness of job $j$ with completion time $C_j$ is defined as $T_j = \max\{0, C_j - d_j\}$.

The configuration space and domain are the set of all orderings of the jobs. Given an ordering $\boldsymbol{\iota}$ of the jobs in which job $i$ is the $\sigma_i$-th job to start, the corresponding schedule $\mathbf{s}(\boldsymbol{\iota})$ is that in which each job starts as soon as the earlier jobs finish: $s_j(\boldsymbol{\iota}) = \sum_{i=1}^{\sigma_j-1} p_{\iota_i}$.

For a job $i$ starting at time $s_i$, consider the expression

$$\min_{y_i \in [0, d_i - p_i]} (s_i + p_i - d_i + y_i)^2 = \begin{cases} (s_i + p_i - d_i)^2, & s_i + p_i > d_i, \\ 0, & \text{otherwise.} \end{cases} \tag{59}$$

When the "slack" variable $y_i \in [0, d_i - p_i]$ is minimized, this expression is equal to the square of the tardiness of job $i$. Therefore, we recast SMS as the minimization of

$$f(\boldsymbol{\iota}, \mathbf{y}) = \sum_{i=1}^n w_i (s_i(\boldsymbol{\iota}) + p_i - d_i + y_i)^2 \tag{60}$$

over the configuration space of orderings $\boldsymbol{\iota}$ and slack variables $y_i$.

Using the direct one-hot encoding defined in Sec. 4.1.2, in which $x_{j,\alpha}$ indicates that job $j$ is the $\alpha$-th to start, this is equivalent to

$$f(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n w_i (s_i(\mathbf{x}) + p_i - d_i + y_i)^2, \tag{61}$$

where

$$s_i(\mathbf{x}) = \sum_{\alpha=2}^n x_{i,\alpha} \sum_{j \neq i} p_j \sum_{\beta=1}^{\alpha-1} x_{j,\beta}. \tag{62}$$

Note Eq. (61) may seem to be quartic; however, the encoding constraints $\sum_i x_{i,\alpha} = \sum_\alpha x_{i,\alpha} = 1$ that come with the direct one-hot encoding imply that the quartic terms disappear in the full expansion. The objective function is thus a cubic pseudo-Boolean function, which corresponds to a 3-local diagonal Hamiltonian for the phase separator.

**Mixer and initial state** We use the same initial state preparation, and the same mixer as in TSP for mixing the ordering, in addition to any of the single-qudit mixers from Sec. 3.1.1 for each of the slack variables. Because the ordering and slack mixers act on separate sets of qubits ($\mathbf{x}$ and $\mathbf{y}$), they can be implemented in parallel. Note that the only requirement for the upper bound of the range of the slack variable $y_i$ is that it be *at least* $d_i - p_i + 1$. In particular, it could be $2^{\lceil \log_2(d_i - p_i + 1) \rceil}$, allowing us to use the binary encoding without modification.

## 4.3 SMS, minimizing total tardiness

**Problem.** $(1|d_j|\sum w_j T_j)$ Given a set of jobs with processing times $\mathbf{p}$, deadlines $\mathbf{d} \in \mathbb{Z}_+$, and weights $\mathbf{w}$, find a schedule minimizing the total weighted tardiness $\sum_{j=1}^n w_j T_j$.

### 4.3.1 Encoding and mixer

The configuration space is the set of orderings of the jobs; the domain is the same.

**Absolute and positional encodings.** In the "absolute" encoding of the ordering $\boldsymbol{\iota} = (\iota_1, \ldots, \iota_n)$, we assign each item $i$ a value $s_i \in [0, h]$, where the "horizon" $h$ is a parameter of the encodings, such that for all $i < j$, $s_{\iota_i} < s_{\iota_j}$. In certain cases, there will be an item-specific horizon $h_i$ such that $s_i \in [0, h_i]$. Note that in general the relationship between encoded states and the orderings they encode is not injective, but it will be in the domains to which we apply it. Once the ordering $\boldsymbol{\iota}$ is encoded as a string $\mathbf{s}(\boldsymbol{\iota}) \in \times_{i=1}^n [0, h_i]$, the resulting string can be encoded using any of the string encodings previously introduced. We call the special case of the "absolute" encoding with $h = n$ the "positional" encoding; using the one-hot encoding of the resulting strings, the "direct" and "positional" encodings are the same.

**Time-swap mixer.** We now introduce a mixer that is specific to the absolute encoding in which there is a single horizon $h$ and each job $i$ has a processing time $p_i$. Let the horizon be $h = \sum_{i=1}^n p_i$. Each job can start between time 0 and $h_i = h - p_i$. (Other optimizations may be made on an instance-specific basis, though we neglect those for ease of exposition.) We use a "time swap" partial mixer that acts on absolutely encoded orderings

$$H_{\text{TS},t,\{i,j\}} = |t + p_j, t\rangle \langle t, t + p_i|_{i,j} + |t, t + p_i\rangle \langle t + p_j, t|_{i,j}, \tag{63}$$

which swaps jobs $i$ and $j$ when they are scheduled immediately after one another with the earlier one starting at time $t$. To swap them regardless of the time at which the earlier one starts we use

$$H_{\text{TS},\{i,j\}} = \sum_{t=0}^{h-p_i-p_j} H_{\text{TS},t,\{i,j\}}. \tag{64}$$

The *simultaneous* time swap mixer is as usually constructed,

$$U_{\mathcal{P}\text{-TS}}(\beta) = e^{-i\beta H_{\text{TS}}}, \tag{65}$$

where $H_{\text{TS}} = \sum_{\{i,j\} \in \binom{[n]}{2}} H_{\text{TS},\{i,j\}}$. Note that while the simultaneous versions of the total time swap and adjacent permutation mixers are exactly the same, and in particular

$$H_{\text{TS},\{i,i'\}} = \sum_{j=1}^{n-1} H_{\text{PS},j,\{i,i'\}}, \tag{66}$$

the individual partial mixer $H_{\text{TS},t,\{i,j\}}$ has no equivalent that acts on the unencoded ordering, because it acts depending on the total processing times of the preceding jobs rather than their number.

Now consider the "time-color" ordered partition $\mathcal{P}_{\text{TC}} = [0, h] \times [\kappa]$, where, as in Sec. 4.1.1, we use a particular $\kappa$-edge-coloring of the complete graph. (Again, further optimizations may be made on an instance-specific basis.) The partition $P_{t,c}$ contains the partial time swap mixers $U_{\text{TS},t,\{i,j\}}$ for which the edge $\{i, j\}$ is colored $c$. The full "time-color" mixer is

$$U_{\text{TS}}(\beta) = \prod_{t=0}^{h} \prod_{c=1}^{\kappa} U_{(t,c)\text{-TS}}(\beta), \tag{67}$$

where $U_{(t,c)\text{-TS}}$ is the product of the (mutually commuting) partial mixers in the part $P_{t,c}$.

---

### 4.3.2 Mapping and compilation

We use the absolute one-hot encoding, in which the ordering is encoded as a string using the absolute encoding and then the string is further encoded using the one-hot encoding. Specifically, we encode an ordering $\boldsymbol{\iota}$ into $\prod_i (h_i - 1) \leq nh$ qubits, where qubit $(i, t)$ indicates if job $i$ starts at time $t$:

$$|\boldsymbol{\iota}\rangle \mapsto \bigotimes_{i=1}^{n} \left[ |0\rangle^{\otimes(s_i - 1)} |1\rangle |0\rangle^{\otimes(h_i - s_i)} \right]. \tag{68}$$

**Phase separator.** The objective function is the weighted total tardiness:

$$f(\boldsymbol{\iota}) = \sum_{i=1}^{n} w_i \max\{0, d_i - s_i(\boldsymbol{\iota}) - p_i\}. \tag{69}$$

This yields the encoded phase Hamiltonian

$$H_{\mathrm{P}}^{(\mathrm{enc})} = \sum_{i=1}^{n} w_i \sum_{t=d_i - p_i + 1}^{h_i} (t + p_i - d_i) Z_{i,t}. \tag{70}$$

**Mixer.** The partial time-swap mixer

$$H_{\mathrm{TS},t,\{i,j\}} = |t + p_j, t\rangle \langle t, t + p_i|_{i,j}, \tag{71}$$

in the absolute one-hot encoding is equivalent to

$$H_{\mathrm{TS},t,\{i,j\}}^{(\mathrm{enc})} = S_{i,t+p_j}^{+} S_{j,t}^{+} S_{i,t}^{-} S_{j,t+p_i}^{-} + S_{i,t}^{+} S_{j,t+p_i}^{+} S_{i,t+p_j}^{-} S_{j,t}^{-}. \tag{72}$$

Using the time-color partitioned time-swap mixer, this corresponds to a circuit of $h\binom{n}{2}$ 4-qubit gates in depth $h\kappa$. $H_{\mathrm{TS},t,\{i,j\}}$ corresponding to edges $\{i,j\}$ colored $c$.

**Initial state.** We use an arbitrary ordering of the jobs as the initial state.

## 4.4 SMS, with release dates

**Problem.** $(1|d_j, r_j|*)$ Given a set of jobs with processing times $\mathbf{p}$, deadlines $\mathbf{d}$ release times $\mathbf{r} \in \mathbb{Z}_+$ and weights $\mathbf{w}$, find a schedule that minimizes some function of the tardiness, such that each job starts no earlier than its release time.

We now consider SMS with release dates $\{r_j\}$. We will not specify the objective function here, as any of those used in previous sections are still applicable. Our focus in the section is introducing a modification of the time swap mixer that preserves feasibility.

Let the horizon $h$ be some upperbound on the maximum completion time, e.g. $\max_j d_j + \sum_j p_j$. Let $b_j = h + \sum_{i=1}^{j-1} p_i$ be a special "buffer" time for job $j$. Let $W_j = [r_j, h - p_j]$ be the window of times in which job $j$ can start.

Consider the configuration space $\times_{j=1}^{n} [W_j \cup \{b_j\}]$, i.e. schedules $\{\mathbf{s}\}$ in which a job is scheduled either between its release time and the horizon or at its buffer time slot. The domain is the subset of the configuration space that satisfies the problem constraint that no two jobs overlap.

