Combining Hard and Soft Constraints in Quantum Constraint-Satisfaction Systems

Ellis Wilson
North Carolina State University
Raleigh, North Carolina 27695-8206
Email: ejwilso2@ncsu.edu

Frank Mueller
North Carolina State University
Raleigh, North Carolina 27695-8206
Email: mueller@cs.ncsu.edu

Scott Pakin
Los Alamos National Laboratory
Los Alamos, New Mexico 87545
Email: pakin@lanl.gov

***Abstract*—This work presents a generalization of NchooseK, a constraint satisfaction system designed to target both quantum circuit devices and quantum annealing devices. Previously, NchooseK supported only hard constraints, which made it suitable for expressing problems in NP (e.g., 3-SAT) but not NP-hard problems (e.g., minimum vertex cover). In this paper we show how support for soft constraints can be added to the model and implementation, broadening the classes of problems that can be expressed elegantly in NchooseK without sacrificing portability across different quantum devices.**

Through a set of examples, we argue that this enhanced version of NchooseK enables problems to be expressed in a more concise, less error-prone manner than if these problems were encoded manually for quantum execution. We include an empirical evaluation of performance, scalability, and fidelity on both a large IBM Q system and a large D-Wave system.

***Index Terms*—circuit-model quantum computing, quantum annealing, programming models**

I. Introduction

Much like GPUs, which have become omnipresent in high-performance computing (HPC) systems, quantum processing units (QPUs) are intended to accelerate computational kernels. The difference is that QPUs offer the potential of solving computationally hard problems in shorter time than would be possible via *any* form of classical computing—either by a constant though polynomial factor (termed "quantum advantage") or in a few cases even exponentially, making classically intractable problems tractable (termed "quantum supremacy"). In today's age of noisy, intermediate-scale quantum (NISQ) computation [1], practical experiments are limited by the number of available qubits and their high susceptibility to noise. Consequently, quantum supremacy in particular has been demonstrated to date on actual QPUs only for problems or input sizes that lack practical applicability [2], [3], [4]. Nevertheless, the hope that future, fault-tolerant quantum computers will usher in a new era of HPC makes quantum computing an area with significant research potential and relevance to the HPC community.

Quantum programming requires a way of thinking that is very unlike that of classical programming and as such can have a high barrier of entry even for those already comfortable with coding in a variety of classical computer languages. Furthermore, there is substantial architectural variety among different quantum computers—analogous to CPUs vs. GPUs vs. TPUs [5] vs. IPUs [6] vs. RDUs [7] and the like in the classical world—which frustrates the creation of a portable programming model.

Currently, the two dominant architectural models for quantum computers are the *circuit model* and the *annealing model*. Most hardware vendors, including IBM, IonQ, Rigetti, Honeywell, ColdQuanta, PsiQuantum, Quantum Brilliance, and many more, are basing their products on the circuit model [8], [9], [10], [11], [12]. At its core, a circuit-model program is an enormous ($2^n \times 2^n$) unitary matrix, expressed as the product of tensor products of small (usually 2×2 and 4×4) unitary matrices.

D-Wave [13] is the lone vendor championing the annealing model, although Fujitsu's Digital Annealer [14] represents a classical analogue (same computational model but a classical rather than a quantum implementation). Although both the circuit model and the annealing model are ultimately governed by the Schrödinger equation, an annealing-model program is essentially a quadratic pseudo-Boolean function. The hardware searches (heuristically) for the inputs that minimize this function [15], [16].

Being tied specifically to a particular type of optimization problem, the annealing model is more restrictive than the general-purpose circuit model. However, the annealing model offers an important engineering advantage: scalability. D-Wave has manufactured annealing devices with about two orders of magnitude more qubits than what is available today for the circuit model. At the time of this writing, D-Wave's largest machine provides nearly 5,760 qubits, while IBM's largest machine provides only 127.

To date, there have been virtually no attempts to develop a high-level programming model that bridges these two quantum computational models. Because of the popularity of the circuit model, most programming systems target that. Some recent examples of circuit-model programming languages are Twist [17], Silq [18], Q# [19], ProjectQ [20], 2WIRE [21], Scaffold [22], and Quipper [23]. D-Wave's Ocean API [24] facilitates the expression of annealing-model programs. All of these work at a fairly low level of abstraction. The circuit-model systems provide mechanisms for juxtaposing small unitary matrices in a large matrix product, and the annealing-model system provides mechanisms for specifying coefficients for a quadratic pseudo-Boolean function.

SC22, November 13-18, 2022, Dallas, Texas, USA
978-1-6654-5444-5/22/$31.00 ©2022 IEEE

A rare example of cross-paradigm quantum programming is NchooseK [25], [26]. NchooseK is a domain-specific language focusing on the domain of constraint satisfaction problems. It seeks to work at a sufficiently high level of abstraction as to both facilitate programming, even for quantum novices, and enable execution on both circuit-model and annealing-model devices. The fundamentals of a simplistic NchooseK abstraction was first used for a Grover search by Khetawat et al. [25] and developed further for simple constraint-satisfaction problems in a workshop paper by Wilson et al. [26]. Section II elaborates further, but a small example of an NchooseK program is $nck(\{a,b\},\{0,1\}) \wedge nck(\{b,c\},\{1\})$, which is interpreted as "Neither or exactly one of a and b must be TRUE, and, simultaneously, exactly one of b and c must be TRUE."

This paper presents a more generalized variant of NchooseK for expressing complex constraint-satisfaction problems. Specifically, the paper makes the following contributions:

- It introduces soft constraints—constraints, which, if broken, will incur a penalty but will not invalidate the problem. Soft constraints are crucial for expressing minimization or maximization problems in NchooseK.
- It evaluates a larger set of NchooseK problems, including both hard and soft constraints, than had previous been studied.
- It compares both the complexity of NchooseK and the quality of the *quadratic unconstrained binary optimization* (QUBO) expressions used as an intermediate representation of NchooseK, by comparing them to manually created QUBOs for the same problems.
- It evaluates quantum computations of much larger scale in today's terms than previous work—of up to 65 qubits on the IBM gate-based machines, utilizing every qubit on the ibmq_brooklyn [27], and 1163 qubits on the D-Wave quantum annealers, even within a range where correct answers were potentially no longer found.

II. Background

NchooseK is a programming paradigm based on expressing constraint-satisfaction problems over a set of boolean variables. Each constraint in a problem specification takes the form, "Given a variable collection of size N, a specified number of them, K, must be TRUE." Before elaborating we state some relevant definitions:

Definition 1 (Variable collection)**.** A *variable collection* comprises a number of Boolean variables in which variables can be repeated, but order does not matter. Its cardinality is the number of elements (which can exceed the number of unique variables due to repetitions).

Definition 2 (Selection set)**.** A *selection set* comprises a set of disjoint whole numbers, none of which can be greater than the cardinality of a corresponding variable collection.

Definition 3 (Hard constraint)**.** An NchooseK *hard constraint*, written as $nck(N,K)$, consists of a variable collection N and a selection set K. It is satisfied if the cardinality of the variable collection whose variables are TRUE equals one of the numbers in the selection set:

$$nck(N,K) \equiv \left(\sum_{n\in N} n\right) \in K,$$

where $n \in \{0,1\}$ and we associate FALSE with 0 and TRUE with 1.

Definition 4 (NchooseK program)**.** An *NchooseK program* is a conjunction of NchooseK hard constraints written as $nck(N_1,K_1) \wedge nck(N_2,K_2) \wedge \cdots \wedge nck(N_n,K_n)$. The result of executing a program is either an assignment of Boolean values to all variables over the variable collections such that all hard constraints are honored or an indication that no such assignment exists.

To create useful NchooseK constraints, a programmer must focus on the relationships among variables. For example, consider a collection containing the variables a and b. The problem formulation in which a and b must both be TRUE is given by the constraint $nck(\{a,b\},\{2\})$. This indicates that exactly two of a and b must be TRUE, and therefore none can be FALSE. If they need to have the same value but it does not matter which, this would be expressed as $nck(\{a,b\},\{0,2\})$. By including two numbers in the selection set, K, this constraint will be satisfied if two variables are TRUE or if zero variables are TRUE but not if exactly one is TRUE. If, on the other hand, the two variables need to have different values, the constraint would be $nck(\{a,b\},\{1\})$, indicating that exactly one must be TRUE, and, therefore, the other must be FALSE. If at least one of a and b need to be TRUE, the constraint would be $nck(\{a,b\},\{1,2\})$. Omitting 0 from the selection set ensures that they cannot both be FALSE.

