Mapping Constraint Problems onto Quantum Gate and Annealing Devices

Ellis Wilson
and Frank Mueller
Department of Computer Science
North Carolina State University
Raleigh, North Carolina 27695-8206
Email: ejwilso2@ncsu.edu, mueller@cs.ncsu.edu

Scott Pakin
Computer, Computational, and Statistical Sciences Division
Los Alamos National Laboratory
Los Alamos, New Mexico 87545
Email: pakin@lanl.gov

2021 IEEE/ACM Second International Workshop on Quantum Computing Software (QCS) | 978-1-7281-8674-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/QCS54837.2021.00016

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

***Abstract*—This work presents NchooseK, a unified programming model for constraint satisfaction problems that can be mapped to both quantum circuit and annealing devices through Quadratic Unconstrained Binary Operators (QUBOs). Our mapping provides an approachable and effective way to program both types of quantum computers. We provide examples of NchooseK being used.**

I. Motivation

Quantum computing has the potential to provide a performance advantage over classical computing, but as a practical technology it is newly emerging. The two main quantum computational models, quantum annealing and the quantum gate model, are distinct from each other as well as from classical programming and lead to different performance and engineering trade-offs.

To program a quantum annealer, one encodes a problem as either a classical, 2-local Ising-model Hamiltonian or a quadratic unconstrained binary optimization (QUBO) problem—the two are isomorphic—and the system uses quantum effects to solve for the two-level (Boolean) inputs that minimize overall energy. To program a gate-model quantum computer, one encodes the problem as a sequence of unitary transformations applied to an input state, and a final measurement projects the result to a vector of Booleans.

The disconnect between these two computing models, as well as the potential of quantum computing, has inspired us to create a new programming model that enables programs to run on quantum annealers, circuit-based quantum computers, and classical computers. This model, called NchooseK, is designed to express a particular type of constraint-satisfaction problem. NchooseK strives to strike a balance between ease of programmability and performance. An early investigation into the use of NchooseK [1] was based on Grover's search algorithm. This paper reports on an improved implementation approach based instead on the quantum approximate optimization algorithm (QAOA) [2].

II. NchooseK Overview

NchooseK is a constraint-based programming model and a specific type of Integer Linear Programming (ILP). Programs consist of Boolean variables and a set of constraints applied to them. Each constraint takes the form, "Given a collection of variables of cardinality N, a subset of them with cardinality K must be *true*".

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 (NchooseK constraint): An NchooseK 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 $n \in \mathbb{Z}_2$ and we associate *false* with 0 and *true* with 1.

Definition 4 (NchooseK program): An NchooseK program is a conjunction of NchooseK constraints, notated $nck(\{N_1\},\{K_1\}) \wedge nck(\{N_2\},\{K_2\}) \wedge \cdots \wedge nck(\{N_n\},\{K_n\})$. The result of executing a program is either an assignment of Boolean values to all variables appearing in all variable collections such that every NchooseK constraint is honored or an indication that no such assignment exists.

Constructing NchooseK constraints involves focusing on the relationships among variables. Consider a variable collection $\{a,b\}$. If a and b must have different values, the constraint is expressed as $nck(\{a,b\},\{1\})$. This indicates that exactly one of a and b must be *true* and therefore the other *false*. If they need to have the same value, instead $nck(\{a,b\},\{0,2\})$ would be used: either zero variables are *true* (so both *false*) or two variables are *true* (so both *true*). As additional examples, $nck(\{a,b\},\{1,2\})$ constrains at least one of a and b to be *true*, and $nck(\{a,b\},\{0,1\})$ constrains at least one to *false*.

The same variable can appear in the variable collection of multiple constraints in an NchooseK program. In this case, the variable takes the same value in both constraints. For example,

978-1-7281-8674-0/21/$31.00 ©2021 IEEE
DOI 10.1109/QCS54837.2021.00016

Authorized licensed use limited to: University of Maryland College Park. Downloaded on July 20,2022 at 17:32:28 UTC from IEEE Xplore. Restrictions apply.