### 4.4.1 Partial Mixer: controlled-null-swap mixer

We now introduce a mixer, specifically a controlled null swap mixer used in graph coloring, 3.3. that preserves feasibility and avoids getting "stuck":

$$H_{\text{NS},i,t} = (|b_i\rangle \langle t|_i + |t\rangle |b_i\rangle) \prod_{j \neq i} \left( I_j - \sum_{t' \in \text{nbhd}_{i,t}(j)} |t'\rangle \langle t'|_j \right), \tag{73}$$

which corresponds to the unitary

$$U_{\text{NS},i,t}(\beta) = \Lambda_{\bar{\mathbf{s}}_{\text{nbhd}_{i,t}(j)}} \left( e^{-i\beta\left(|b_i\rangle \langle t|_i + |t\rangle \langle b_i|\right)} + \sum_{t' \in W_i \setminus \{t\}} |t'\rangle \langle t'| \right) \tag{74}$$

where

$$\text{nbhd}_{i,t}(j) = [\max\{t - p_j + 1, r_j\}, t + p_i - 1] \tag{75}$$

is the temporal "neighborhood" of job $i$ at time $t$ with respect to job $j$, i.e. the set of times at that starting job $j$ would conflict with job $i$, and $\mathbf{s}_{\text{nbhd}_{i,t}} = \left( (s_{j,t'})_{t' \in \text{nbhd}_{i,t}(j)} \right)_{j \neq i}$. The role of the buffer site is similar to the "uncolored" option in finding the maximal colorable induced subgraph in Sec. 3.3. Such mixing terms enables jobs to move freely in and out $[0, T_0]$ without inducing job-overlap, hence enables exploration of the whole feasible subspace. The controlled unitary notation $\Lambda_{\mathbf{y}}(Q)$ is defined in Eq. (18) in Sec. 3.2.

### 4.4.2 Encoding and compilation

Given the $\sum_{i=1}^{n} |W_i|$ partial mixers, one for each job $i$ and time $t \in W_i$, we can define simultaneous and sequential mixers as above. Using the one-hot encoding the partial mixer is

$$H_{\text{NS},i,t}^{(\text{enc})} = \left( \prod_{j \neq i} \prod_{t' \in \text{nbhd}_{i,t}(j)} \frac{1}{2}(I + Z_{j,t'}) \right) (X_{i,t} X_{i,b_i} + Y_{i,t} Y_{i,b_i}) . \tag{76}$$

The controlled null swap can be further compiled using ancilla qubits, as in Section 3.2. While in the cost of compilation could be bounded based the degree of the graph, the overlap of the various partial mixers may be more complicated (with respect to partitioning into disjoint sets) and expensive (with respect to number of gates and ancilla qubits) depending on the SMS instance.

**Objective function.** As an example objective function, we again consider minimizing the weighted total tardiness, Eq. (69). The one-hot encoded phase Hamiltonian takes the form of Eq. (70), with $b_j$ included in the summation range of $t$:

$$H_{\text{P}}^{(\text{enc})} = \sum_{i=1}^{n} w_i \left[ (b_i - d_i) Z_{i,b_i} + \sum_{t=d_i-p_i+1}^{h_i} (t + p_i - d_i) Z_{i,t} \right] \tag{77}$$

It can be implemented with $\sum_j (h_j - d_j + 1)$ single qubit $Z$-rotation gates.

**Initial state.** Any feasible schedule can be used as the initial state. In particular, we use a greedy earliest-release-date schedule. Assume without loss of generality that the jobs are ordered by their release times, i.e. $r_1 \leq r_2 \leq \cdots \leq r_n$. Then set $s_1 = r_1$ and recursively set $s_i = \max\{r_i, s_{i-1} + p_{i-1}\}$, which is feasible if likely suboptimal.

### 4.4.3 Mapping variants

In the construction above, each job $j$ is assigned a "buffer" time $b_j$, and a phase $b_j - d_j$ applied whenever that job is scheduled at its buffer time. The factor $b_j - d_j$ is arbitrary. Rather than consider schedules in which some jobs are at their buffer time, one could consider "partial schedules", in which a job is either scheduled at a time between 0 and $h$ or is in its buffer. The phase applied when a job is in its buffer need not be $b_j - d_j$ but in fact can be arbitrary, e.g. some common "buffer phase factor" $B$. In this case, we must define a scheme for associating each partial schedule with a canonical complete schedule, e.g. greedily starting the buffered jobs after those that are already scheduled. In this way, the states corresponding to partial schedules can still be considered as part of the domain.

## 5   Conclusions

We introduced a Quantum Alternating Operator Ansatz (QAOA), an extension of Farhi et al.'s Quantum Approximate Optimization Algorithm, and showed how to apply the ansatz to a diverse set of optimization problems with hard constraints. The essence of this extension is the consideration of general parameterized families of unitaries rather than only those corresponding to the time evolution of a local Hamiltonian, which allows it to represent a larger, and potentially more useful, set of states than the original formulation. In particular, refocusing on unitaries rather than Hamiltonians in the specification allows for more efficiently implementable mixing operators.

The original algorithm is already a leading candidate quantum heuristic for exploration on near-term hardware; our extension makes early testing on emerging gate-model quantum processors possible for a wider array of problems at an earlier stage. After formally introducing the ansatz, and providing design criteria for its components, we worked through a number of examples in detail, illustrating design techniques and exhibiting a variety of mixing operators. In the appendix, we provide a compendium of QAOA mappings for over 20 problems, organized by type of mixer.

While the approach of designing mixing operators to keep the evolution within the feasible subspace appears quite general, as illustrated by the wide variety of examples we have worked out, it is not universally applicable. Many of the problems in Zuckerman [79] have the form of optimizing a quantity within a feasible subspace consisting of the solutions to an NP-complete problem. Not only is an initial starting state (corresponding to one or more feasible solutions) hard to find, designing the mixing operator is also problematic. Even given a set of feasible solution to an NP-complete problem, it is typically computationally difficult to find another [61], making it difficult to design a mixer that fully explores the feasible subspace. The situation here is somewhat subtle, with it being easy to show in the case of SAT that finding a second solution when given a first remains NP-complete, but for Hamiltonian cycle on cubic graphs, given a first solution, a second is easy to find (but not a third). See [70, 76] for results on the complexity of "another solution problems (ASPs)." This difficulty in mapping Zuckerman's problems to QAOA further illustrates reasons for the difficulty of these approximate optimization problems.

While we have given basic design criteria for initial states, mixing operators, and phase-separation operators, we have barely scratched the surface in terms of which possibilities perform better than others. For most of the example problems, we discussed multiple mixers, coming from different partitions and orderings of partial mixers, different choices related to connectivity, or different numbers of repeats of operators. Analytical, numerical, and ultimately experimental work is required to understand which of these mixers are most

effective, and also to determine potential trade-offs with respect to robustness to noise or control error, efficiency of implementation, effectiveness of the mixing, and difficulty in optimizing the parameters. Similar questions arise with respect to choosing an initial state.

Effective parameter setting remains a critical, but mostly open, area of research. While for fixed $p$, brute-force search of parameter search space was proposed [22], it is practical only for small p; as $p$ increases, the parameter optimization becomes inefficient due to the curse of dimensionality, Guerreschi et al. [30] providing a detailed analysis. In certain simple [73] or highly symmetric cases [41], some insight into parameter setting for $p > 1$ has been obtained, but even in the simplest cases, understanding good choices of parameters seems non-trivial [73]. Improved parameter setting protocols may come from adapting techniques from existing control theory and parameter optimization methods, and by using insights gained from classical simulation of quantum circuits and experimentation on quantum hardware as it becomes available. In particular, classical simulation of the quantum circuits can take advantage of the local structure of the objective function (when it is indeed local) and the feasibility of classically simulating the measurement of $\log(n)$-qubits in a QAOA circuit by adapting results for IQP circuits [11].

To run on near-term quantum hardware, further compilation will be required. In many cases, we have left the compilation at the level of multiqubit operators that need to be further compiled to the gate set natively available on the hardware, most likely certain one- and two-qubit gates. Furthermore, near-term hardware will have additional restrictions, including which qubits each gate can be applied to, duration and fidelity of the gates, and cross-talk, among others. This necessitates additional compilation, especially to optimize success probability on pre-fault-tolerance devices. Other architectures, e.g. ones based on higher-dimensional qudits, may prompt other sorts of compilations as well. Recently, approaches for compiling circuits, including QAOA circuits, to realistic hardware have been explored [71], but that research direction remains open to innovation.

The robustness of these circuits to realistic error requires further exploration. In the near term, the question is how robust QAOA circuits are to realistic noise and how best to incorporate resource-efficient techniques to improve the robustness. In the longer term, the question becomes how to best incorporate the full spectrum of error correction techniques into this framework.

We expect some cross-fertilization between research on QAOA and research on AQO and quantum annealing, especially in the $p \gg 0$ regime. A particularly fruitful area of further study would be to build on the results of Yang et al. [75]. They used Pontryagin's minimization principle to show that a bang-bang schedule is (essentially) always optimal, providing support for QAOA, and gate-base approaches, generally. But the argument does not provide an efficient means to find such a schedule. Thus, it remains open in the AQO realm how to find effective schedules, with it currently being completely open as to whether allowing non-bang-bang schedules makes the finding of good schedule parameters easier. Similarly, exploiting certain structural commonalities with VQE is likely to be fruitful. In the small $p$ regime, we expect further cross-fertilization between QAOA and other models being considered for early quantum supremacy experiments, such as random circuits, boson sampling, and IQP circuits. Results for IQP circuits, whose close structural similarities with QAOA circuits has already provided insights [21], and whose advanced status with respect to error analysis and fault tolerance [12, 13] are likely to prove especially useful.

A number of extensions are possible. We briefly mention two classes. The first is hybrid methods. As one example, some other algorithm, either classical or quantum, could be run first in order to provide a good initial state, and QAOA then used to improve upon it; this

could be repeated several times with different initial states when the other algorithm is stochastic. Another example of a hybrid method is a parameter-setting protocol in which a classical algorithm uses the results of measurements during and after runs of a QAOA circuit to iteratively improve the parameters. A second class of extensions is versions of QAOA with many parameters. For example, we introduced mixing operators that are repeated applications of some basic mixer; each application could use a different parameter. The same could be done with mixers consisting of many partial mixers. Until we have a better grasp of parameter setting in the single parameter per mixing operator case, and for the effectiveness of the different mixers, it makes sense to restrict to the simple case. The one exception is to take advantage of the specific gates natively available, and to use essentially a VQE approach, as suggested in [24]. That approach makes excellent use of the capabilities of near-term hardware but may be more limited than single-parameter-per-phase QAOA in what it can tell us about parameter setting and the design of scalable quantum heuristics.

The biggest open question is the effectiveness of this approach as a quantum heuristic, and its potential impact in broadening the array of established applications of quantum computing. While obtaining further analytic results may be possible in some cases, in general we will have to try it out and see. Improved simulation techniques for quantum circuits, including potentially approaches tailored to QAOA circuits, can provide some insight, but the ultimate test will be experimentation on quantum hardware itself.