As a more complicated example, consider satisfiability problems, discussed more in depth in Section VI. A satisfiability problem accepts an expression in conjunctive normal form (conjunctions of unions of possibly negated variables) and reports whether there exists a variable assignment that makes the expression TRUE. "$(v_1 \vee v_2 \vee \neg v_3) \wedge (\neg v_2 \vee \neg v_3 \vee v_4) \stackrel{?}{=}$ TRUE" is an example of a 3-SAT problem, which is a satisfiability problem in which each clause contains at most three variables. For a single 3-SAT clause $(x \vee y \vee z)$ to be TRUE, at least one of the three variables must be TRUE. This is expressed in NchooseK with the constraint

$$nck(\{x,y,z\},\{1,2,3\}) .$$

This constraint is illustrated graphically in Figure 1.

III. Related Work

A number of quantum circuit languages are being developed, either as standalone languages or as embedded domain-specific languages. These include Q# [19], Twist [17], Silq [18], ProjectQ [20], 2WIRE [21], Scaffold [22], and Quipper [23]. D-Wave's Ocean API [24] likewise functions as a language for their annealing devices and simulators. While not a language per se, Xanadu's PennyLane is a quantum machine-learning

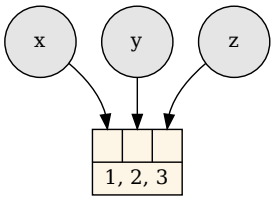

Fig. 1: A visual representation of a 3-SAT clause with the variables x, y, and z. The nodes represent the Boolean variables, and the box indicates the constraint.

software package designed to work across a number of circuit-model systems [28]. In contrast to those efforts, which target a single computational model apiece, NchooseK programs run unmodified on both annealing-model and circuit-model machines.

XACC [29] is a software infrastructure that can interface to multiple hardware platforms, including both circuit-model and annealing-model systems. It enables classical programs to embed blocks of quantum code, e.g., written in Quil [30], and designate a quantum computer on which to run it. The primary difference with NchooseK is that NchooseK raises the level of abstraction above that of the underlying form of quantum computation, enabling true portability across computational models. XACC, in contrast, enables a program to integrate circuit-model-specific code that runs only on circuit-model quantum computers and annealing-model-specific code that runs only on quantum annealers. Despite defining its own intermediate representation, XACC is not designed to run any given piece of code on both circuit-model quantum computers and quantum annealers. Another difference between the two systems is that one can program in NchooseK without any knowledge of quantum computing while XACC programmers must be familiar with at least one quantum computational model.

The closest related work to ours is Wilson et al. [26], which introduces NchooseK for hard constraints. However, their work lacks soft constraints, which are essential for generalizing NchooseK's applicability to maximization and minimization problems. Our work not only fills this gap but also presents a problem complexity analysis, considers symmetrical constraints in doing so, and more thoroughly evaluates success characteristics through an empirical study involving both quantum circuit and annealing devices.

IV. Soft Constraints

In this work we propose a generalized NchooseK model that additionally supports *soft* constraints: constraints whose satisfaction is desired but not required. To motivate the need for soft constraints we consider an example of a problem that cannot be expressed in the existing NchooseK paradigm. We attempt to solve this problem first using only hard constraints and then, after showing how that fails, including soft constraints to make the problem expressible.

A. Problem requirements and initial formulation

Minimum Vertex Cover is a well-known graph problem: Given an undirected graph $G = (V,E)$, a *vertex cover* is a subset of vertices $W \subseteq V$ such that each edge in E is connected to at least one member of W. The Minimum Vertex Cover is the smallest W in cardinality that meets this requirement.

The first step in solving any NchooseK problem is deciding what the variables should represent. Because the solution to a Minimum Vertex Cover problem is formulated in terms of vertices, we associate one variable per vertex such that the variable is TRUE if and only if the corresponding vertex is in W.

B. Setting up the vertex cover

As a running example, consider the graph in Figure 2, which has five vertices and five edges.

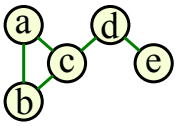

Fig. 2: A graph of 5 vertices for reference

Consider first a smallest possible subgraph, e.g., the graph $G' = (\{a,b\},\{(a,b)\})$. For G', we can easily determine a minimum vertex cover immediately by expressing the problem with the constraint $nck(\{a,b\},\{1\})$. This ensures that exactly one of the two variables will be TRUE and gives W a cardinality of 1.

An inductive step is non-trivial. If we add to G' vertex c and edges (a,c) and (b,c), the resulting constraints, $nck(\{a,c\},\{1\})$ and $nck(\{b,c\},\{1\})$, cannot both be satisfied. For instance, if $a \in W$ then a is TRUE. In this case, b and c must both be FALSE by the constraints $nck(\{a,b\},\{1\})$ and $nck(\{a,c\},\{1\})$, which ensure that exactly one of the variables in the collections is TRUE, and a must have the same value in all NchooseK constraints within the same program. This leaves the constraint $nck(\{b,c\},\{1\})$ unsatisfiable.

Instead, we need to refine our original constraint to allow both variables to be TRUE if necessary. Using $nck(\{a,b\},\{1,2\})$, as illustrated in Figure 3, not only expresses a constraint that finds a vertex cover for our minimal subgraph but can be extended over the entire graph to ensure that a solution can be found.

The refined NchooseK program for five vertices is

$$nck(\{a,b\},\{1,2\}) \wedge nck(\{a,c\},\{1,2\}) \wedge$$
$$nck(\{b,c\},\{1,2\}) \wedge nck(\{c,d\},\{1,2\}) \wedge nck(\{d,e\},\{1,2\}),$$

and this is illustrated in Figure 4. Unfortunately, this program is incorrect in that it will be satisfied by *any* vertex cover of the graph in Figure 2, not necessarily a minimum vertex cover. The problem is that NchooseK requires all constraints to be met, but this is not generally possible in a minimization or maximization problem.

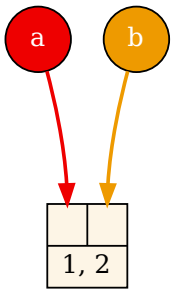

Fig. 3: A single edge in a vertex cover. Each node corresponds to a vertex in the original graph and a variable in the NchooseK program. The box represents an NchooseK constraint.

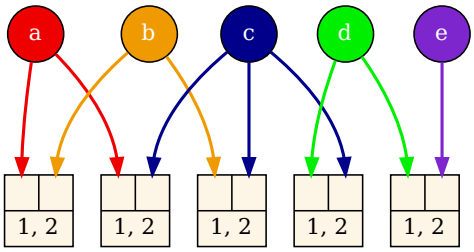

Fig. 4: A full vertex cover representation in NchooseK. This will be satisfied by every valid vertex cover.

C. Minimization via soft constraints

To find specifically a *minimum* vertex cover we propose generalizing NchooseK to support soft constraints in addition to its existing hard constraints:

Definition 5 (Soft constraint)**.** An NchooseK soft constraint, written as $nck(N,K,soft)$, acts as a desired but not required constraint.

Definition 6 (Generalized NchooseK program)**.** A *generalized NchooseK program* is a conjunction of NchooseK hard and soft constraints written as $nck(N_1,K_1)\wedge nck(N_2,K_2)\wedge nck(N_i,K_i)\wedge nck(N_{i+1},K_{i+1},soft)\wedge nck(N_{i+2},K_{i+2},soft)\wedge nck(N_m,K_m,soft)$. The result of executing a program is either an assignment of Boolean values to all variables over the variable collections such that all hard constraint are honored and the number of satisfied soft constraints is maximized; or an indication that no such assignment exists.

In short, the semantics is that an NchooseK program execution will satisfy all hard constraints (or fail if this is not possible) and as many soft constraints as it can.

For a minimization problem, one wants as few variables as possible to be TRUE. To this end, one can associate a soft constraint with each variable: $nck(\{v\},\{0\},soft)$, to indicate a preference but not a demand that v be 0. Consequently, adding the following constraints to our Minimum Vertex Cover program requests that the solution represent a minimum vertex cover:

$$nck(\{a\},\{0\},soft)\wedge nck(\{b\},\{0\},soft)\wedge nck(\{c\},\{0\},soft)\wedge nck(\{d\},\{0\},soft)\wedge nck(\{e\},\{0\},soft)$$

The resulting Minimum Vertex Cover program is illustrated in Figure 5.

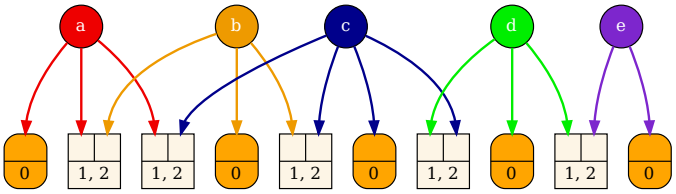

Fig. 5: A visual representation of a minimum vertex cover represented in NchooseK. The filled boxes with rounded corners are soft constraints and act to minimize the number of vertices in the cover.