$nck(\{a,b\},\{1\}) \wedge nck(\{b,c\},\{1\})$ is satisfied when both a and c are *true* while b is *false* or when both a and c are *false* and b is *true* but no other combinations.

III. Case Studies

To clarify how one could express a computational problem as an NchooseK program we consider two case studies. Section III-A discusses the exact-cover problem, and Section III-B discusses the map-coloring problem. Each of these presents a concrete example of a problem and discusses the steps needed to formulate this example in terms of NchooseK constraints.

A. Exact Cover

As an example on how to form a problem with NchooseK, consider the exact cover problem: Given a set $E = \{e_1, e_2, \ldots, e_n\}$ of n elements and a set $S = \{s_1, s_2, \ldots, s_m\}$, of m subsets of E, i.e.,$s_i \subseteq E$, the goal is to find some subset of S that includes every element of E exactly once—or report that no such cover exists. We call a subset of S that includes every element of E a "cover". The "exactly once" condition makes this subset an exact cover. Consider the following problem:

$$\begin{aligned} E &= \{a,b,c,d,e,f,g\} \\ S &= \{s_1,s_2,s_3,s_4,s_5,s_6\} \\ s_1 &= \{b,c,e,f\} \\ s_2 &= \{a,d,e\} \\ s_3 &= \{a,d,e,g\} \\ s_4 &= \{a,g,f\} \\ s_5 &= \{c,f\} \\ s_6 &= \{b,g\} \end{aligned}$$

In this case, one solution is the subset $\{s_2, s_5, s_6\}$ of S because this subset contains each of a, b, c, d, e, f, and g exactly once, making it an exact cover. An example of a non-solution is the subset $\{s_1, s_3\}$, which covers E—it includes all seven elements of E—but is not an exact cover because element e occurs twice. The subset $\{s_1, s_2\}$ is also not a solution because it is missing element g, implying that $\{s_1, s_2\}$ does not cover E.

Given that a solution to the exact-cover problem indicates which subsets are in the cover, we include one variable in the corresponding NchooseK problem per element of set S. Specifically, an NchooseK variable v_i is *true* if and only if its associated subset s_i belongs to the cover. The requirements of a valid solution are that (1) each element of E must be included in the cover and (2) no element of E may be in the cover more than once. Because element e of E must be included exactly once, we must constrain exactly one of the variables associated with a subset containing e to *true*. That is, we will include one NchooseK constraint per element of E. These variables and constraints are all that are needed to express the exact cover.

In our example problem, the NchooseK constraint for the element a is

$$nck(\{v_2, v_3, v_4\}, \{1\})$$

because subsets s_2, s_3, and s_4 are the ones that contain element a, and exactly one of them needs to be in the cover. The complete NchooseK program is

$$\begin{aligned} &nck(\{v_2,v_3,v_4\},\{1\}) \wedge && \triangleright\ a \\ &nck(\{v_1,v_6\},\{1\}) \wedge && \triangleright\ b \\ &nck(\{v_1,v_5\},\{1\}) \wedge && \triangleright\ c \\ &nck(\{v_2,v_3\},\{1\}) \wedge && \triangleright\ d \\ &nck(\{v_1,v_2,v_3\},\{1\}) \wedge && \triangleright\ e \\ &nck(\{v_1,v_4,v_5\},\{1\}) \wedge && \triangleright\ f \\ &nck(\{v_3,v_4,v_6\},\{1\}) && \triangleright\ g \end{aligned}$$

These constraints and variables are illustrated graphically in Figure 1.

B. Map Coloring

Another example of solving a problem with NchooseK is the map coloring problem. Given a map of territories, some of which share borders, the map should be colored such that no two territories with a common border have the same color.