# 6 Acknowledgements

# A Compendium of mappings and compilations

We summarize QAOA mappings for a variety of problems. All problems considered in NPO [3, 69]. Problem names are compact versions of the names in Ausiello et al. [3] unless a different reference is given. Most of the mappings are new to this paper; for the exceptions, a reference is given, though in all such cases only H-QAOA mappings have been considered previously. Approximability results are taken from Ausiello et al. [3] unless otherwise mentioned.

We specify partial mixing Hamiltonians for each problem, which can then be used to define two types of full mixers, simultaneous mixers in H-QAOA that correspond to time evolution under the sum of the partial mixing Hamiltonians, and partitioned mixers (which are not in H-QAOA) that correspond to products of unitary operators defined by all the

partial mixers in an order specified by an ordered partition, possibly repeated. We will not specify the full mixers, since they are straightforward to derive from the partial mixers. For each problem, we provide a compilation with resource counts for at least one mixer or a class of mixers. In most cases, we mention only one partial mixers and compilations, though as we have seen, there are many possibilities for partial mixers, and for compilations of the operators.

The resource counts given are upper bounds. We have not worked to find optimal compilations. For the simpler problems that have the potential for implementation in the near term, we give exact resource bounds when we have such results, rather than only giving the complexity. We give bounds for the number of computational qubits required, and also for the number of ancilla qubits, when used. We give resource counts for the number of arbitrary two-qubit gates required to implement a given operator, merging single qubit operations into two-qubit gates before or after when possible. We compute depth for such circuits, which gives a lower bound on the depth on nearterm hardware, which will generally be higher due to topological constraints. In many cases, we will not compile all the way down to two-qubit gates, but will instead specify the resource count in terms of multiqubit operators which would then yield a two-qubit gate count and depth given compilations of those multiqubit mixers. Throughout, the initial state is always obtained by appling a constant-depth circuit to the state $|0\rangle^{\otimes *}$.

An algorithm $\mathcal{A}$ achieves an *approximation ratio* $r \leq 1$ if for all instances $x$ of a maximization problem it satisfies $\mathcal{A}(x)/\mathrm{OPT}(x) \geq r$, where $\mathrm{OPT}(x)$ is the optimal solution of the given instance. Both cases in which $r$ is constant and in which $r$ is a function of the problem parameters are considered. For minimization problems the approximation ratio is similarly defined with $r \geq 1$ and $\mathcal{A}(x)/\mathrm{OPT}(x) \leq r$. Unfortunately, there are multiple conventions in the literature, with the approximation ratio given as $1/r$ for minimization, maximization, or both. Fortunately, there is no ambiguity for a given value of $r$ to which convention convention is being used. Here, instead of being internally consistent, we will simply take the ratio as stated in the reference cited so as to facilitate easy comparison with the literature. All approximation results below concern efficient classical algorithms.

For all problems on a graph $G = (V, E)$, let $|V| = n$, $|E| = m$, and $D_G$ be the maximum vertex degree.

## A.1  Bitflip $(X)$ mixers

In this section we consider problems where all states of the configuration space are feasible, and hence the original QAOA construction may be used. Specifically, for all problems we can use the following:

**Variables:** $n$ binary variables.

**Initial state:** $|0\rangle^{\otimes n}$.

**Mixer:** The standard mixer $U_{\mathrm{M}}^{H}(\beta) = e^{-i\beta B}$
where $B = \sum_{j=1}^{m} X_j$, which can be implemented with depth 1. (Since all terms commute and act on separate qubits, we do not need to consider partitioned variants of the standard mixer.)

**Phase separator:** $U_{\mathrm{P}}(\gamma) = \exp[-i\gamma H_{\mathrm{P}}]$, we will specify $H_{\mathrm{P}}$ for each problem.

All problems in this section can be trivially extended to their weighted version by multiplying the terms of the phase Hamiltonian by the corresponding weights.

### A.1.1 Maximum cut

**Problem:** Given a graph $G = (V, E)$, find a subset $S \subset V$ such that the number of edges between $S$ and $V \setminus S$ is the largest.

**Prior QAOA work:** [22].

**Approximability:** APX-complete [45, 60]. NP-hard to approximate better than 16/17 [35]. Semidefinite programming acheives 0.8785 [27], which is optimal under the unique games conjecture [46]. On bounded degree graphs with $D_G \geq 3$ can be approximated to within $0.8785 + O(D_G)$ [25], in particular 0.921 for $D_G = 3$, but remains APX-complete [60].

**Configuration space, Domain, and Encoding:** $\{0, 1\}^n$, indicating whether each vertex is in $S$ or not.

**Objective:** $\max |\{\{u, v\} \in E : u \in S, v \notin S\}|$.

**Phase separator**: $H_{\mathrm{P}} = \sum_{\{u,v\} \in E} Z_u Z_v$.

**Resource count for phase separator:** $m$ gates with depth at most $D_G + 1$.

**Variant:** Directed-MaxCut (Max-Directed-Cut), where we seek to maximize the number of the *directed* edges leaving $S$. The phase separator needs to be replaced by $H_{\mathrm{P}} = \sum_{(u,v) \in E}(Z_u - Z_v + Z_u Z_v)$.

### A.1.2 Max-$\ell$-SAT

**Problem:** Given $m$ disjunctive clauses over $n$ Boolean variables $\mathbf{x}$, where each clause contains at most $\ell \geq 2$ literals, find a variable assignment that maximizes the number of satisfied clauses. Let $\mathbf{x}_\alpha$ be the variables involved in clause $\alpha$ and $C_\alpha(\mathbf{x}_\alpha) = 1$ (0) if the state of $\mathbf{x}_\alpha$ (resp., does not) satisfies the clause.

**Prior QAOA work:** [74].

**Approximability:** APX-complete [60]. The best classical approximation ratio for Max-2-Sat is 0.940 [50], which cannot be improved beyond 0.943 under the unique games conjecture [46], or beyond 0.954 unless $P = NP$ [36]. For Max-3-Sat, an efficient 7/8-approximation [44] is known, which is optimal unless P=NP [36]. Remains APX-complete when any literal appears in at most 3 clauses [3].

**Objective:** $\max \sum_{j=\alpha}^m C_\alpha(\mathbf{x}_\alpha)$.

**Phase separator:** $H_{\mathrm{P}} = \sum_{\alpha=1}^m H_{C_\alpha(\mathbf{x}_\alpha)}$.

**Resource count for phase separator:** Can be implemented with $\binom{\ell}{k}$ many $k$-local $ZZ \cdots Z$ gates, for $k = 1, \ldots, \ell$, overall requiring $O(m2^\ell)$ 2-qubit gates.

### A.1.3 Min-$\ell$-SAT

**Problem:** The same as Max-$\ell$-SAT, except with minimization instead of maximization of the number of satisfied clauses.

**Approximability:** APX-complete [47]. Unless $P = NP$, no classical algorithm can do better than 1.3606 for MinSAT [19], or, for Min-$\ell$-SAT, better than $15/14 \simeq 1.0714$ for $k = 2$ or $7/6 \simeq 1.1667$ for arbitrary $\ell$ [4]. On the other hand, constructive algorithms are known which achieve approximatio ratios of 1.1037, 1.2136, and $1 - 2^{1-\ell}$ for Min-2-SAT, Min-3-SAT, and Min-$\ell$-SAT, respectively [4, 8].

**Mapping:** The same as Max-$\ell$-SAT.

### A.1.4 Max-Not-All-Equal-$\ell$-SAT (NAE-$\ell$-SAT)

**Problem:** The same as in Max-$\ell$-SAT, except that in NAE-$\ell$-SAT each clause is satisfied only if all $\ell \geq 3$ variables do not all have the same value.

**Approximability:** APX-complete [60]. A classical 1.38-approximation algorithm is known [2]. For $\ell = 3$ approximable to 1.138 but no better than 1.090 [3, 81].
**Mapping:** Same as Max-$\ell$-SAT.
**Variant:** Similar considerations apply to Max-1-in-$\ell$-Sat, in which a clause is satisfied when exactly one of its variables is true.

### A.1.5    Set Splitting

**Problem:** Given a set $\mathcal{S}$ and a collection of subsets $\{S_j\}_{j=1}^{m}$, seek a partition $\mathcal{S} = \mathcal{S}_1 \cup (\mathcal{S} \setminus \mathcal{S}_1)$ that maximizes the number of split sets, i.e. $S_j$ with elements in both $\mathcal{S}_1$ and $\mathcal{S} \setminus \mathcal{S}_1$.
**Approximability:** APX-complete [63]. Can be approximated to 0.7499 [78]. Remains APX-complete if each $S_j$ is restricted to have at most or exactly $k \geq 2$ elements [51]. For each $S_j$ having exactly $k \geq 4$ elements, unless P=NP there is no efficient classical algorithm that does essentially better than a random partition [31, 36]. The generalization MaxHypergraphCut, in which each subset is given a weight and we seek to maximize the total weight of the split sets, can be approximated to 0.7499 [78].
**Reduction to Not-All-Equal-$\ell$-SAT:** This problem is a special case of Not-All-Equal-$\ell$-SAT where none of the variable are negated.

### A.1.6    MaxE3LIN2

**Problem:** Given a set of $m$ three-variable equations $\mathcal{A} = \{A\}$, over $n$ binary variables $\mathbf{x} \in \{0, 1\}^n$, where each equation $A_j$ is of the form $x_{a_{1,j}} + x_{a_{2,j}} + x_{a_{3,j}} = b_j \mod 2$ where $a_{1,j}, \ a_{2,j}, \ a_{3,j} \in [n]$ and $b_j \in \{0, 1\}$, find an assignment $\mathbf{x} \in \{0, 1\}^n$ that maximizes the number of satisfied equations.
**Prior QAOA work:** [23].
**Approximability:** No efficient $1 + \epsilon$ classical approximation algorithm unless P=NP.[35]
**Objective:** number of equations satisfied.
**Phase Separator**: $H_{\mathrm{P}} = \sum_{j=1}^{m} H_j$ where $H_j = (-1)^{b_j} Z_{a_{1,j}} Z_{a_{2,j}} Z_{a_{3,j}}$ .
**Resource count for phase separator:** Can be implemented with $m$ many 3-qubit gates.