Conversely, if one wanted to maximize the variables in a particular problem, one could incorporate a constraint with a selection set of one, i.e., $nck(\{v\},\{1\},soft)$. Constraints like this are among the most common soft constraints used in solving minimization or maximization problems, but they can take other forms as well, potentially opening up problems to more efficient solutions. For example, with the Max Cut problem, one solution is to add an extra variable per edge which is set up to be TRUE if and only if the edge has been cut, then add a soft maximization constraint to each of these new variables. This works, but adds many unnecessary variables and greatly increases the number and complexity of constraints. Another option is to instead have a soft constraint of $nck(\{u,v\},\{1\},soft)$ to every edge. This expresses a preference that every edge be cut, and NchooseK attempts to maximize the number of soft constraints which have been met. This solves the Max Cut problem more efficiently.

V. Implementation

One of NchooseK's design goals is to run problems on both circuit-model devices and annealing-model devices. The implementation of NchooseK uses a *quadratic unconstrained binary optimization* (QUBO) format as an intermediate representation. A QUBO seeks to minimize a quadratic equation in which every term comprises either one or two binary variables and a real, constant coefficient. These equations are of the form

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} a_i x_i + \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} b_{i,j} x_i x_j \,, \tag{1}$$

and the objective is to find a set of values for variables $\boldsymbol{x} = x_1,\ldots,x_n$ that minimize $f(\boldsymbol{x})$.

The challenge in creating QUBOs is determining a_i and $b_{i,j}$ coefficients such that the values of $\boldsymbol{x}$ that minimize $f(\boldsymbol{x})$ correspond to the constraints of the target problem. One feature of QUBOs that facilitates the identification of appropriate

coefficients is that QUBOs are compositional with respect to addition. If a problem can be broken into small parts before being translated into simple QUBOs, those QUBOs can be combined via addition to form an overall problem QUBO. NchooseK exploits this property by translating each *nck* constraint individually to a QUBO, using QUBO variables to represent the NchooseK variables, before summing all of them into a final QUBO. The NchooseK implementation finds the coefficients of each per-constraint QUBO by expressing the coefficients in terms of a satisfiability modulo theories (SMT) problem, which it then solves using the Z3 SMT solver [31].

Once an NchooseK program has been compiled to a QUBO, it can be run essentially natively on quantum annealers. NchooseK targets D-Wave quantum annealers by passing the QUBO directly to D-Wave's Ocean API [32]. For circuit-model devices, NchooseK expresses the QUBO as a problem Hamiltonian suitable for use with the QAOA [33] algorithm—a software analogue of the quantum-annealing process. To run on IBM Q circuit-model quantum computers, NchooseK currently invokes the QAOA function provided by IBM's Qiskit library [34]. In either case, each QUBO variable and therefore each NchooseK variable is represented by a qubit, with the state of that qubit corresponding to the value of the variable in the solution. Both of these types of machines may also use additional qubits; this is discussed in more detail in Section VIII.

As an example, consider the (a,b) edge from the minimum vertex cover, constrained by $nck(\{a,b\},\{1,2\})$. We translate this constraint to

$$f(a,b) = ab - a - b\,, \quad a,b \in \{0,1\} \tag{2}$$

which is minimized when at least one of a or b has a value of 1. If both edges (a,b) and (b,c) are constrained with $nck(\{a,b\},\{1,2\}) \wedge nck(\{b,c\},\{1,2\})$, this expression will be transformed into $f(a,b)+f(b,c) = (ab-a-b)+(bc-b-c)$, which in turn is minimized over a, b, and c.

Soft constraints introduce additional complexity to the implementation. There is no inherent distinction between hard and soft constraints in QUBOs. To incorporate soft constraints in NchooseK we consider another property of QUBOs: a QUBO function can be scaled by any positive real-valued factor without altering the values that minimize it. However, when multiple QUBOs are combined, larger-in-magnitude coefficients bias the solution towards minimizing those coefficients' associated variables over the variables associated with smaller-in-magnitude coefficients.

We exploit this property in order to strengthen hard constraints over soft constraints. When creating the QUBO for a hard constraint, we multiply its coefficients by a factor of one higher than the total weight of all soft constraints. Doing so ensures that meeting a single hard constraint reduces the overall value of $f(\boldsymbol{x})$ more than would meeting *all* soft constraints. Nevertheless, the more soft constraints are satisfied, the more $f(\boldsymbol{x})$ is further reduced beyond its value from satisfying hard constraints alone.

```
import nchoosek

env = nchoosek.Environment()
verts = ['a', 'b', 'c', 'd', 'e']
edges = [['a', 'b'], ['a', 'c'], ['b', 'c'],
         ['c', 'd'], ['d', 'e']]
for vert in verts:
    env.register_port(vert)
    env.nck([vert], {0}, soft=True)
for edge in edges:
    env.nck([edge[0], edge[1]], {1, 2})
print(env.solve())
```

Fig. 6: An NchooseK program to solve the minimum vertex cover for the graph shown in Figure 2.

NchooseK is implemented as an embedded domain specific language written in Python. Figure 6 shows the final vertex cover from Figure 5 as a runnable program. Other problems have a similar code structure: the environment is set up, each variable needs to be registered, then each constraint is added with the same syntax as described in this paper. When executed, this program produces the following QUBO:

$$f(a,b,c,d,e) = -11a - 11b - 17c - 11d - 5e + 6ab + 6ac + 6bc + 6cd + 6de$$

This QUBO is isomorphic in the term structure to what one might create by hand, up to the choice of coefficients, which could be chosen differently, e.g., by multiplying by a common positive, real-valued factor.

VI. Complexity Comparison

NchooseK is intended to be more programmer-friendly than lower-level computational models. We therefore compare the complexity of constructing a problem using NchooseK constraints versus directly constructing a QUBO, which is how one would normally program a quantum annealer or set up a QAOA problem for a circuit-model quantum computer. The set of problems considered is summarized in Table I. Besides distinguishing the complexity class of problems in column 2 (NP-hard and NP-complete), we assess at the number of non-symmetric constraints (column 3) to demonstrate the simplicity of setting up a problem using NchooseK as opposed the less intuitive and error-prone task of formulating a QUBO with changing coefficients dependent on problem size. We observe that problems either fall into the group of (a) constant (1 or 2) or (b) linear non-symmetric constraints relative to their input, which illustrates the ease of programming with the NchooseK abstraction.

Definition 7 (Symmetric Constraints)**.** Two NchooseK constraints are considered symmetric with one another if they have the same selection set and their variable collections have the same cardinality.

Problem	Class	# non-symm. constraints	NchooseK constraints	QUBO terms
1. Exact Cover	NP-C	n	n	nN^2
2. Min. Cover	NP-H	n	nN	nN^2
3. Min. Vert. Cover	NP-H	2	$\|V\|+\|E\|$	$\|V\|+\|E\|$
4. Map Color	NP-C	2	$\|V\|+\|E\|n$	$\|V\|n^2+\|E\|n$
5. Clique Cover	NP-C	2	$n\|V\|^2-\|E\|$	$n\|V\|^2-\|E\|$
6. k-SAT	NP-H	2	$n+m$	nm^2+n^2m
7. Max. Cut	NP-H	1	$\|E\|$	$\|E\|+\|V\|$

TABLE I: Sample problems, each listed with its complexity class (NP-complete or NP-hard), number of non-symmetric (different types of) constraints, total number of constraints, and number of terms if expressed directly as a QUBO. For Exact Cover and Minimum Set Cover, n refers to the number of the original elements and N refers to the number of subsets.

For example, the constraints $nck(\{a,b,c\},\{0,2\})$ and $nck(\{b,c,d\},\{0,2\})$ are symmetric, but $nck(\{a,b,c\},\{0,2\})$ and $nck(\{b,c,d\},\{1,2\})$ are non-symmetric, as are $nck(\{a,b,c\},\{0,2\})$ and $nck(\{b,c\},\{1,2\})$.

When simpler to express a problem, we consider two-local Ising Hamiltonians, in which the variables have values of -1 or 1, as opposed to QUBOs, in which the variables have values of 0 or 1. A simple linear transformation maps between the two problem forms.

Columns 4 and 5 indicate the worst-case complexity of problem formulations as NchooseK constraints vs. as QUBOs, respectively. In most cases, the number of constraints generated by NchooseK is lower than the number of equivalent QUBO terms, often reduced by at least one polynomial order with few a few exceptions (minimum cover, clique cover), again a reflection of NchooseK's conciseness as an abstraction.

A. Number of terms and number of constraints

a) Exact set cover: The exact cover problem, which is NP-complete, is covered in depth in a related workshop paper [26] and will be described only briefly here. Given a set E and a set S of subsets of E, find a subset of S such that every element of E is included exactly once. This can be solved with NchooseK by adding a constraint for each element of E with a variable collection containing a variable corresponding to each subset which contains that element, and a selection set of $\{1\}$.

For an exact cover problem with n elements and N subsets, NchooseK requires n constraints, all of which may be non-symmetric and could have a variable collection cardinality of up to N. To formulate the QUBO directly one can adapt the Ising Hamiltonian

$$H_A = A\sum_{\alpha=1}^{n}\left(1-\sum_{i:\alpha\in V_i} x_i\right)^2$$

along the lines of Lucas [35], where α refers to an element and V_i refers to subset i. The factor A may be omitted ($A=1$) in this context. With this equation, removing constant terms and x_i^2 terms (because $x_i=-1$ or 1, which becomes the constant 1 when squared), we have at least n terms, but realistically would encounter more constraints as a problem where each element is only in one subset would be trivial.