An NchooseK solution of the map-coloring problem constrains which territories are colored with which color. This is not a binary choice, unlike the exact cover in which the only question is whether a subset is part of the cover or not. When the solution covers multiple dimensions—in this case, n and m, where n is the number of territories and m is the number of colors—the NchooseK variables have to defined to reflect those dimensions. For the map-coloring problem, this means there are $n \cdot m$ variables, one variable per territory per color. Consider the simple case of two territories, P and Q, and four colors, red, orange, green, and blue. (It can be shown that any two dimensional map can be colored with only four colors [3], [4], which is why we use four colors in this example.) In this case, eight variables are needed: P_{red}, P_{orange}, P_{green}, P_{blue}, Q_{red}, Q_{orange}, Q_{green}, and Q_{blue}.

Ultimately, each territory can be assigned only a single color. Hence, a constraint is needed for each territory, indicating that only one color can be *true* (assigned). For territory P in our example, this NchooseK constraint is expressed as $nck(\{P_{red}, P_{orange}, P_{blue}, P_{green}\}, \{1\})$. An analogous constraint is specified for territory Q.

Requiring that no bordering territories share a color requires additional constraints, namely one constraint per border per color, indicating both cannot be *true* (i.e., cannot be colored identically). For the border between territory P and Q, this NchooseK constraint is expressed as

$$\begin{aligned} &nck(\{P_{red},Q_{red}\},\{0,1\}) \wedge nck(\{P_{orange},Q_{orange}\},\{0,1\}) \\ &\wedge nck(\{P_{blue},Q_{blue}\},\{0,1\}) \wedge nck(\{P_{green},Q_{green}\},\{0,1\}). \end{aligned}$$

The selection sets in the above must include both 0 and 1 because neither territory may be a given color. For example, $nck(\{P_{red}, Q_{red}\}, \{0,1\})$ allows for either P or Q to be red or for neither P nor Q to be red. The only case that is prohibited is both P and Q being red.

This example is illustrated in Figure 2. A map with more regions would follow the same pattern but with a corresponding increase in the number of constraints.

Authorized licensed use limited to: University of Maryland College Park. Downloaded on July 20,2022 at 17:32:28 UTC from IEEE Xplore. Restrictions apply.

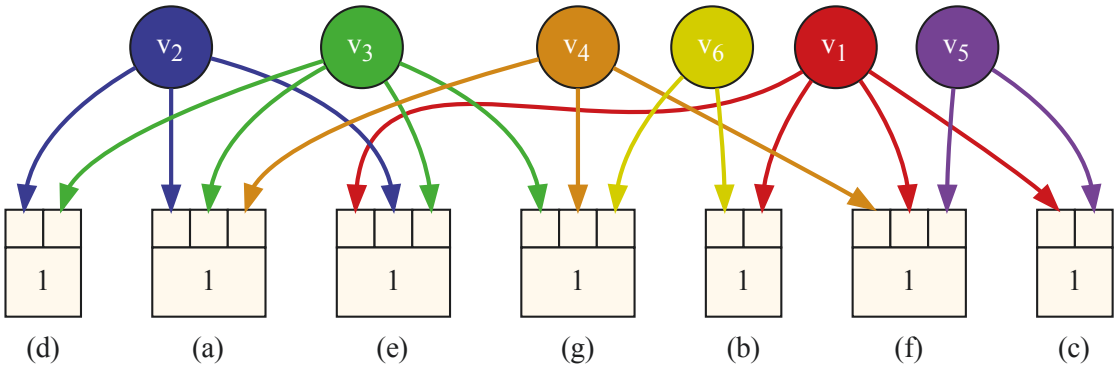

Fig. 1: A visualization of the exact cover example problem represented in NchooseK. Each circle corresponds to one of the variables in the NchooseK problem or subset in the original problem. The colors are for convenience only, to help distinguish the arrows. The boxes represent NchooseK constraints. Within each box, the small squares represent the variable collection—containing the variables pointing to them—and the text indicates the selection set. In this case the selection set is $\{1\}$ for each constraint. Parenthesized letters underneath each box indicate the element of set E in the original problem.