## A.2    Controlled-bit-flip ($\Lambda_f(X)$) mixers

All problems in this subsection are graph or subset problems with QAOA mappings that use controlled-bit-flip mixers on bit strings. (See Sec. 3.2.) Note that MaxSetPacking (A.2.4) reduces to MaxIndependentSet (A.2.1) on the constraint graph; MinVertexCover (A.2.3) is a special case of MinSetCover (A.2.5) in which the universe is the set of edges and each subset in the collection corresponds to the edges adjacent to a single vertex; and MaxClique (A.2.2) is simply MaxIndependentSet on the complement graph.
**Configuration space:** The configuration space of each problem is the set of subsets $V'$ of some set $V$, represented by a bitstring.
**Constraint graph:** An instance of each problem is either specified by a graph or has a natural corresponding constraint graph whose vertices correspond to the variables and with respect to which each variable $v$ has a neighborhood $\mathrm{nbhd}(v)$.
**Domain:** The domain is the subset of the configuration that satisfies some CNF formula (whose clauses correspond to the edges of the problem or constraint graph, except for Min-SetCover).
**Objective:** The cardinality of the subset.
**Mixing rule:** Swap an element $v$ in or out of $V'$ if some predicate $\chi(\mathbf{x}_{\mathrm{nbhd}(v)})$ is satisfied

by the partial state of its neighbors. (The predicate $\chi$ will depend on the problem.)

**Partial mixing Hamiltonian:** $H_{\mathrm{CX},v} = X_v H_{P_v}$, which can be used to define both a simultaneous controlled-bit-flip mixer and a class of partitioned controlled-bit-flip mixers. (See Sec. 3.2).

### A.2.1 MaxIndependentSet [See Sec. 3.2]

**Problem:** Given $G = (V, E)$, maximize the size of a subset $V' \subset V$ of mutually non-adjacent vertices.

**Prior QAOA work:** [22]. Also, see Sec. 3.2 for a detailed discussion of the mapping.

**Approximability:** Poly-APX-complete [7], and has no constant factor approximation unless P = NP. On bounded degree graphs with maximum degree $D_G \geq 3$ can be approximated to $(D_G + 2)/3$ [7], but remains APX-complete [60].

**Configuration space:** All subsets $V'$ of $V$, represented by elements $\mathbf{x}$ of $\{0,1\}^n$, with $x_v = 1$ indicating $v \in V'$.

**Domain:** Independent sets, $\left\{ \mathbf{x} : \bigwedge_{\{u,v\} \in E} \bar{x}_u \vee \bar{x}_v \right\}$.

**Objective:** $\max |V'| = \sum_{v \in V} x_v$.

**Mixing rule:** Swap $v$ in or out of $V'$ if none of its neighbors are in $V'$.

**Partial mixing Hamiltonian:** $H_{\mathrm{CX},v} = X_v H_{\mathsf{NOR}(x_{\mathrm{nbhd}(v)})}$.

**Phase separator:** $U_P(\gamma) = \exp(-i\gamma \sum_{u \in V} Z_u)$

**Initial state:** $|s\rangle = |0\rangle^{\otimes n}$, i.e. $V' = \emptyset$.

**Resource count:**
- **Controlled-bit-flip mixers:** $n$ multiqubit-controlled-$X(\beta)$ gates, each with at most $D_G$ controls (exactly $D_v$ controls for each vertex). Depth at most $n$, but will be much less for sparsely connected graphs.
- **Phase separator:** $n$ single-qubit $Z$-rotations. Depth 1.

### A.2.2 MaxClique

**Problem:** Given $G = (V, E)$, maximize the size of a clique (a subset $V' \subset V$ that induces a subgraph in which all pairs of vertices are adjacent).

**Approximability:** Cannot be classically approximated better than $O(n^{1-\epsilon})$ for any $\epsilon$ unless $P = NP$ [80].

**Configuration space:** All subsets $V'$ of $V$, represented by elements $\mathbf{x}$ of $\{0,1\}^n$, with $x_v = 1$ indicating $v \in V'$.

**Domain:** Cliques, $\left\{ \mathbf{x} : \bigwedge_{\{u,v\} \in E(\overline{G})} \bar{x}_u \vee \bar{x}_v \right\}$.

**Objective:** $\max |V'| = \sum_{v \in V} x_v$.

**Reduction to MaxIndependentSet:** Every clique on $G = (V, E)$ gives on independent set on the complement graph $\overline{G} = (V, E(\overline{G}))$ where $E(\overline{G}) = \binom{V}{2} \setminus E$. Therefore, a mapping for MaxClique is given by the mapping for MaxIndependentSet applied to the complement graph $\overline{G}$.

**Resource count:** The same resources as for MaxIndependentSet, except that the controlled bit flip mixers are now multiqubit-controlled-X gates with at most $n - D_G - 1$ controls (exactly $n - D_v - 1$ controls for each vertex $v$).

**Variants:** Extends easily to weighted versions MaxIndependentSet, with objective functions $f = \sum_{i=1}^{n} w_i x_i$.

### A.2.3   MinVertexCover

**Problem:** Given $G = (V, E)$, minimize the size of a subset $V' \subset V$ such that for every $(u, v) \in E$, $u \in V'$ or $v \in V'$.

**Approximability:** APX-complete [60]. Has a $2 - \Theta(1/\sqrt{\log n})$-approximation [43], but cannot be approximated better than 1.3606 unless P=NP [18].

**Configuration space:** All subsets $V'$ of $V$, represented by elements $\mathbf{x}$ of $\{0, 1\}^n$, with $x_v = 1$ indicating $v \in V'$.

**Domain:** Vertex covers, $\left\{ \mathbf{x} : \bigwedge_{\{u,v\} \in E(\overline{G})} x_u \vee x_v \right\}$.

**Objective:** $\min |V'|$.

**Initial state:** $|s\rangle = |1\rangle^{\otimes n}$, i.e. $V' = V$.

**Mixing rule:** Swap $v$ in or out of $V'$ if all of the edges incident to $v$ are covered by $V' \cap \mathrm{nbhd}(v)$.

**Phase Separator:** $U_\mathrm{P}(\gamma) = \exp(-i\gamma \sum_{u \in V} Z_u)$.

**Reduction to MaxIndependentSet:** A subset $V' \subset V$ is a vertex cover if and only if $V \backslash V'$ is an independent set, so the problem of finding a minimum vertex cover is equivalent to that of finding a maximum independent set. While as approximation problems they are not equivalent [69], we can use the same mapping as for MaxIndependentSet with each $\bar{x}_v$ replaced by $x_v$. The resource counts are the same as for MaxIndependentSet.

**Reduction to MinSat:** Maranthe et al. [54] give an approximation-preserving reduction to Min-$D_G$-SAT enabling us to use the QAOA constuction given above for MinSat. The resource counts are the same as for MinSat with $m$ variables and $n$ clauses.

### A.2.4   MaxSetPacking

**Problem:** Given a universe $[n]$ and $m$ subsets $\mathcal{S} = (S_j)_{j=1}^n$, $S_j \subset [n]$, find the maximum cardinality subcollection $\mathcal{S}' \subset \mathcal{S}$ of pairwise disjoint subsets.

**Approximability:** As difficult as MaxClique [3]. Cannot be approximated within any constant factor [37]. The best known algorithm gives a $O(\sqrt{n})$-approximation [34]. APX-complete with the restriction $|S_j| \le k$ (Max-$k$-Set-Packing).

**Configuration space:** All subsets $\mathcal{S}'$ of $\mathcal{S}$, represented by elements $\mathbf{x}$ of $\{0, 1\}^m$, with $x_j = 1$ indicating $S_j \in \mathcal{S}'$.

**Constraint graph:** Vertices $V = [m]$ corresponding to elements of $\mathcal{S}$, edges $E$ corresponding to pairs of intersecting subsets.

**Domain:** Subcollections containing mutually disjoint subsets, $\left\{ \mathbf{x} : \bigwedge_{\{i,j\} \in E} \bar{x}_i \vee \bar{x}_j \right\}$.

**Objective:** $\max |\mathcal{S}'| = \sum_{j=1}^m x_j$

**Initial state:** $|s\rangle = |0\rangle^{\otimes m}$, i.e. $\mathcal{S}' = \emptyset$.

**Mixing rule:** Swap $S_j$ in or out of $\mathcal{S}'$ if $S_j$ is disjoint from the other subsets in $\mathcal{S}'$, i.e. $\mathbf{x}_{\mathrm{nbhd}(j)} = \mathbf{0}$.

**Partial mixing Hamiltonian:** $H_{\mathrm{CX},j} = 2^{-D_j} X_j \prod_{i \in \mathrm{nbhd}(j)} (I + Z_i) = X_j H_{\mathsf{NOR}(x_{\mathrm{nbhd}(j)})}$.

**Phase separator:** $U_\mathrm{P}(\gamma) = \exp(-i\gamma \sum_{j=1}^m Z_j)$.

**Resource count:**

- **Controlled-bit-flip mixers:** Each partial mixer $H_{\mathrm{CX},j}$ is implemented as a controlled-$R_X$ gate with $D_j$ control qubits. Partial mixer depth at most $m$.
- **Phase separator:** $m$ single-qubit $Z$-rotations. Depth 1.
- **Initial state:** Depth 0.

**Variants**: Also called MaxHypergraphMatching. Equivalent to MaxClique under a $PTAS$-reduction [3].

### A.2.5 MinSetCover

**Problem:** Given a universe $[n]$ and $m$ subsets $\mathcal{S} = (S_j)_{j=1}^n$, $S_j \subset [n]$, find the minimum cardinality subcollection $\mathcal{S}' \subset \mathcal{S}$ of the $S_j$ such that their union recovers $[n]$.

**Approximability:** There exists a $1 + \ln n$ algorithm [42]. Cannot be efficiently approximated to $(1 - o(1)) \ln n$ unless P=NP [20]. APX-complete with the restriction $|S_j| \leq k$ (Max-$k$-Set-Cover) [3].

**Configuration space:** All subsets $\mathcal{S}'$ of $\mathcal{S}$, represented by elements $\mathbf{x}$ of $\{0,1\}^m$, with $x_j = 1$ indicating $S_j \in \mathcal{S}'$.

**Domain:** Set covers, $\left\{ \mathbf{x} : \bigcup_{j \in [m]:x_j = 1} S_j = [n] \right\}$.

**Constraint graph:** Vertices $[m]$ corresponding to subcollections in $\mathcal{S}$; two vertices $\{i, j\}$ are adjacent if and only if their corresponding sets interset, $S_i \cap S_j \neq \emptyset$.

**Objective:** $\min |\mathcal{S}'| = \sum_{j=1}^m x_j$.

**Initial state:** $|s\rangle = |1\rangle^{\otimes m}$, i.e. $\mathcal{S}' = \mathcal{S}$.

**Mixing rule:** Swap set $S_j$ in or out of $\mathcal{S}'$ if $\mathcal{S}' \setminus S_j$ covers $[n]$.

**Partial mixing Hamiltonian:**

$$H_{\text{CX},j} = \left( \prod_{i=1}^n \left( I - \prod_{\ell \in [m]:\ell \neq j, i \in S_\ell} \frac{I + Z_l}{2} \right) \right) X_j = X_j H_{\chi_{\text{MSC},j}}, \tag{78}$$

where $\chi_{\text{MSC},j}(\mathbf{x}_{\text{nbhd}(j)}) = \bigwedge_{i \in S_j} \bigvee_{\ell \in \text{nbhd}(j):i \in S_\ell} x_\ell$.