If an element is included in m subsets, however, that element alone would introduce $m(m+1)/2$ terms. This direct formulation has a worst-case complexity of $nN(N+1)/2$ or $O(nN^2)$ compared to only $O(n)$ for NchooseK. Both formulations have the same best case.

b) Minimum set cover: The minimum set cover is NP-hard and is the same as the exact cover problem with two key differences: each element of E can be in the solution multiple times, and the goal is to find the smallest subset of S which contains every element of E. This needs the same number of constraints using the same variable collections as the exact set cover, with the selection set now containing every positive integer up to the cardinality of the variable collection. It also requires one soft constraint per subset in order to minimize the number of subsets in the cover.

Both NchooseK and the QUBO formulation for this problem are set up initially as in the exact cover, but require n additional terms to express the minimization; the worst-case complexity is therefore unchanged. It should be noted that in this case these additional terms in the QUBO can be combined, but two different coefficients for these terms need to be chosen and balanced against each other.

c) Minimum vertex cover: For the minimum vertex cover, an NP-hard problem described in Section IV, the NchooseK solution requires $|E|$ hard constraints and $|V|$ soft constraints. The corresponding Hamiltonian is formulated as the QUBO

$$H = A\sum_{uv\in E}(1-x_u)(1-x_v)+B\sum_v x_v$$

where u and v denote vertices. This results in $3|E|+|V|$ terms, the same complexity as NchooseK. The number of mutually non-symmetric constraints for NchooseK is only two; every constraint corresponds either to an edge of the form $nck(\{u,v\},\{1,2\})$ or to a vertex of the form $nck(\{v\},\{0\},soft)$.

d) Map coloring: The map coloring problem with n colors is another NP-complete problem covered in depth by Wilson et al. [26]. The solution uses one-hot encoding, meaning it assigns n variables per vertex, with each variable indicating if the vertex has the associated color. If vertex v has color options 1, 2, and 3, it has variables v_1, v_2, and v_3. If v_1 is TRUE, the other two will be FALSE, and vertex v will have color 1. We need one constraint per vertex to ensure that the vertex has only one color. The variable collection contains n variables, one for each color, and the selection set is $\{1\}$. This problem also requires n constraints per edge. For these constraints, the variable collection contains two variables corresponding to the same color on each of the vertices the edge is connecting. The selection set is $\{0,1\}$, ensuring that two adjacent vertices do not share a color: $nck(\{u_i,v_i\},\{0,1\})$. Every constraint in the map coloring problem will be symmetric with one of these two types.

Our NchooseK solution therefore requires $|V|+n|E|$ constraints. A QUBO using the same one-hot encoding scheme

is

$$\sum_{v}\left(1-\sum_{i=1}^{n}x_{v,i}\right)^{2}+\sum_{(uv)\in E}\sum_{i=1}^{n}x_{u,i}x_{v,j}$$

This uses $|V|n/2(n+1)+|E|n$ terms, leading to $O(|V|n^2+|E|n)$ compared to NchooseK's $O(|V|+|E|n)$. This same trend is seen any time one-hot encoding is used; if n designations are used between V vertices, the QUBO results in $O(Vn^2)$ terms while NchooseK uses only $O(V)$ constraints.

e) Clique cover: The clique cover problem is NP-complete. It requires the coloring of a graph with n colors such that the nodes of each color form a clique within the color. As in the map coloring problem, the solution to this problem requires one-hot encoding with one constraint per vertex. It also needs n constraints per edge *absent* from the graph to ensure that two vertices that are not adjacent do not share a color, similar to the constraints in the map coloring problem. It also needs only two types of non-symmetric constraints.

The clique cover solutions are nearly identical in terms of NchooseK constraints and QUBO terms. Both depend on the number of possible edges not included in E. This is enumerated as $|V|(|V|-1)/2-|E|$. In both cases, the solution requires $O(n|V|^2-|E|)$ terms or constraints.

f) k-satisfiability: The NP-complete k-satisfiability problem establishes m constraints over n boolean variables, each constraint of cardinality k. One or more variables per constraint must have the value of either TRUE or FALSE specified by the constraint. This is similar to how NchooseK constraints are built, with one major exception: NchooseK requires either twice as many variables or much more complicated constraints. The satisfiability constraints can force variables to be either TRUE or FALSE in their constraints without treating them any differently, but NchooseK does not have that capability.

One solution is to create one ancilla variable per original variable, where the ancilla has the opposite value, for example x and $\neg x$. These need a constraint to ensure that they have opposite values, with a selection set of $\{1\}$. Furthermore, one constraint is required per satisfiability constraint with the same variables in the variable collection. The selection set contains every positive integer up to and including k, as seen in Figure 1 for 3-SAT. Using this solution, two non-symmetric types of constraints are used.

The other solution is to create more complicated constraints. Variables can be treated differently from one another by inserting additional copies of them in the variable collection. For the satisfiability constraint $\{x,y,\neg z\}$, the NchooseK specification $nck(\{x,y,z,z,z\},\{0,1,2,4,5\})$ establishes the same constraint, as all instances of z must have the same value. This approach requires fewer NchooseK variables and fewer constraints, but the more complicated constraints run the risk of requiring more ancillary qubits. Copying variables in this manner also changes the number of non-symmetric constraints, giving us a worst case of k. Copying variables further impedes simplicity of expression, which motivated the creation of NchooseK in first place.

When considering its complexity, the dual variable setup of NchooseK for a satisfaction problem with n variables and m constraints requires $n+m$ constraints, while the same problem with larger variable collections requires only m constraints. QUBO formulation of this problem is more complicated. One common solution translates the 3-SAT problem into a Maximum Independent Set problem [36], [37], [35]. This requires km variables, one variable for each variable within each constraint and one term per variable. $k(k-1)m/2$ terms are required between the variables within constraints. Additional terms result from each instance of TRUE/FALSE versions of the variables—if there are i constraints with x and j with $\neg x$, ij terms would be needed to ensure that a variable never has more than one value. In the worst case, this amounts to $m^2k/4$, giving the QUBO a worst-case complexity of $O(km^2+k^2m)$, compared to the NchooseK worst case of $O(n+m)$.

g) Maximum cut: The NP-hard max cut problem is one of the simplest to express in NchooseK: only one soft constraint is needed per edge. The variable collection contains the vertices of the edge, and the selection set is $\{1\}$. These soft constraints ensure that as many vertices as possible have the opposite value to their adjacent vertices. All constraints are symmetric with one another. The max cut problem produces an equal number of NchooseK constraints and Ising terms: $O(|E|)$. However, conversion from Ising to QUBO increases the complexity to $O(|E|+|V|)$ for this particular problem.

B. Generated versus manually produced QUBOs

As NchooseK translates to QUBOs before solving on both gate-based and annealing devices, an important question then is how these translated QUBOs compare to handcrafted QUBOs for the same problem.

QUBO creation is itself computationally difficult. NchooseK uses the Z3 SMT solver [31] to map an individual constraint to a QUBO. For every problem discussed in this paper with the exception of the satisfaction problem and minimum set cover, the QUBO used in NchooseK is the same as the handcrafted QUBO for that problem. This holds regardless of problem size for three reasons:

- NchooseK converts each constraint individually. In most of the problems discussed here, extending the problem means adding additional symmetric constraints (e.g., $nck(\{a,b\},\{0,1\})$ and $nck(\{c,d\},\{0,1\})$). These additional constraints will be converted to QUBOs with the same performance as the previous ones.
- QUBOs are compositional. Two constraints which have been converted into QUBOs are combined with simple addition, meaning that the number of constraints used has no effect on the efficacy of the conversion.
- Constraints with a selection set of $\{1\}$ are trivial to convert to a QUBO, even for large variable collections. No efficacy of conversion is lost for those problems in which extending the problem likewise extends the size of the variable collection, such as adding additional colors in the map coloring problem or subsets in the exact cover problem.

Discussion: Many problems require the introduction of ancillary variables to enable their expression as a QUBO. For example, the NchooseK constraint $nck(\{a,b,c\},\{1,3\})$ cannot be expressed as a three-variable QUBO; it requires a fourth, ancillary variable for an additional degree of freedom in computing the QUBO coefficients. In the minimum set cover problem, constraints with a large variable collection and a large selection set will occasionally have ancillary variables added, whereas there are none in the handmade QUBO for the same problem. Even in this case, the number of additional terms is upper-bounded by $O(nN^2)$. Satisfiability problems exhibit a similar difference in the number of ancillary variables between NchooseK and handmade QUBOs.