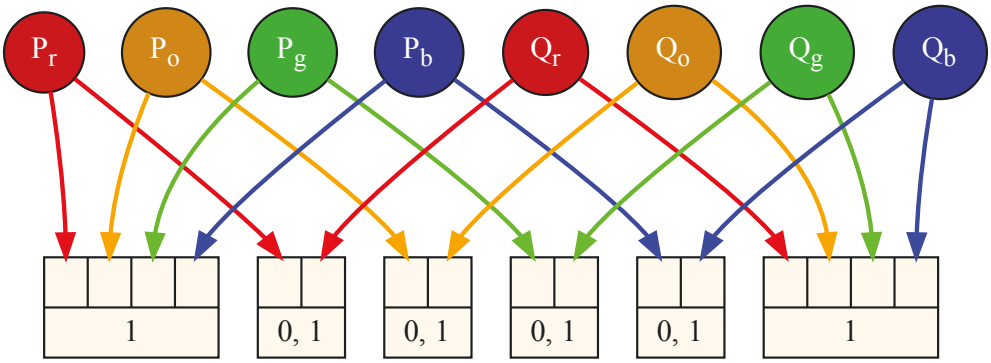

Fig. 2: A visualization of the map coloring example problem represented in NchooseK. Each circle corresponds to one of the variables in the NchooseK problem or combination of territory and color in the original problem. The boxes represent NchooseK constraints. Within each box, the small squares represent the variable collection—containing the variables pointing to them—and the text indicates the selection set.

IV. Implementation

The NchooseK model is intended to be portable to both gate-model quantum computers and quantum annealers as well as to admit a classical solution. Our current implementation uses a quadratic unconstrained binary optimization (QUBO) formulation as the intermediate representation of an NchooseK program. We first discuss the translation to QUBOs and then describe how these QUBOs are executed on quantum computers.

A QUBO problem can be expressed as the argument minimum of a quadratic pseudo-Boolean function. That is, given

$$f(x) = \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} a_{i,j} x_i x_j \tag{1}$$

with variables $x_i \in \mathbb{Z}_2$ and constants $a_{i,j} \in \mathbb{R}$, we seek the $\mathbf{x}$ values that minimize $f(\mathbf{x})$:

$$\arg\min_{\mathbf{x}} f(\mathbf{x}) \quad . \tag{2}$$

The first step in our NchooseK implementation is to translate an NchooseK problem to QUBO form such that the values returned by Equation 2 satisfy all of the NchooseK constraints.

Consider the NchooseK program from Section II of $nck(\{a,b\},\{1\}) \wedge nck(\{b,c\},\{1\})$. We translate the constraint $nck(\{x_0,x_1\},\{1\})$ to

$$f(x_0,x_1) = 2x_0x_1 - x_0 - x_1 \,, \quad x_0,x_1 \in \mathbb{Z}_2 \tag{3}$$

because this f is minimized when exactly one of x_0 and x_1 has a value of 1. QUBO problems are additive; the solution to a sum of QUBOs is the intersection of the solutions to its constituent QUBOs as long as this intersection is non-empty. Hence, $f(a,b)+f(b,c) = (2ab-a-b)+(2bc-b-c)$ is minimized over the same values of a, b, and c that satisfy $nck(\{a,b\},\{1\}) \wedge nck(\{b,c\},\{1\})$.

We have implemented the NchooseK model in a domain-specific language (DSL) embedded in Python. Figure 3 shows how one could express $nck(\{a,b\},\{1\}) \wedge nck(\{b,c\},\{1\})$ in this DSL.