**Phase separator:** $U_{\text{P}}(\gamma) = \exp(-i\gamma \sum_{j=1}^m Z_j)$.

**Resource count:**
- **Controlled-bit-flip mixers:** For each $H_{\text{CX},j}$, use $|S_j|$ ancilla qubits. Use each ancilla qubit $i$ to compute $\bigvee_{\ell \in \text{nbhd}(j):i \in S_\ell} x_\ell$ using a controlled NOT gate with $|\{\ell \in \text{nbhd}(j) : i \in S_\ell\}| \leq |\text{nbhd}(j)| = D_j$ control qubits. Then implement $H_{\text{CX},j}$ using a controlled X gate on qubit $j$ with the $|S_j|$ ancilla qubits as the control. Finally, uncompute the ancilla qubits using the same $|S_j|$ controlled NOT gates as in the first step. Depth at most $2D_j + 1$ per partial mixer.
- **Phase separator:** $m$ single-qubit $Z$-rotations. Depth 1.
- **Initial state:** Depth 0.

**Variants:** Equivalent to Minimum Hitting Set [3] and under $L$-reductions equivalent to Minimum Dominating Set (which is a special case of Minimum Set Cover).

## A.3 $XY$ mixers

### A.3.1 Max-$\kappa$-ColorableSubgraph [See Sec. 3.1]

**Problem:** Given a graph $G$ and $\kappa$ colors, maximize the size (number of edges) of a properly colored subgraph.

**Approximability:** A random coloring properly colors a fraction $1 - 1/k$ of edges in expectation. Equivalent to MaxCut for $k = 2$. For $k > 2$, semidefinite programming gives a $(1 - 1/k + (2 + o(k))\frac{\ln k}{k^2})$-approximation [26], which is optimal up to the $o(k)$ factor under the unique games conjecture [46]. APX-complete for $k \geq 2$ [60] and no PTAS unless P=NP [63].

**Configuration space and domain:** The set of all colorings with at most $\kappa$ colors, $\{\mathbf{x} \in [\kappa]^n\}$.

**Objective:** $\max \sum_{\{u,v\} \in E} \text{NEQ}(x_u, x_v)$

**Initial state:** All vertices colored with color 1, $|1\rangle^n$.

**Partial Mixing Hamiltonian:** $H_{r\text{-NV}} = \sum_{a=1}^{r} \left( \breve{X}^a + (\breve{X}^\dagger)^a \right)$.

**Encoding:** One-hot.

- **Phase Separator:** $U_{\mathrm{P}}(\gamma) = \prod_{\{u,v\}\in E} \prod_{j=1}^{\kappa} \exp\left(-i\gamma Z_{v,j} Z_{v,j}\right)$
- **Resource count:**
  - **Number of qubits:** $n\kappa$.
  - **Parity ring mixer:** $n\kappa$ two-qubit $(XY)$ gates, with depth at most 2 ($\kappa$ even) or 3 ($\kappa$ odd).
  - **Phase separator:** $n\kappa$ two-qubit $(ZZ)$ gates. Depth at most $D_G + 1$.
  - **Initial state:** $n$ single-qubit $X$ gates. Depth 1.

**Alt. Encoding (for $\kappa = 2^l$):** Binary

- **Phase separator:** $U_{\mathrm{P}}(\gamma) = \prod_{\{u,v\}\in E} \Lambda_{\mathsf{EQx}_u,\mathbf{v}}\left(e^{-i\gamma}\right)$
- **Mixer:** $U_{\mathrm{M}}(\beta) = \prod_{i=1}^{n} e^{-i\beta X_{i,0}} \mathrm{ADD}_i(1)^\dagger e^{-i\beta X_{i,0}} \mathrm{ADD}_i(1)$. where $\mathrm{ADD}_i(z)$ adds $z$ to the register $i$ encoding an integer in binary.
- **Resource count:**
  - **Parity ring mixer:** $2n$ ADDs, $2n$ single qubit rotations; completely parallelizable.
  - **Phase separator:** $m$ controlled-phase gates with $2l$ controls; depth $O(D_G)$.
  - **Initial state:** $n$ single-qubit $X$ gates in depth 1.

**Variants:** Also known as Max-$\kappa$-Cut [26, 46], sometimes with weighted edges. For the weighted case same approximability results apply.

### A.3.2  Graph Partitioning (Minimum Bisection)

**Problem:** Given a graph $G$ such that $n$ is even, find a subset $V_0 \subset V$ satisfying $|V_0| = n/2$ such that the number of edges between $V_0$ and $V \setminus V_0$ is minimized.

**Prior AQO work:** Studied in the context of AQO for constrained optimization [39].

**Approximability:** An efficient $O(\log^{1.5} n)$-approximate algorithm is known [48].

**Configuration space, domain, and encoding:** Bit string $\mathbf{x}$ of Hamming weight $n/2$.

**Objective**: $\min \sum_{\{uv\}\in E} \mathsf{NEQ}(x_u, x_v)$.

**Initial state:** Any $\mathbf{x}$ of Hamming weight $n/2$.

**Phase separator**: $U_{\mathrm{P}}(\gamma) = \exp(-i\gamma \sum_{\{uv\}\in E} Z_u Z_v)$.

**Partial mixing Hamiltonian:** Qubit ring XY mixer $H_{\mathrm{ring}}^{(\mathrm{enc})} = \sum_{u=1}^{n} X_u X_{u+1} + Y_u Y_{u+1}$.

**Resource count:**

- **Number of qubits:** $n$.
- **Parity ring mixer:** $n$ two-qubit (XX+YY) gates, with depth at most 2 ($n$ even) or 3 ($n$ odd).
- **Phase separator:** $m$ two-qubit (ZZ) gates, with depth no more than $D_G + 1$.
- **Initial state:** $n/2$ single-qubit $X$ gates. Depth 1.

### A.3.3  Maximum Bisection

**Problem:** Given a graph $G$ such that $n$ is even, and edge weights $w_j$, find a subset $V_0 \subset V$ satisfying $|V_0| = n/2$ such that the total weight of edges crossing between $V_0$ and $V \setminus V_0$ is maximized.

**Approximability:** A random bisection gives an 0.5-approximation in expectation, improved to 0.65 [26].

**Mapping:** The same as for Graph Partitioning in Sec. A.3.2 with weights included in the phase separator.

### A.3.4 Maximum Vertex $k$-Cover

**Problem**: Variant of Vertex Cover optimization problem. Given a graph $G$ and an integer $k \leq n$, find a subset $V_0 \subset V$ of size $|V_0| = k$ such that the number of edges covered by $V_0$ is maximized.

**Approximability:** It is NP-hard to decide whether or not a fraction $(1 - \epsilon)$ of the edges can be $k$-covered [63].

**Mapping:** The same as for Graph Partitioning in Sec. A.3.2 with the Hamming weight $n/2$ replaced by $k$.

## A.4 Controlled-$XY$ mixers

### A.4.1 Max-$k$-ColorableInducedSubgraph (Sec.3.3)

**Problem:** Given a graph $G$ and $\kappa$ colors, maximize the size of a subset of vertices $V' \subset V$ whose induced subgraph is $\kappa$-colorable.

**Approximability:** Equivalent to MaxIndependentSet for $k = 1$. Both as easy and as hard to approximate as MaxIndependentSet for $k \geq 1$ [3, 59]. On bounded degree graphs can be approximated to $(D_G/k + 1)/2$ [33], but remains APX-complete.

**Configuration space:** $[\kappa + 1]^n$. (0-th color is "uncolored" and represents $v \notin V'$.)

**Domain:** $\kappa$-colorable induced subgraphs, $\left\{ \mathbf{x} : \bigwedge_{\{i,j\} \in E} \left( \mathsf{EQ}(x_i, 0) \vee \mathsf{EQ}(x_j, 0) \vee \mathsf{NEQ}(x_i, x_j) \right) \right\}$.

**Objective:** $\min \sum_{i=1}^n \mathsf{NEQ}(x_i, 0)$

**Initial state:** All vertices uncolored, $|10\ldots0\rangle^{\otimes n}$.

**Mixing rule:** A vertex can switch between being uncolored and colored $j$ only if none of its neighbours are colored $j$.

**Partial mixing Hamiltonian:** Controlled null swap mixer at a vertex.

**Encoding:** One hot.

**Phase separator:** $U_\mathrm{P}(\gamma) = \exp(\frac{i}{2}\gamma \sum_{u \in V} \sum_{j=1}^\kappa Z_{u,j})$.

**Resource count:**
- **Partitioned controlled null-swap mixers:** $n\kappa$ partial mixers, each acting on at most $D_G + 1$ qubits. Depth at most $n\kappa$, but will be much less for sparsely connected graphs.
- **Phase separator:** $n$ single-qubit $Z$-rotations. Depth 1.
- **Initial state:** $|s\rangle = \left( |1\rangle \otimes |0\rangle^{\otimes \kappa} \right)^{\otimes n}$, implemented in depth 1 with $n$ $X$ gates.

### A.4.2 MinGraphColoring (Sec. 3.4)

**Problem:** Given a graph $G$, minimize the number of colors required to properly color it.

**Approximability:** The best classical algorithm [32] achieves approximation ratio $O(n\frac{(\log\log n)^2}{\log^3 n})$, and we cannot do better than $O(n^{1-\epsilon})$ for any $\epsilon > 0$ unless P=NP [80]. For edge-colorings of multigraphs, there is a $(1.1 + 0.8/\kappa^*)$-approximate algorithm [57].

**Configuration space:** $[\kappa]^n$, $\kappa = D_G + 2$.

**Domain:** Proper $\kappa$-colorings of $G$ (many of which use fewer than $\kappa$ colors), $\left\{ \mathbf{x} : \bigwedge_{\{i,j\} \in E} \mathsf{NEQ}(x_i, x_j) \right\}$.

**Objective:** Minimize number of used colors: $\sum_{a=1}^\kappa \mathsf{OR}(\mathsf{EQ}(x_1, a), \ldots, \mathsf{EQ}(x_n, a))$.

**Mixing rule:** The color of vertex $u$ may be swapped between colors $c$ and $c'$ if none of its neighbours are already colored $c$ or $c'$.