C. Ease of construction

Setting up a problem in NchooseK is simpler and more intuitive than setting up the same problem directly as a QUBO even though the number of NchooseK constraints is often similar to the number of QUBO terms. This is due to the fact that constraints are often symmetric across variable sets, and their corresponding selection sets correspond to the problem specification. That is, a constant number of constraint forms tend to be replicated over variable permutations. In contrast, QUBO coefficients change as problem sizes change, and for some constraints ancillary variables may be required. It is not apparent from a problem formulation how many ancillary variables, if any, will be required.

Wilson et al. [26] examine the difference in creating an NchooseK and a QUBO for the equation $A \oplus B = C$. We reiterate their conclusions here: To write an XOR equation $c = a \oplus b$ in NchooseK, the constraint $nck(\{a,b,c\},\{0,2\})$ can easily be obtained by inspection of the XOR truth table. To write the same equation as a QUBO, a number of algebraic transformations are needed. In addition, this equation requires an ancillary variable. The final QUBO is given by

$$f(a,b,c,\kappa) = a+b+c+4\kappa \\ -2ab-2ac-4a\kappa-2bc-4b\kappa+4c\kappa\,, \quad (3)$$

where κ is an ancillary variable without which $f(a,b,c)$ cannot be expressed as a QUBO.

Not only are QUBOs difficult to create by hand, but, as is apparent from Eq. 3, QUBOs are also not particularly human-readable. This is especially true when ancillary variables are used. Compared to $nck(\{a,b,c\},\{0,2\})$, Eq. 3 is complex and obtuse.

VII. Experimental Setup

We ran a variety of experiments on IBM's 65-qubit circuit-based machine, ibmq_brooklyn [27], and one of D-Wave's annealing machines, Advantage 4.1 [38]. In the case of the circuit-based machines, running the program relies on preparing a subroutine (a Hamiltonian function known as a "phase separator") for the Quantum Approximate Optimization Algorithm (QAOA) [33]. QAOA sequentially runs multiple circuits—in our case, 4000 times each—which produce a single result. In contrast, the annealing machines run a single circuit multiple times—in our case, 100. Each run produces a result. For the experiments described in this section we consider only the best (lowest-energy) result.

All problems in Section VI are either NP-hard or NP-complete. They fall under three categories. (1) Problems exclusively with soft constraints (NP-hard): max cut; (2) Problems with a mix of hard and soft constraints (NP-hard): minimum vertex cover and minimum set cover; (3) Problems exclusively with hard constraints (NP-complete): clique cover, map coloring, satisfaction, and exact cover. Of these problems, only those without soft constraints could be solved by the original NchooseK abstraction prior to us adding soft constraints in this paper, and only map coloring and exact cover had been discussed in prior NchooseK work [26] and only for small problems.

In the world of classical computing, metrics tend to focus on execution time. In contrast, the noise of contemporary quantum devices forces researchers to assess *which, if any, of the provided answers are correct in the first place*. To this end, we establish the following terminology for NchooseK:

Definition 8 (Optimal, suboptimal and incorrect)**.** An NchooseK solution over h hard and s soft constraints is *optimal* if all hard and as many soft constraints as possible are satisfied; it is *suboptimal* if all hard (but less than maximum soft) constraints are satisfied; and it is *incorrect* if fewer than h hard constrains are satisfied.

The rationale here is that for problems only using hard constraints, an optimal solution requires full constraint satisfaction, but more than one optimal result may exist. For mixed hard/soft problems, suboptimal solutions still meet all hard constraints but not the maximum number of soft ones, which provides a solution that can be considered non-minimal.

We determined if the results with soft constraints were optimal by checking against the Z3 solver, which solves the problems classically. For mixed problems run on ibmq_brooklyn, results were optimal at smaller scale before becoming suboptimal and then incorrect at larger scale. That is, there seems to be a discrete barrier to optimal solutions. Exposing the same problems to Advantage 4.1 resulted in more suboptimal solutions than optimal ones. Because we are more interested in optimal solutions, we report how many optimal solutions were found.

Subsequent experiments focus on how complex NchooseK problems can become before only incorrect answers are returned. Scaling up the problems from Section VI, we study how the addition of variables and constraints affects the answers obtained. The clique cover problem and map coloring problems require many more qubits than the others. Up to the limit of these two problems, which varies depending on the physical machine, all of the graph problems (Minimum Vertex Cover, Max Cut, Clique Cover, and Map Coloring) are performed on the same graphs.

We ran two different scaling studies: vertex scaling and edge scaling. For vertex scaling, each iteration adds a clique

of three vertices connected to the previous iteration by two edges up to 33 vertices. After 33 vertices the scaling continues in larger increments until the max cut and minimum vertex cover problems use all of the qubits on the IBM machine, and correct (optimal/suboptimal) results are no longer found on the D-Wave system.

For edge scaling, 12 vertices are used—this is where the clique cover problem fails on the D-Wave system. The first one to fail under vertex scaling is the clique cover problem on Advantage 4.1. This problem initially has four cliques and 18 edges. Six or seven edges are added each time up until 48 edges, where adding a single edge between any two disconnected vertices would allow it to be covered by only three cliques. More edges are then added up until 63 edges, at which point adding another edge would allow it to be covered by only two cliques. In this region, the clique cover problem is run with a target of both three and four cliques for comparison.

For the exact cover, minimum set cover, and satisfaction problems, each problem is generated randomly in increasing size with the exact cover and minimum set cover using the same sets and subsets. The k-satisfiability problems are all 3-SAT problems, i.e., every satisfiability constraint contains three terms. The same problems are run on each type of machine.

VIII. Results

A. D-Wave Advantage 4.1

Figure 7 presents measurements the percentage of results (y axis) that are optimal (as opposed to suboptimal or incorrect) over the number of qubits (x axis) on the D-Wave system. With the exception of the exact set problem, the problems with soft constraints generally perform worse than problems exclusively using hard constraints. This is due to the fact that in mixed problems hard constraints receive a higher bias (in terms of constraint factors) than soft constraints. This makes the energy gap relatively small between one solution and another with an additional soft constraint satisfied. If we, instead, reported the percentage of optimal *and* suboptimal results in the y axis, mixed problems would have a higher success rate (omitted due to space). We also observed that the total number of optimal+suboptimal solutions for mixed problems is larger than the number of optimal solutions for hard ones using similar numbers of qubits.

The number of qubits and the connectivity between them for D-Wave's annealing devices are important considerations. First, the Advantage 4.1 system has 5,640 qubits so any problem that requires more will not be able to be run on that machine. Second, problem variables (e.g., nodes of a graph) are often coupled to many other variables. Given the physical qubit graph topology of a D-Wave device, a variable may need to be mapped to a *chain* of qubits to establish these couplings. Hence, the more densely connected the problem, the more qubits are required to represent each variable. This ratio tends to become significant for larger problems.

This explains why the number of qubits used on D-Wave systems relates not only to the number of NchooseK variables used, but also to the number of constraints, which affect the

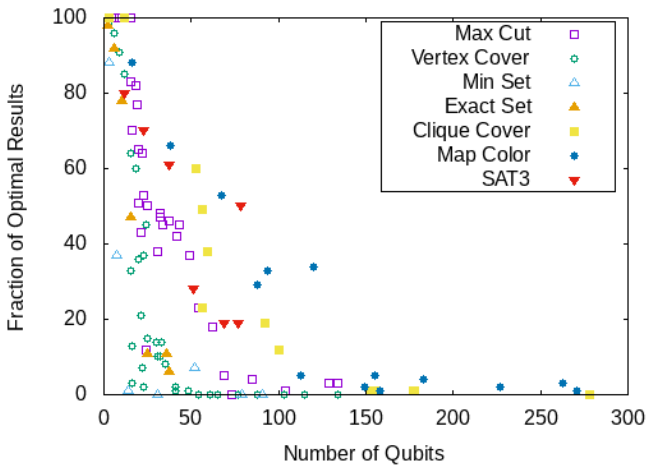

Fig. 7: Fraction of optimal results on D-Wave systems versus number of qubits.

number of connections needed on the physical annealing device. For the clique cover, 48 variables and 18 edges requires 188 qubits, but increasing the number of edges *reduces* the number of constraints for this particular problem formulation. For 37 edges, optimal results are found again as only 132 qubits are needed. At the extreme of 63 edges, still using 48 variables, only 52 qubits are used, increasing the success rate to 65%.

In fact, reducing the number of constraints can have as great an effect on the accuracy as reducing the number of variables does. For the clique cover again with 48 variables, increasing the number of constraints from 24 to 36 results in a drop in success rate from 65% to 20%. These solutions use 52 and 55 qubits, respectively, i.e., only a small increase in the number of qubits is imposed. Instead, if we use 27 variables and 78 constraints, 57 qubits are required with a success rate of just 39%. Decreasing the number of variables used from 48 to 27 still results in a significant drop in success rate because the number of constraints increases dramatically, even though the number of qubits used is similar.