Authorized licensed use limited to: University of Maryland College Park. Downloaded on July 20,2022 at 17:32:28 UTC from IEEE Xplore. Restrictions apply.

```
import nchoosek
from nchoosek.solve import ocean

env = nchoosek.Environment()
a = env.register_port('a')
b = env.register_port('b')
c = env.register_port('c')
env.nck([a, b], {1})
env.nck([b, c], {1})
print(ocean.solve(env))
```

Fig. 3: The NchooseK program $nck(\{a,b\},\{1\}) \wedge nck(\{b,c\},\{1\})$ expressed as an embedded domain-specific language in Python. The code creates an execution environment, registers variables with that environment, establishes a pair of NchooseK constraints, and solves for the variables using the Ocean library [5]. Typical output is `{'a': False, 'b': True, 'c': False}`.

Our DSL compiler first converts each NchooseK constraint to a quadratic pseudo-Boolean function of the form shown in Equation 1. It does so by expressing each constraint in terms of a Boolean satisfiability problem and uses the Z3 satisfiability modulo theories (SMT) solver [6] to find coefficients for the corresponding quadratic pseudo-Boolean function, such as the one shown in Equation 3, for example. It then sums all of the functions for all of the constraints into a single function to be solved as a QUBO problem.

The compiler can use either a classical or a quantum computer to solve for the variables that minimize the QUBO. The classical solution relies once again on the Z3 SMT solver. The solution on a quantum annealer, whose native input form is essentially a QUBO, uses D-Wave's Ocean library [5]. The solution on a gate-model quantum computer uses the QAOA [2] implementation from IBM's Qiskit library [7] to search for suitable variable assignments. QAOA is a hybrid quantum-classical method that (approximately) solves optimization problems. It utilizes a classical optimizer to determine some of its parameters over tens of jobs on a quantum computer. We use Qiskit's default COBYLA [8] optimizer, but any other optimizer supported by Qiskit could have been used instead.

The approach outlined here is different from that employed by Khetawat et al. [1], which scaled poorly, was not fully automated, and did not have a way to create circuits that combined multiple NchooseK constraints. The trade off relative to our approach is that QAOA requires running multiple circuits, while Khetawat creates a single, complicated circuit solved with a Grover search [9].

V. NchooseK vs. QUBO

NchooseK, as we have implemented it, converts a problem to a QUBO before running it on the various architectures. Quantum annealers, at the lowest level, minimize the energy of a classical, 2-local, Ising-model Hamiltonian function, which is almost identical to solving a QUBO problem. (The former uses variables $x \in \{-1,+1\}$ while the latter uses variables $x \in \{0,1\}$.) Hence, in order to run on a quantum annealer, the conversion to QUBO/Ising would need to be done at some point. For quantum devices following the gate model, extensive prior work has been conducted to convert optimization problems, of which NchooseK is an example, to quantum circuits. Because our implementation already needs to convert an NchooseK program to a QUBO to run it on a quantum annealer, we decided to use this same approach for the gate model as well. The Qiskit library even provides a "quadratic program" interface, which can solve QUBO problems in a number of different ways.

The fact that we convert an NchooseK program to a QUBO before running it on either type of machine raises the following question: *Why not skip NchooseK and create QUBOs directly from the problem?* Our answer is twofold. First, it is often easier to set up a problem with NchooseK than to determine QUBO coefficients directly. Second, it is easier to read and comprehend the semantics of a program written with NchooseK than it is to interpret a QUBO. To illustrate the difference between programming in NchooseK versus programming directly to QUBOs, we compare several NchooseK problems to their equivalent QUBOs in the following discussion.

A. XOR

XOR is a common binary operation. How can $A \oplus B = C$ be represented in both NchooseK and QUBO notation? To start investigating simple problems like this, it is often useful to create a truth table. The truth table for $A \oplus B = C$ is as shown in Table I. A quick inspection of this table uncovers an easy conversion to NchooseK. Each row contains either zero or two *true* values, and each combination of three values *not* appearing as a row in Table I contains either one or three *true* values. The NchooseK constraint that represents this XOR problem is therefore $nck(\{A,B,C\},\{0,2\})$.