**Partial mixing Hamiltonian:** Controlled-swap partial mixing Hamiltonian.

**Encoding:** One-hot.

**Phase Separator:** $\prod_{a=1}^\kappa \Lambda_{\mathsf{OR}(\mathbf{x}_{[n],a})} \left( e^{-i\gamma} \right)$.

**Resource count:**

- **Partitioned controlled-swap mixers:** $\kappa(\kappa - 1)n/2$ controlled gates on no more than $D_G + 2$ qubits.
- **Phase separator:** $\kappa$ partial phase separators acting on $n + 1$ qubits, one target qubit and $n$ control qubits. Depth 2 in partial phase separators, or depth 1 with the addition of $\kappa$ ancilla qubits.
- **Initial state:** Any valid $\kappa$ coloring (can be efficiently computed classically). Can be implemented in depth 1 using $n$ single-qubit $X$ gates.

**Reduction from MinEdgeColoring:** In MinEdgeColoring, the objective is to minimize the number of colors need to color the edges so that no two adjacent edges have the same color. This is equivalent to MinGraphColoring on the line graph.

### A.4.3 MinCliqueCover

**Problem:** Given a graph $G$, we seek the smallest collection of cliques $S_1, \ldots, S_k \subset V$, such that every vertex belongs to at least one clique.

**Approximability:** If MaxClique is approximable within $f(n)$ for a given instance then MinCliqueCover approximable to $O(f(n))$ [3]. Not approximable within $n^\epsilon$ for any $\epsilon > 0$ [53].

**Reduction to MinGraphColoring:** A partition of the vertices of $G$ is a $k$-clique cover if and only if it is a proper $k$-coloring of the complement graph $G' = (V, E^c)$, and moreover the smallest clique cover corresponds to the chromatic number of the complement graph. Thus the previous construction suffices.

## A.5 Permutation mixers

### A.5.1 TravelingSalespersonProblem (TSP) [see Sec. 4.1]

**Problem:** Given a set of $n$ cities and distances $d : [n]^2 \to \mathbb{R}_+$ (with $d(i, i) = 0$), find an ordering of the cities that minimizes the total distance traveled on the corresponding tour.

**Approximability:** NPO-complete [58]. MetricTSP is APX-complete [62] and has $3/2$-approximation [16]. The corresponding MaxTSP problem is approximable within $7/5$ for symmetric distance, and $63/38$ if asymmetric.

**Configuration space and domain**: Orderings of the cities $\{\mathbf{i}\}$.

**Objective:** $f(\boldsymbol{\iota}) = \sum_{j=1}^n d_{\iota_j, \iota_{j+1}}$.

**Partial mixer:** Partial permutation swap mixer.

**Encoding:** Direct one-hot.

**Compilation:**
- **Phase separator:** $H_{\mathrm{P}}^{(\mathrm{enc})} = \sum_{i=1}^n \sum_{u=1}^n \sum_{v=1}^n d(u, v) Z_{u,i} Z_{v,i+1}$.
- **Partial mixer:** $U_{\mathrm{PS}, \{i,j\}, \{u,v\}}^{(\mathrm{enc})}(\beta) = e^{-i\beta H_{\mathrm{PS}, \{i,j\}, \{u,v\}}^{(\mathrm{enc})}}$, where $H_{\mathrm{PS}, \{i,j\}, \{u,v\}}^{(\mathrm{enc})}$ is given in Eq. (55).

**Initial state:** Arbitrary ordering.

**Resource count:**
- **Color-parity permutation swap mixer (Sec. 4.1):** at most $(n-1)\binom{n}{2}$ 4-qubit partial mixers, in depth at most $2n$.
- **Phase separator:** $n^2(n-1)$ mutually commuting 2-qubit gates. Depth no more than $D_G + 1$.
- **Initial state:** $n$ single-qubit $X$ gates. Depth 1.

### A.5.2 SMS, minimizing total weighted squared tardiness (Sec. 4.2)

**Problem:** $(1|d_j| \sum w_j T_j^2)$ Given a set of jobs with processing times $\mathbf{p}$, deadlines $\mathbf{d} \in$

$\mathbb{Z}_+$, and weights $\mathbf{w}$, find a schedule that minimizes the total weighted squared tardiness $\sum_{j=1}^n w_j T_j^2$.

**Approximability:** Considered in [67].

**Configuration space and domain:** Orderings of the jobs $\{\mathbf{i}\}$, and an integer slack variable $y_j \in [0, d_j - p_j]$ for each job.

**Objective:** $f(\iota, \mathbf{y}) = \sum_{j=1}^n w_j (s_j(\iota) + p_j - d_j + y_j)^2$.

**Partial mixer:** Partial permutation swap mixer for computational qubits; binary mixer mixer for the slack qubits.

**Encoding:** Direct one-hot for the ordering variables; binary for the slack variables.

**Compilation:**

- **Phase separator:** The encoded phase separator is a 3-local Hamiltonian containing
    - all 1-local terms;
    - all 2-local terms of two computational qubits corresponding to different jobs at different places in the ordering, all 2-local terms of two ancilla qubits corresponding to the same job, and all 2-local terms of one computational qubit and one ancilla qubit except when they correspond to different jobs and the computational qubit corresponds to that job being last in the ordering;
    - all 3-local terms of three computational qubits corresponding to different jobs at different places in the ordering and all 3-local terms containing two computational qubits corresponding to different jobs at different places in the ordering and one ancilla qubit corresponding to the later job.
- **Partial mixer:** $H_{\mathrm{PS}, \{i,j\}, \{u,v\}}^{(\mathrm{enc})} = S_{u,i}^+ S_{v,j}^+ S_{u,j}^- S_{v,i}^- + S_{u,i}^- S_{v,j}^- S_{u,j}^+ S_{v,i}^+$.
- **Initial state** Arbitrary ordering.

**Resource count:**

- **Color-parity permutation swap mixer (Sec. 4.1):** at most $(n-1)\binom{n}{2}$ 4-qubit partial mixers, in depth at most $2n$. Single-qubit $X$ mixer for slack binary variables can be done in parallel with the permutation swap mixer.
- **Phase separator:** Let $\mu_i = \lceil \log_2(d_i - p_i + 1) \rceil$ be the number of bits needed for the slack variable $y_i$, and $\mu = \sum_{i=1}^n \mu_i$.
    - Number of 1-local gates: $n^2 + \mu$.
    - Number of 2-local gates: $2\binom{n}{2}^2 + \sum_{i=1}^n \binom{\mu_i}{2} + \mu(n^2 - n + 1)$.
    - Number of 3-local gates: $6\binom{n}{3}^2 + 2\mu\binom{n}{2}^2$.
- **Initial state:** $n$ single-qubit $X$ gates. Depth 1.

### A.5.3   SMS, minimizing total weighted tardiness [See Sec. 4.3]

**Problem:** $(1|d_j|\sum w_j T_j)$ Given a set of $n$ jobs with processing times $\mathbf{p}$, deadlines $\mathbf{d} \in \mathbb{Z}_+$, and weights $\mathbf{w}$, find a schedule that minimizes the total weighted tardiness $\sum_{j=1}^n w_j T_j$.

**Approximability:** There is an $(n-1)$-approximate algorithm [14]. The decision version is strongly NP-hard [49].

**Configuration space and domain:** Orderings of the jobs $\{\mathbf{i}\}$.

**Objective:** $f(\iota) = \sum_{i=1}^n w_i \max\{0, d_i - s_i(\iota) - p_i\}$.

**Encoding:** Absolute one-hot.

**Mixing rule:** Swap two jobs only if they are scheduled in consecutive order.

**Partial mixer:** Partial time-swap mixer (specific to the absolute encoding).

**Initial state:** Arbitrary ordering.

**Compilation:**

- **Phase separator:** $H_{\mathrm{P}}^{(\mathrm{enc})} = \sum_{i=1}^n w_i \sum_{t=d_i-p_i+1}^{h_i} (t + p_i - d_i) Z_{i,t}$.
- **Partial mixer:** $H_{\mathrm{TS}, t, \{i,j\}}^{(\mathrm{enc})} = S_{i,t+p_j}^+ S_{j,t}^+ S_{i,t}^- S_{j,t+p_i}^- + S_{i,t+p_j}^- S_{j,t}^- \cdot S_{i,t}^+ S_{j,t+p_i}^+$.

**Resource count:**
- **Color-time partitioned time-swap mixer:** $h\binom{n}{2}$ 4-qubit gates in depth $h\kappa$.
- **Phase separator:** at most $nh$ single qubit gates, depth 1.
- **Initial state:** $n$ single-qubit $X$ gates, depth 1.

### A.5.4 SMS, with release dates (Sec. 4.4)

**Problem:** $(1|d_j, r_j|f)$ Given a set of jobs with processing times $\mathbf{p}$, deadlines $\mathbf{d}$ release times $\mathbf{r} \in \mathbb{Z}_+$ and weights $\mathbf{w}$, find a schedule that minimizes some (given) function $f$ of the schedule, e.g. weighted total tardiness such that each job starts no earlier than its release time.

**Approximability:** For all deadlines being zero, the minimal weighted total tardiness (completion times in this special case) is 1.685-approximable [28].

**Configuration space:** $\{\mathbf{s}\} = \times_{j=1}^{n} [W_j \cup \{b_j\}]$, where $W_j = [r_j, h - p_j]$ is the window of times in which job $j$ can start and $b_j$ is the job's "buffer" slot.

**Domain:** Schedules in the configuration space such that no two jobs overlap.

**Objective:** A given function $f(\mathbf{s})$, e.g. weighted total tardiness $f(\mathbf{s}) = \sum_{j=1}^{n} w_j T_j$, where $T_j = \max\{0, d_i - s_i - p_i\}$ is the tardiness of job $j$.

**Mixing rule:** Swap a job between time $t \in h$ and its buffer slot if no other job is running at time $t$.

**Partial mixer:** Controlled null-swap mixer; see Sec. 4.4.

**Initial state:** Greedy earliest-release-date schedule.

**Encoding:** One-hot encoding.

**Compilation:**
- **Partial mixer:** $H_{\mathrm{NS},i,t}^{(\mathrm{enc})} = \left( \prod_{j \neq i} \prod_{t' \in \mathrm{nbhd}_{i,t}(j)} \frac{1}{2}(I + Z_{j,t'}) \right) \left( X_{i,t} X_{i,b_i} + Y_{i,t} Y_{i,b_i} \right)$ .
- **Phase separator (for min weighted total tardiness):** See Eq. (77).

**Resource count:**
- **Controlled null swap mixer:** SMS-instance dependent, see discussion in Sec. 4.4.
- **Phase separator (for minimizing weighted total tardiness):** At most $\sum_j (h - d_j + 1)$ single qubit $Z$ gates. Depth 1.
- **Initial state:** $n$ single-qubit $X$ gates. Depth 1.