B. IBM Q Brooklyn

The problems performed worse on ibmq_brooklyn than on Advantage 4.1; different problems failed to find an optimal result at a lower number of variables and constraints than used for annealing. Despite this, it should be stressed that using QAOA a single result is returned and found to be optimal or not, while using an annealer the problem is considered to be solved correctly if any of the hundred solutions returned is optimal.

As with annealing devices, the number of qubits is an important consideration when utilizing circuit-model devices. The most obvious reason is that the machine has far fewer qubits; no NchooseK problem with more than 65 variables can be mapped onto ibmq_brooklyn. Another factor is that some qubits and some connections between qubits are worse than others in terms of noise. Small problems may select the best performing qubits on a given device, while larger ones must

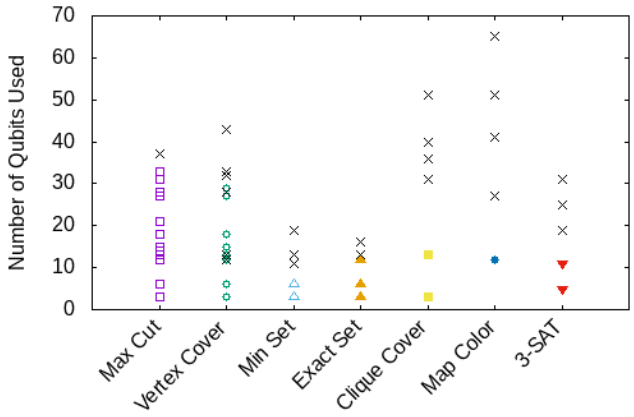

Fig. 8: Optimal (colored tics) and suboptimal or incorrect (block × tics) results of the QAOA problems for ibmq_brooklyn vs. number of qubits used.

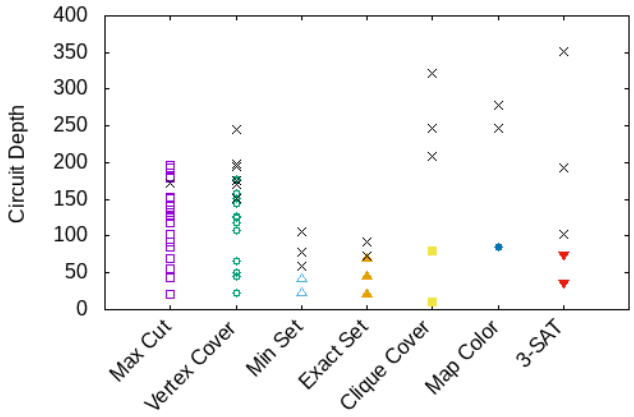

Fig. 9: Optimal (colored tics) and suboptimal or incorrect (block × tics) results of the QAOA problems for ibmq_brooklyn vs. circuit depth. Six failed clique cover problems were omitted for clarity; they used circuits of depth 432, 516, 537, 676, 697, and 717.

use more error-prone ones as the fraction of utilized qubits increases. Due to limited qubit connectivity in the physical topology, circuit-model machines cannot directly perform two-qubit operations on arbitrary pairs of qubits. Hence, they must frequently swap the state of adjacent qubits in sequence to move pairwise interactions to physical neighbors. The compiler sometimes prioritizes a shorter but lower-quality (higher-noise) path of swaps. This affects solution quality as the number of qubits and circuit depth increase.

Recall from the discussion in Section VI-B that the QUBO formulation of a problem often requires the introduction of ancillary variables. This explains why the number of qubits sometimes exceeds the number of variables in an NchooseK problem.

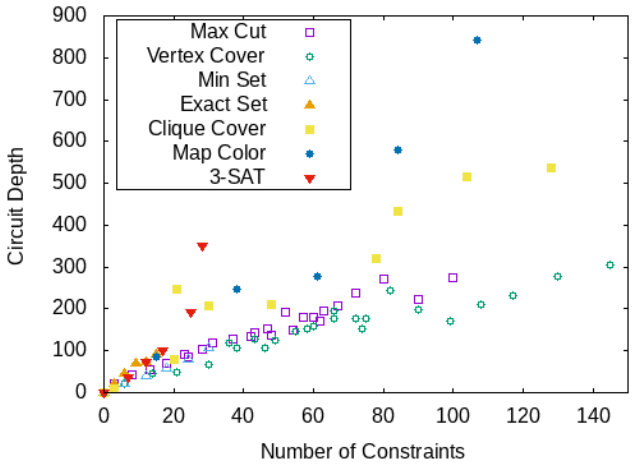

Fig. 10: The depth of QAOA circuits with respect to the number of constraints in the NchooseK problem.

Figure 8 depicts the number of qubits used (y axis) for problems (x axis) from Table I indicating both optimal (colored tics) and suboptimal (block × tics) results. We observe that there is a correlation between the number of qubits and obtaining optimal results. Figure 9 depicts results for the same programs (x axis) over the circuit depth (y axis) measured as the number of gates in the longest path of a single QAOA circuit with the same tic mark colors as before. While each QAOA runs around 30 different circuits (slight variations are due to convergence properties), these circuits differ by the parameters of the gates (qubit rotation angles), not the type or number of gates. Circuit depth is an important considerations when experimenting with circuit model devices. This is true not only because each gate adds a small amount of probabilistic error (noise) to a circuit, but also because a deeper circuit needs to stay active on the machine longer, leading to an increase in chance of qubits decohering before results can be measured.

These two figures show the trends in correctness for the different problems. Note that the edge study and the vertex study are both included for the map problems. This explains the low qubit failures for the vertex cover seen in Figure 8: Even using few qubits, a sufficient number of constraints will add enough complexity to the problem to cause a failure. This relationship between circuit depth, which can be thought of as a simplistic measure of circuit complexity, and the number of constraints is exposed in Figure 10, which depicts the number of constraints (x axis) over circuit depth (y axis) for each problem type. The general trend shows increasing depth as more variables and constraints are added during problem scaling, albeit at different rates per problem, i.e., in a problem-specific manner. Exceptions include the minimum vertex cover: At 30 variables and 82 constraints, it uses 32 qubits with a depth of 245. At 33 variables and 90 constraints, only 33 qubits are used with a depth of 199. Hence, depth is not always related to the success rate (optimality) of results. This was also visible in Figure 9, where a suboptimal solution for Max Cut at depth

172 is followed by optimal solutions at 179 and thereafter. Nonetheless, these problems scale up to mid to high teens of qubits on the IBM device (25–100% of qubit utilization) and into the hundreds of qubits on the D-Wave device (4–6% of physical qubit utilization).

C. Timing

Given the limitations of contemporary quantum computers in terms of qubit counts, coherence times, control precision, resilience to noise, and qubit connectivity, raw execution time is generally not the focus of current quantum-computing research. Nevertheless, we include a brief summary of the time taken and the bottlenecks of running a sample of our problems. The client-side operations for experiments described in this section were performed on a 4 GHz Quad-Core i7 processor with 40 GB memory.

For the problems run on the IBM systems, each execution of the QAOA algorithm implicitly submits approximately 25 to 35 jobs, the number of which does not discernibly depend on the size of the problem. Each job comprised 4000 shots, the default for Qiskit's QAOA, and took between 7 and 23 seconds.

We were unable to determine any correlation between problem size and time per job. Figure 11 shows a box plot of job run time (y axis) versus the number of variables (x axis) in the original NchooseK environment.

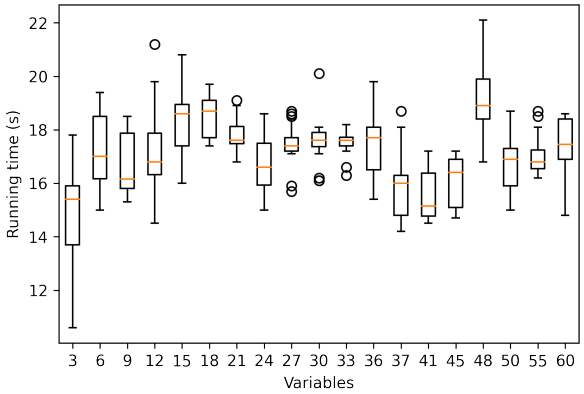

Fig. 11: The run time of QAOA circuits with respect to the number of variables used.

Aside from the time spent running on the quantum computer, a job also requires computation time on the IBM server. It takes a few seconds to create, transpile, and validate a job plus an indeterminate amount of time waiting in the queue for access to the machine. All together, our jobs spent roughly 500 seconds on IBM's servers, not counting communication or queue time. This time can vary greatly, depending on how full the queue is with unrelated jobs.

On the client side, some amount of time is spent generating the QUBO and working with the optimizer. Relative to the amount of time spent in IBM's cloud, the time spent creating the QUBO is not only negligible—taking a second or less—but is also overshadowed by the variance in communication and number of jobs run, not to mention the indeterminate time spent waiting in the job queue. Finally, the classical optimization step in the QAOA process typically takes two to three seconds per job. All together, even a small problem takes about 500 seconds plus queueing time to solve. IBM has recently started offering the option to run QAOA more closely tied to the IBM servers through Qiskit Runtime [39], which should cut down on communication time and possibly time spent on classical optimization.