## B    Glossary of mapping terms

### B.1    Mixers

We list partial mixers based on mixing mechanism in Sec. B.1.1, and different ways to partition, used for partitioned mixers, in Sec. B.1.2

### B.1.1    **Partial Mixing Hamiltonians:**

For strings and subsets thereof, we focus on mixing operators that are composed of independent mixing operators on each of the variables. Specifically, we consider the following single-qudit mixing operators in the absence of constraints:

- $r$-nearby-values mixer: $H_{r\text{-NV}} = \sum_{a=1}^{r} \left( \breve{X}^a + (\breve{X}^\dagger)^a \right)$. The special cases of $r = 1$ and $r = d - 1$ are called the "ring mixer" and "fully-connected mixer", respectively.

- simple binary mixer: when $d = 2^l$ is a power of two: $H_{\mathrm{binary}}^{(\mathrm{enc})} = \sum_{i=1}^{l} X_i$,

- "Null swap mixer": for cases when one of the $d$ values corresponds to a "null" value (e.g. black or uncolored in graph coloring), $H_{\mathrm{NS}} = \sum_{a=1}^{d-1} (|0\rangle \langle a| + |a\rangle \langle 0|)$.

For problem in the ordering/schedules family, we consider swapping-based mixers.

- value-selective permutation swap mixer: swaps the $i$th and $j$th elements in the ordering if those elements are $u$ and $v$, Eq. (46) in Sec. 4.1.

- value-independent permutation swap mixer: swaps the $i$th and $j$th elements of the ordering regardless of which items those are, see Eq. (48). in Sec. 4.1.

When there are constraints, we considered modifications of the above mixers that are controlled on not violating the constraints. A few examples:

- Controlled Null swap mixer:

  - Sec. 3.3 for MaxColorableInducedSubgraph, Eqs. (27).
  - Sec. 4.4 for SMS with release dates, Eqs. (73).

- Controlled-SWAP mixer: Sec. 3.4 for MinGraphColoring, Eq. (34).

### B.1.2  **Partitions:**

The two elementary partitions we use for generating the family of partitioned mixing unitaries are:

- parity-mixer: For Hamiltonian terms of type $\sum_u A_u B_{u+1}$, where $u \in [n]$ and $A_u$ and $B_{u+1}$ are operators acting on qubit $u$ and $u+1$, respectively. Partition the index set $\{u\}$ into even and odd subsets. See Sec. 3.1 for details.

- color-mixer: For index pairs $(u,v) \in \binom{[n]}{2}$, let $\mathcal{P}_{\mathrm{col}} = (P_1, \ldots, P_\kappa)$ be an ordered partition of the indices $\binom{[n]}{2}$ into $\kappa$ parts such that each part contains only mutually disjoint pairs of indices from $[n]$. This is equivalent to considering a $\kappa$-edge-coloring of the complete graph $K_n$, and assigning an ordering to the colors, so we call $\mathcal{P}_{\mathrm{col}}$ the "color partition". For even $n$, $\kappa = n - 1$ suffices, and for odd $n$, $\kappa = n$. See Sec. 4.1 for its use.

For mixing Hamiltonians of different coupling connectivity, these partitions can be combined and tailored as desired. For example, the color-parity permutation swap mixer for TSP in Sec. 4.1 and the time-color partition for SMS $(1|d_j| \sum w_j T_j)$ in Sec. 4.3.

### B.2  Encodings

We consider two types of encodings of strings into qubit space. The one-hot encoding enables more concise circuits at the expense of requiring more qubits, whereas the binary encoding makes the opposite trade-off.

- one-hot encoding: The state $|a\rangle \in \mathbb{C}^d$ is encoded as $|0\rangle^{\otimes a} \otimes |1\rangle \otimes |1\rangle^{\otimes d-1-a} \in \mathbb{C}^{2^d}$. See Sec. 3.1.1

- binary encoding: The state $|a\rangle$ is encoded as $|\mathbf{b}\rangle \in \mathbb{C}^{2^{l^*}}$, where $l^* = \lceil \ln(d-1) \rceil$ and $\mathbf{b} \in [2]^{l^*}$ is the binary representation of $a$. See Sec. 3.1.1.

  A generalization of the binary encoding is the radix encoding, which represents $a$ in $r$-base with an arbitrary integer $r$. While the binary bears the advantage in

implentability, it is plausible that for some problems the general radix encoding could be a natural choice. Radix encoding is a generalization of the binary encoding with the latter bears the advantage of easy implentability.

For problem in the ordering/schedule family, the encoding is composed of two-steps: first encoding into strings, and then apply the above encodings into qubit space. The following encodings are considered for step-one:

- direct encoding: an ordering $\boldsymbol{\iota} = (\iota_1, \ldots, \iota_n)$ is encoded directly as a string $[n]^n$ of integers.

  It is demonstrated for TSP and SMS $(1|d_j| \sum_j w_j T_j^2)$ in Sec. 4.1, and 4.2, respectively.

- absolute encoding: To endoce the ordering $\boldsymbol{\iota} = (\iota_1, \ldots, \iota_n)$, we assign each item $i$ a value $s_i \in [0, h]$, where the "horizon" $h$ is a parameter of the encodings, such that for all $i < j$, $s_{\iota_i} < s_{\iota_j}$.

  It is demonstrated for SMS $(1|d_j| \sum_j w_j T_j)$ in Sec. 4.3.

The name of the overall encoding features both steps. For example
- direct one-hot encoding, see Sec. 4.1.
- absolute one-hot encoding, see Sec. 4.3.

## C  Elementary operators

Here, we elaborate on some of basic quantum operators and their properties that are used in the text. Sec. C.1 explains the the relations between the SWAP gate and the XY-model Hamiltonian $XX + YY$, which are used as building blocks for encoded mixers in many mappings. Sec. C.2 contains a brief review of generalized Pauli operators for qudits of arbitrary dimension.

### C.1  SWAP and XY opertors

We examine the relations between the quantum SWAP operator $\text{SWAP}_{i,j}$, which swaps the state of qubit $i$ and $j$, and the XY operator, $X_i X_j + Y_i Y_j$. In many of our mapping constructions, these operators can be used interchangeably.

The XY operator can be expressed as

$$X_i X_j + Y_i Y_j = \frac{1}{2} \left( |1_i 0_j\rangle\langle 0_i 1_j| + |0_i 1_j\rangle\langle 1_i 0_j| \right) . \tag{79}$$

We observe the following connections between the two Hamiltonians:

- In the subspace spanned by $\{|01\rangle, |10\rangle\}$, $X_i X_j + Y_i Y_j$ and $\text{SWAP}_{i,j}$ behave identically.

- In the subspace $\{|11\rangle, |00\rangle\}$, $X_i X_j + Y_i Y_j$ acts as null while $\text{SWAP}_{i,j}$ acts as identity.

- The operators $X_i X_j + Y_i Y_j$ and $\text{SWAP}_{i,j}$ are both unitary and Hermitian.

- Applied to a many-qubit system, both $H_{\text{SWAP}} = \sum_{i,j} \text{SWAP}_{i,j}$ and $H_{\text{XY}} = \sum_{i,j} X_i X_j + Y_i Y_j$ preserve Hamming weights, hence so do the corresponding unitaries $\exp[-i\beta H_{\text{XY}}]$ and $\exp[-i\beta H_{\text{SWAP}}]$. Although the two do not behave identically on the full Hilbert space, they can both serve as mixers in situations in which Hamming weight is the relevant constraint.

- To enforce simultaneous swaps of multiple qubit pairs, in $\prod_{\{i,j\}} \text{SWAP}_{i,j}$, the $\text{SWAP}_{i,j}$ cannot be directly replaced by $X_i X_j + Y_i Y_j$; see the TSP problem in Sec. 4.1 as an example.

## C.2 Generalized Pauli gates for qudits

We consider $d$-dimensional qudits. Let $\omega = e^{2\pi i/d}$. We have the following gates

$$\breve{Z} = \sum_{a=0}^{d-1} \omega_d^a |a\rangle \langle a|, \qquad\qquad \breve{X} = \sum_{a=0}^{d-1} |a+1\rangle \langle a|, \qquad (80)$$

where all arithmetic is modulo $d$. As in the main text, we use operator notations "$\breve{Z}$" and "$\breve{X}$" for qudits of arbitrary dimension, and reserve "$X$" and "$Z$" specifically for the qubit case. For $d = 2$, they are the same. Note that for $d > 2$, the generalized Pauli operators, while unitary, are not Hermitian. In many cases, we use the sum of a generalized Pauli operator and its conjugate to generalize its qubit analog, e.g. $\breve{X} + \breve{X}^\dagger$ to generalize $X$.

Because $\breve{X}^r = \sum_{a=0}^{d-1} |a+r\rangle\langle a|$, it and its Hermitian conjugate $\breve{X}^r + (\breve{X}^\dagger)^r$ together generate transitions between $|a\rangle$ and $|a+r\rangle$ for any $a$. Sec. 3.1.1 uses this operator in the coloring of a single vertex.

## References

[1] Soifer Alexander. The mathematical coloring book, 2008.

[2] Gunnar Andersson and Lars Engebretsen. Better approximation algorithms for Set splitting and Not-All-Equal SAT . *Information Processing Letters*, 65(6):305–311, 1998.

[3] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer Science & Business Media, 2012.

[4] Adi Avidor and Uri Zwick. Approximating min 2-sat and min 3-sat. *Theory of Computing Systems*, 38(3):329–345, 2005.

[5] Boaz Barak, Ankur Moitra, Ryan O'Donnell, Prasad Raghavendra, Oded Regev, David Steurer, Luca Trevisan, Aravindan Vijayaraghavan, David Witmer, and John Wright. Beating the random assignment on constraint satisfaction problems of bounded degree. *arXiv:1505.03424*, 2015.

[6] Stephen D Bartlett, Hubert de Guise, and Barry C Sanders. Quantum encodings in spin systems and harmonic oscillators. *Physical Review A*, 65(5):052316, 2002.

[7] Cristina Bazgan, Bruno Escoffier, and Vangelis Th Paschos. Completeness in standard and differential approximation classes: Poly-(D) APX-and (D) PTAS-completeness. *Theoretical Computer Science*, 339(2-3):272–292, 2005.

[8] Dimitris Bertsimas, Chungpiaw Teo, and Rakesh Vohra. On dependent randomized rounding algorithms. *Operations Research Letters*, 24(3):105–114, 1999.

[9] Rupak Biswas, Zhang Jiang, Kostya Kechezhi, Sergey Knysh, Salvatore Mandrà, Bryan O'Gorman, Alejandro Perdomo-Ortiz, Andre Petukhov, John Realpe-Gómez, Eleanor Rieffel, et al. A NASA perspective on quantum computing: Opportunities and challenges. *Parallel Computing*, 64:81–98, 2017.

[10] Sergio Boixo, Vadim N. Smelyanskiy, Alireza Shabani, Sergei V. Isakov, Mark Dykman, Vasil S. Denchev, Mohammad H. Amin, Anatoly Yu Smirnov, Masoud Mohseni, and Hartmut Neven. Computational multiqubit tunnelling in programmable quantum annealers. *Nat Commun*, 7, 01 2016.