The problems run on the D-Wave systems were submitted as a single job consisting of 100 samples. According to the D-Wave documentation [40], each job has a single, relatively long programming step (observed to be on the order of 15ms) in addition to the cost of the 100 samples. The cost of a sample includes the cost of the anneal itself, a parameter that can be defined by the user (our experiments used the default of 20µs); a readout time with a cost that is usually 3–4 times as long as the annealing time; and an added delay between each readout and the subsequent anneal (about 20µs each). The total time for the 100 samples is slightly less than the time than the programming step. Finally, a few more milliseconds are needed for post-processing. Neglecting the time in the queue, our jobs each spent about 30ms apiece on the Advantage system.

A large cost on the client side is the conversion of constraints to individual QUBOs. This procedure is currently under development and is not yet optimized. Specifically, it redundantly computes QUBOs for symmetric constraints instead of caching previously computed QUBOs. Due to this wasted computation, the total time to compile a complete NchooseK problem to a QUBO is 40–50x the time needed for direct (non-QUBO) solution by the Z3 solver of problems of the size covered in this paper. After constructing the QUBO, preparing it to send to a D-Wave system takes approximately an additional 40ms.

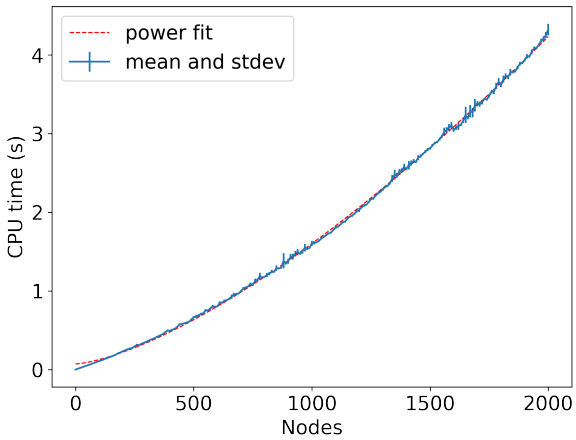

Fig. 12: The run time of minimum vertex cover on Z3. Each problem was run 30 times on a circulant graph with the indicated number of nodes.

Z3 is a highly optimized classical SMT solver, and it is

able to solve each of the problems contained here in less than three seconds. It can also solve problems much larger than can fit on current quantum hardware, scaling quite well. The minimum vertex cover problems we ran fit very close to a polynomial equation as shown in Figure 12. However, when presenting Z3 with problems after they have been translated into a QUBO, many of them perform quite poorly: solving a minimum vertex cover problem with 10 vertices of degree 3 takes less than a second while 20 vertices takes a minute and a half, and 30 vertices takes multiple hours. NchooseK's classical Z3 back end runs faster than either of the two quantum back ends on current quantum hardware. However, we note that the D-Wave Advantage machine completes the optimization step proper in a fraction of a second. This suggests that there exists opportunities to close the performance gap between D-Wave and Z3 through additional software optimizations.

IX. Future Work

One of the current limitations of NchooseK is its reliance on QAOA for operation on circuit-based machines. We are investigating different methods of converting NchooseK programs into quantum circuits. This may involve abandoning QAOA entirely for an alternative variational quantum algorithm, or it may involve devising NchooseK-specific or problem-specific customizations to QAOA's problem and mixer Hamiltonians. This is the basic concept underlying the Quantum Alternating Operator Ansatz [41] (a refinement of the Quantum Approximate Optimization Algorithm that is also abbreviated QAOA). The custom mixers used in this version of QAOA seem especially appropriate to NchooseK problems with both hard and soft constraints.

X. Conclusions

NchooseK is an effective and relatively simple method of expressing and solving NP-complete problems on both quantum annealers and circuit-based quantum computers. Our contribution is a generalization of NchooseK to include soft constraints, which widens the scope of problems that can be expressed to include NP-hard problems. We show that NP-complete and NP-hard problems can be solved using NchooseK on current, noisy, intermediate-scale quantum (NISQ) devices utilizing up to 65 qubits on IBM's devices and hundreds of qubits on D-Wave's annealing devices. One contribution of NchooseK is given by its intuitive problem formulation with (typically) only a constant or a linear number of non-symmetric constraints, whereas manual QUBO formulations are more complex and require computing different coefficients depending on problem size. Another contribution is that NchooseK enables a transformation even of soft constraints into QUBOs, which make a suitable intermediate representation for enabling portability across the circuit model and the annealing model. QUBO generation is fully automated, and NchooseK produces QUBOs that are comparable to those painstakingly developed by hand.

Acknowledgments

Research presented in this paper was supported by the Laboratory Directed Research and Development program of Los Alamos National Laboratory under project number 20210397ER. Los Alamos National Laboratory is operated by Triad National Security, LLC for the National Nuclear Security Administration of U.S. Department of Energy (contract no. 89233218CNA000001). This work was also supported in part by LANL subcontract 725530 and by NSF awards DMR-1747426, PHY-1818914, OAC-1917383, MPS-2120757, and CISE-2217020.

References

[1] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018.

[2] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. McClean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, and H. N. J. M. Martinis, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, Oct. 23, 2019.

[3] E. Pednault, J. Gunnels, D. Maslov, and J. Gambetta, "On 'quantum supremacy'," Oct. 2019. [Online]. Available: https://www.ibm.com/blogs/research/2019/10/on-quantum-supremacy/

[4] S. Aaronson, "Shtetl-optimized," Sep. 2019. [Online]. Available: https://www.scottaaronson.com/blog/

[5] N. Jouppi, C. Young, N. Patil, and D. Patterson, "Motivation for and evaluation of the first tensor processing unit," *IEEE Micro*, vol. 38, no. 3, pp. 10–19, 2018.

[6] T. R. Louw and S. N. McIntosh-Smith, "Using the Graphcore IPU for traditional HPC applications," in *3rd Workshop on Accelerated Machine Learning (AccML), HiPEAC 2021 Conference*, ser. EasyChair preprint, no. 4986. European Network on High-performance Embedded Architecture and Compilation, Jan. 18, 2021. [Online]. Available: https://easychair.org/publications/preprint/ztfj

[7] R. Prabhakar, S. Jairath, and J. L. Shin, "SambaNova SN10 RDU: A 7nm dataflow architecture to accelerate software 2.0," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 350–352.

[8] J. Clarke and F. K. Wilhelm, "Superconducting quantum bits," *Nature*, vol. 453, no. 7198, p. 1031, 2008.

[9] J. I. Cirac and P. Zoller, "Quantum computations with cold trapped ions," *Physical review letters*, vol. 74, no. 20, p. 4091, 1995.

[10] IBM, "IBM Q Experience," https://quantumexperience.ng.bluemix.net/qx.

[11] "Welcome to quantum cloud services—QCS documentation," 2022. [Online]. Available: https://docs.rigetti.com/qcs/

[12] K. Wright, K. M. Beck, S. Debnath, J. M. Amini, Y. Nam, N. Grzesiak, J.-S. Chen, N. C. Pisenti, M. Chmielewski, C. Collins, K. M. Hudek, J. Mizrahi, J. D. Wong-Campos, S. Allen, J. Apisdorf, P. Solomon, M. Williams, A. M. Ducore, A. Blinov, S. M. Kreikemeier, V. Chaplin, M. Keesan, C. Monroe, and J. Kim, "Benchmarking an 11-qubit quantum computer," *arXiv:1903.08181*, 2019. [Online]. Available: https://arxiv.org/abs/1903.08181

[13] K. Boothby, C. Enderud, T. Lanting, R. Molavi, N. Tsai, M. H. Volkmann, F. Altomare, M. H. Amin, M. Babcock, A. J. Berkley, C. B. Aznar, M. Boschnak, H. Christiani, S. Ejtemaee, B. Evert, M. Gullen, M. Hager, R. Harris, E. Hoskinson, J. P. Hilton, K. Jooya, A. Huang, M. W. Johnson, A. D. King, E. Ladizinsky, R. Li, A. MacDonald, T. M. Fernandez, R. Neufeld, M. Norouzpour, T. Oh, I. Ozfidan, P. Paddon, I. Perminov, G. Poulin-Lamarre, T. Prescott, J. Raymond, M. Reis,

C. Rich, A. Roy, H. S. Esfahani, Y. Sato, B. Sheldan, A. Smirnov, L. J. Swenson, J. Whittaker, J. Yao, A. Yarovoy, and P. I. Bunyk, "Architectural considerations in the design of a third-generation superconducting quantum annealing processor," Aug. 5, 2021, arXiv:2108.02322v1 [quant-ph].

[14] M. Aramon, G. Rosenberg, E. Valiante, T. Miyazawa, H. Tamura, and H. G. Katzgraber, "Physics-inspired optimization for quadratic unconstrained problems using a digital annealer," *Frontiers in Physics*, vol. 7, 2019.

[15] S. Boixo, T. Albash, F. M. Spedalieri, N. Chancellor, and D. A. Lidar, "Experimental signature of programmable quantum annealing," *arXiv:1212.1739*, 2012. [Online]. Available: http://arxiv.org/abs/1212.1739

[16] D. Bacon, S. T. Flammia, and G. M. Crosswhite, "Adiabatic quantum transistors," *Physical Review X*, vol. 3, pp. 021 015:1–17, Jun. 14, 2013.

[17] C. Yuan, C. McNally, and M. Carbin, "Twist: Sound reasoning for purity and entanglement in quantum programs," *Proceedings of the ACM on Programming Languages*, vol. 6, no. POPL, pp. 30:1–32, Jan. 2022.

[18] B. Bichsel, M. Baader, T. Gehr, and M. Vechev, "Silq: A high-level quantum language with safe uncomputation and intuitive semantics," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. New York, New York, USA: Association for Computing Machinery, 2020, pp. 286–300.

[19] K. Svore, A. Geller, M. Troyer, J. Azariah, C. Granade, B. Heim, V. Kliuchnikov, M. Mykhailova, A. Paz, and M. Roetteler, "Q#: Enabling scalable quantum computing and development with a high-level DSL," in *Proceedings of the Real World Domain Specific Languages Workshop 2018*. New York, New York, USA: Association for Computing Machinery, 2018, pp. 7:1–10.

[20] D. S. Steiger, T. Häner, and M. Troyer. (2018, Jan. 29,) ProjectQ: An open source software framework for quantum computing. arXiv:1612.08091v2 [quant-ph].

[21] J. Paykin, R. Rand, and S. Zdancewic, "*2*WIRE: A core language for quantum circuits," in *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. New York, New York, USA: Association for Computing Machinery, 2017, pp. 846–858.

[22] A. JavadiAbhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F. T. Chong, and M. Martonosi, "ScaffCC: Scalable compilation and analysis of quantum programs," *Parallel Computing*, vol. 45, pp. 2–17, 2015.

[23] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron, "Quipper: A scalable quantum programming language," in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*. New York, New York, USA: Association for Computing Machinery, 2013, pp. 333–342.

[24] D-Wave Systems Inc. D-Wave Ocean software documentation. [Online]. Available: https://docs.ocean.dwavesys.com/

[25] H. Khetawat, A. Atrey, G. Li, F. Mueller, and S. Pakin, "Implementing NChooseK on IBM Q quantum computers," in *Reversible Computing*, ser. Lecture Notes in Computer Science, M. K. Thomsen and M. Soeken, Eds., vol. 11497. Springer, Nov. 2019, pp. 209–223.

[26] E. Wilson, F. Mueller, and S. Pakin, "Mapping constraint problems onto quantum gate and annealing devices," in *2021 IEEE/ACM Second International Workshop on Quantum Computing Software (QCS)*. IEEE, Nov. 15, 2021, pp. 110–117.

[27] IBM Quantum Services. ibmq_brooklyn. Accessed 20-May-2022. [Online]. Available: https://quantum-computing.ibm.com/services?services=systems&system=ibmq_brooklyn

[28] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, M. S. Alam, S. Ahmed, J. M. Arrazola, C. Blank, A. Delgado, S. Jahangiri, K. McKiernan, J. J. Meyer, Z. Niu, A. Száva, and N. Killoran, "PennyLane: Automatic differentiation of hybrid quantum-classical computations," 2018. [Online]. Available: https://arxiv.org/abs/1811.04968

[29] A. J. McCaskey, D. I. Lyakh, E. F. Dumitrescu, S. S. Powers, and T. S. Humble, "XACC: A system-level software infrastructure for heterogeneous quantum–classical computing," *Quantum Science and Technology*, vol. 5, no. 2, pp. 024 002:1–23, Feb. 2020.

[30] R. S. Smith, M. J. Curtis, and W. J. Zeng. (2017, Feb. 17,) A practical quantum instruction set architecture. Rigetti Computing, Inc. ArXiv:1608.03355 [quant-ph].

[31] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, C. R. Ramakrishnan and J. Rehof, Eds., vol. 4963. Budapest, Hungary: Springer, Mar. 29–Apr. 6, 2008, pp. 337–340.

[32] D-Wave Systems, Inc., "D-Wave Ocean software documentation, revision 6f16a2d3," https://ocean.dwavesys.com/, accessed 2-Oct-2021.

[33] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," Center for Theoretical Physics, Massachusetts Institute of Technology, Tech. Rep. MIT-CTP/4610, 2014.

[34] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F. J. Cabrera-Hernández, J. Carballo-Franquis, A. Chen, C.-F. Chen, J. M. Chow, A. D. Córcoles-Gonzales, A. J. Cross, A. Cross, J. Cruz-Benito, C. Culver, S. D. L. P. González, E. D. L. Torre, D. Ding, E. Dumitrescu, I. Duran, P. Eendebak, M. Everitt, I. F. Sertage, A. Frisch, A. Fuhrer, J. Gambetta, B. G. Gago, J. Gomez-Mosquera, D. Greenberg, I. Hamamura, V. Havlicek, J. Hellmers, Ł. Herok, H. Horii, S. Hu, T. Imamichi, T. Itoko, A. Javadi-Abhari, N. Kanazawa, A. Karazeev, K. Krsulich, P. Liu, Y. Luh, Y. Maeng, M. Marques, F. J. Martín-Fernández, D. T. McClure, D. McKay, S. Meesala, A. Mezzacapo, N. Moll, D. M. Rodríguez, G. Nannicini, P. Nation, P. Ollitrault, L. J. O'Riordan, H. Paik, J. Pérez, A. Phan, M. Pistoia, V. Prutyanov, M. Reuter, J. Rice, A. R. Davila, R. H. P. Rudy, M. Ryu, N. Sathaye, C. Schnabel, E. Schoute, K. Setia, Y. Shi, A. Silva, Y. Siraichi, S. Sivarajah, J. A. Smolin, M. Soeken, H. Takahashi, I. Tavernelli, C. Taylor, P. Taylour, K. Trabing, M. Treinish, W. Turner, D. Vogt-Lee, C. Vuillot, J. A. Wildstrom, J. Wilson, E. Winston, C. Wood, S. Wood, S. Wörner, I. Y. Akhalwaya, and C. Zoufal, "Qiskit: An open-source framework for quantum computing," Jan. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.2562111

[35] A. Lucas, "Ising formulations of many NP problems," *Frontiers in Physics*, vol. 2, pp. 5:1–5:15, 2014.

[36] V. Choi, "Different adiabatic quantum optimization algorithms for the NP-complete exact cover problem," *Proceedings of the National Academy of Sciences*, vol. 108, no. 7, Jan. 2011. [Online]. Available: https://doi.org/10.1073%2Fpnas.1018310108

[37] T. Gabor, S. Zielinski, S. Feld, C. Roch, C. Seidel, F. Neukart, I. Galter, W. Mauerer, and C. Linnhoff-Popien, "Assessing solution quality of 3SAT on a quantum annealing platform," 2019. [Online]. Available: https://arxiv.org/abs/1902.04703

[38] D-Wave Systems, Inc., "D-Wave Advantage system overview," https://www.dwavesys.com/resources/white-paper/the-d-wave-advantage-system-an-overview/, accessed 20-May-2022.

[39] B. Johnson and G. Ben-Shach. (2022, Apr. 12,) Qiskit Runtime primitives make algorithm development easier than ever. Accessed 23-Aug-2022. [Online]. Available: https://research.ibm.com/blog/qiskit-runtime-for-useful-quantum-computing

[40] D-Wave Systems, Inc. Operation and timing. Accessed 20-Jul-2022. [Online]. Available: https://docs.dwavesys.com/docs/latest/c_qpu_timing.html

[41] S. Hadfield, Z. Wang, B. O'Gorman, E. G. Rieffel, D. Venturelli, and R. Biswas, "From the quantum approximate optimization algorithm to a quantum alternating operator ansatz," *Algorithms*, vol. 12, no. 2, 2019.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

Experiments were run on both the DWave quantum annealer Advantage 4.1 and the IBM Quantum circuit device ibm_brooklyn. Three graph problems: minimum vertex cover, max cut, and clique cover are run. There are two scaling studies: vertex scaling and edge scaling. Problems were run with qiskit 0.34.2 and the dwave-ocean-sdk 0.8.5, on python 3.9.12

AUTHOR-CREATED OR MODIFIED ARTIFACTS:

Artifact 1

Persistent ID: `10.5281/zenodo.6916782`

Artifact name: nchoosek_sc22

Reproduction of the artifact with container: Follow the directions in README.txt An anaconda environment is provided in order to set up dependencies; user accounts are necessary with IBM and DWave to use their quantum devices.