[11] Michael J Bremner, Richard Jozsa, and Dan J Shepherd. Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy. In

*Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, page rspa20100301. The Royal Society, 2010.

[12] Michael J Bremner, Ashley Montanaro, and Dan J Shepherd. Achieving quantum supremacy with sparse and noisy commuting quantum computations. *arXiv preprint arXiv:1610.01808*, 2016.

[13] Michael J Bremner, Ashley Montanaro, and Dan J Shepherd. Average-case complexity versus approximate simulation of commuting quantum computations. *Physical review letters*, 117(8):080501, 2016.

[14] TC Edwin Cheng, CT Ng, JJ Yuan, and ZH Liu. Single machine scheduling to minimize total weighted tardiness. *European Journal of Operational Research*, 165(2): 423–443, 2005.

[15] Jerry Moy Chow. *Quantum information processing with superconducting qubits*. Yale University, 2010.

[16] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[17] S Debnath, NM Linke, C Figgatt, KA Landsman, K Wright, and C Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536 (7614):63–66, 2016.

[18] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, pages 439–485, 2005.

[19] Irit Dinur and Shmuel Safra. The importance of being biased. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 33–42. ACM, 2002.

[20] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 624–633. ACM, 2014.

[21] Edward Farhi and Aram W. Harrow. Quantum Supremacy through the Quantum Approximate Optimization Algorithm. *arXiv:1602.07674 [quant-ph]*, February 2016.

[22] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A Quantum Approximate Optimization Algorithm. *arXiv:1411.4028 [quant-ph]*, November 2014.

[23] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A Quantum Approximate Optimization Algorithm Applied to a Bounded Occurrence Constraint Problem. *arXiv:1412.6062 [quant-ph]*, December 2014.

[24] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Hartmut Neven. Quantum Algorithms for Fixed Qubit Architectures. *arXiv preprint arXiv:1703.06199*, 2017.

[25] Uriel Feige, Marek Karpinski, and Michael Langberg. Improved approximation of max-cut on graphs of bounded degree. *Journal of Algorithms*, 43(2):201–219, 2002.

[26] Alan Frieze and Mark Jerrum. Improved approximation algorithms for maxk-cut and max bisection. *Algorithmica*, 18(1):67–81, 1997.

[27] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[28] Michel X Goemans, Maurice Queyranne, Andreas S Schulz, Martin Skutella, and Yaoguang Wang. Single machine scheduling with release dates. *SIAM Journal on Discrete Mathematics*, 15(2):165–192, 2002.

[29] Daniel Gottesman, Alexei Kitaev, and John Preskill. Encoding a qubit in an oscillator. *Physical Review A*, 64(1):012310, 2001.

[30] Gian Giacomo Guerreschi and Mikhail Smelyanskiy. Practical optimization for hybrid quantum-classical algorithms. *arXiv preprint arXiv:1701.01450*, 2017.

[31] Venkatesan Guruswami. Inapproximability results for set splitting and satisfiability problems with no mixed clauses. *Algorithmica*, 38(3):451–469, 2004.

[32] Magnús M Halldórsson. A still better performance guarantee for approximate graph coloring. *Information Processing Letters*, 45(1):19–23, 1993.

[33] Magnús M Halldórsson. Approximating discrete collections via local improvements. In *SODA*, volume 95, pages 160–169, 1995.

[34] Magnús M Halldórsson, Jan Kratochvıl, and Jan Arne Telle. Independent sets with domination constraints. *Discrete Applied Mathematics*, 99(1):39–54, 2000.

[35] Johan Håstad. Some Optimal Inapproximability Results. *J. ACM*, 48(4):798–859, July 2001. ISSN 0004-5411. DOI: 10.1145/502090.502098.

[36] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM (JACM)*, 48(4):798–859, 2001.

[37] Elad Hazan, Shmuel Safra, and Oded Schwartz. On the complexity of approximating k-set packing. *computational complexity*, 15(1):20–39, 2006.

[38] Itay Hen and Marcelo S Sarandy. Driver Hamiltonians for constrained optimization in quantum annealing. *Physical Review A*, 93(6):062312, 2016.

[39] Itay Hen and Federico M Spedalieri. Quantum Annealing for Constrained Optimization. *Physical Review Applied*, 5(3):034007, 2016.

[40] IBM. IBM Q and Quantum Computing. https://www.research.ibm.com/ibm-q/, 2017. Accessed: 2017-09-01.

[41] Zhang Jiang, Eleanor G Rieffel, and Zhihui Wang. Near-optimal quantum circuit for Grover's unstructured search using a transverse field. *Physical Review A*, 95(6): 062317, 2017.

[42] David S Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 38–49. ACM, 1973.

[43] George Karakostas. A better approximation ratio for the vertex cover problem. *ACM Transactions on Algorithms (TALG)*, 5(4):41, 2009.

[44] Howard Karloff and Uri Zwick. A 7/8-approximation algorithm for MAX 3SAT? In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 406–415. IEEE, 1997.

[45] Sanjeev Khanna, Rajeev Motwani, Madhu Sudan, and Umesh Vazirani. On syntactic versus computational views of approximability. *SIAM Journal on Computing*, 28(1): 164–191, 1998.

[46] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007.

[47] Rajeev Kohli, Ramesh Krishnamurti, and Prakash Mirchandani. The minimum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7(2):275–283, 1994.

[48] Robert Krauthgamer and Uriel Feige. A polylogarithmic approximation of the minimum bisection. *SIAM review*, 48(1):99–130, 2006.

[49] Jan Karel Lenstra, AHG Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. *Annals of discrete mathematics*, 1:343–362, 1977.

[50] Michael Lewin, Dror Livnat, and Uri Zwick. Improved rounding techniques for the MAX 2-SAT and MAX DI-CUT problems. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 67–82. Springer, 2002.

[51] László Lovász. Coverings and colorings of hypergraphs. In *Proc. 4th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, pages 3–12. Utilitas Mathematica Publishing, Winnipeg, 1973.

[52] Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2(5): 1–15, 2014. DOI: 10.3389/fphy.2014.00005.

[53] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM (JACM)*, 41(5):960–981, 1994.

[54] Madhav V Marathe and SS Ravi. On approximation algorithms for the minimum satisfiability problem. *Information Processing Letters*, 58(1):23–29, 1996.

[55] Masoud Mohseni, Peter Read, Hartmut Neven, Sergio Boixo, Vasil Denchev, Ryan Babbush, Austin Fowler, Vadim Smelyanskiy, and John Martinis. Commercialize Quantum Technologies in Five Years. *Nature*, 543:171–174, 2017.

[56] Masoud Mohseni, Peter Read, Hartmut Neven, Sergio Boixo, Vasil Denchev, Ryan Babbush, Austin Fowler, Vadim Smelyanskiy, John Martinis, et al. Commercialize early quantum technologies, 2017.

[57] Takao Nishizeki and Kenichi Kashiwagi. On the 1.1 edge-coloring of multigraphs. *SIAM Journal on Discrete Mathematics*, 3(3):391–410, 1990.

[58] Pekka Orponen and Heikki Mannila. On approximation preserving reductions: Complete problems and robust measures (revised version). *Department of Computer Science, University of Helsinki*, 1990.

[59] Alessandro Panconesi and Desh Ranjan. Quantifiers and approximation. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 446–456. ACM, 1990.

[60] Christos Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[61] Christos H Papadimitriou. Computational Complexity, 1994.

[62] Christos H Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[63] Erez Petrank. The hardness of approximation: Gap location. *Computational Complexity*, 4(2):133–157, 1994.

[64] E. G. Rieffel and W. Polak. *Quantum Computing: A Gentle Introduction*. MIT Press, Cambridge, MA, 2011.

[65] Eleanor G Rieffel, Davide Venturelli, Bryan O'Gorman, Minh B Do, Elicia M Prystay, and Vadim N Smelyanskiy. A case study in programming a quantum annealer for hard operational planning problems. *Quantum Information Processing*, 14(1):1–36, 2015.

[66] Mark Saffman. Quantum computing with atomic qubits and rydberg interactions: progress and challenges. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 49(20):202001, 2016.

[67] Jeffrey Schaller and Jorge MS Valente. Minimizing the weighted sum of squared tardiness on a single machine. *Computers & Operations Research*, 39(5):919–928, 2012.

[68] E. A. Sete, W. J. Zeng, and C. T. Rigetti. A functional architecture for scalable quantum computing. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–6, Oct 2016. DOI: 10.1109/ICRC.2016.7738703.

[69] Luca Trevisan. Inapproximability of combinatorial optimization problems. *arXiv preprint cs/0409043*, 2004.

[70] Nobuhisa Ueda and Tadaaki Nagao. NP-completeness results for NONOGRAM via parsimonious reductions. *preprint*, 1996.

[71] Davide Venturelli, Minh Do, Eleanor Rieffel, and Frank Jeremy. Compiling Quantum Circuits to Realistic Hardware Architectures using Temporal Planners. *arXiv:1705.08927*, 2017.

[72] Frank Verstraete, J Ignacio Cirac, and José I Latorre. Quantum circuits for strongly correlated quantum systems. *Physical Review A*, 79(3):032316, 2009.

[73] Zhihui Wang, Stuart Hadfield, Zhang Jiang, and Eleanor G. Rieffel. The Quantum Approximation Optimization Algorithm for MaxCut: A Fermionic View. *arXiv:1706.02998*, 2017.

[74] Dave Wecker, Matthew B Hastings, and Matthias Troyer. Training a quantum optimizer. *Physical Review A*, 94(2):022309, 2016.

[75] Zhi-Cheng Yang, Armin Rahmani, Alireza Shabani, Hartmut Neven, and Claudio Chamon. Optimizing Variational Quantum Algorithms using Pontryagin's Minimum Principle. *arXiv preprint arXiv:1607.06473*, 2016.

[76] Takayuki Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 86(5):1052–1060, 2003.

[77] Ehsan Zahedinejad and Arman Zaribafiyan. Combinatorial optimization on gate model quantum computers: A survey. *arXiv preprint arXiv:1708.05294*, 2017.

[78] Jiawei Zhang, Yinyu Ye, and Qiaoming Han. Improved approximations for max set splitting and max nae sat. *Discrete Applied Mathematics*, 142(1):133–149, 2004.

[79] David Zuckerman. On unapproximable versions of NP-complete problems. *SIAM Journal on Computing*, 25(6):1293–1304, 1996.

[80] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 681–690. ACM, 2006.

[81] Uri Zwick. Approximation algorithms for constraint satisfaction problems involving at most three variables per constraint. In *SODA*, volume 98, pages 201–210, 1998.