A	B	C
0	0	0
1	0	1
0	1	1
1	1	0

TABLE I: The truth table for $A \oplus B = C$ (XOR). As in the rest of this paper, 0 corresponds to *false*, and 1 corresponds to *true*.

The QUBO creation is trickier. Recall that a QUBO with n binary terms takes the form shown in Equation 2. When creating a QUBO, the challenge is posed by deciding the values to assign to each $a_{i,j}$ in Equation 1. Because $0^2 = 0$ and $1^2 = 1$, we can simplify $a_{i,i}x_i^2$ to $a_i x_i$. The XOR truth table tells us that we need to select factors such that all four states have the same, lowest energy. This results in the following system

Authorized licensed use limited to: University of Maryland College Park. Downloaded on July 20,2022 at 17:32:28 UTC from IEEE Xplore. Restrictions apply.

of equations as each constraint must have the same (minimal) value:

$$\begin{array}{cccccc} A & B & C & A\cdot B & A\cdot C & B\cdot C \\ a_A\cdot 0+a_B\cdot 0+a_C\cdot 0+a_{A,B}\cdot 0\cdot 0+a_{A,C}\cdot 0\cdot 0+a_{B,C}\cdot 0\cdot 0 & = \\ a_A\cdot 1+a_B\cdot 0+a_C\cdot 1+a_{A,B}\cdot 1\cdot 0+a_{A,C}\cdot 1\cdot 1+a_{B,C}\cdot 0\cdot 1 & = \\ a_A\cdot 0+a_B\cdot 1+a_C\cdot 1+a_{A,B}\cdot 0\cdot 1+a_{A,C}\cdot 0\cdot 1+a_{B,C}\cdot 1\cdot 1 & = \\ a_A\cdot 1+a_B\cdot 1+a_C\cdot 0+a_{A,B}\cdot 1\cdot 1+a_{A,C}\cdot 1\cdot 0+a_{B,C}\cdot 1\cdot 0 & . \end{array}$$

To match Table I, the preceding equations use A, B, and C as coefficient indices instead of 0, 1, and 2 as in Equation 1.

By multiplying and removing each zero term, these equalities are simplified to

$$0 = a_A + a_C + a_{A,C} = a_B + a_C + a_{B,C} = a_A + a_B + a_{A,B}.$$

which also implies $a_{i,j} = -a_i - a_j$.

Because the rows of Table I must not only be equal to each other when used as $\mathbf{x}$s in Equation 1's $f(\mathbf{x})$ but must also be less than f applied to any other row, we must additionally consider a system of inequalities for all rows not appearing in the table:

$$\begin{array}{cccccc} A & B & C & A\cdot B & A\cdot C & B\cdot C \\ a_A\cdot 0+a_B\cdot 0+a_C\cdot 1+a_{A,B}\cdot 0\cdot 0+a_{A,C}\cdot 0\cdot 1+a_{B,C}\cdot 0\cdot 1 & > 0 \\ a_A\cdot 0+a_B\cdot 1+a_C\cdot 0+a_{A,B}\cdot 0\cdot 1+a_{A,C}\cdot 0\cdot 0+a_{B,C}\cdot 1\cdot 0 & > 0 \\ a_A\cdot 1+a_B\cdot 0+a_C\cdot 0+a_{A,B}\cdot 1\cdot 0+a_{A,C}\cdot 1\cdot 0+a_{B,C}\cdot 0\cdot 0 & > 0 \\ a_A\cdot 1+a_B\cdot 1+a_C\cdot 1+a_{A,B}\cdot 1\cdot 1+a_{A,C}\cdot 1\cdot 1+a_{B,C}\cdot 1\cdot 1 & > 0. \end{array}$$

From the first three inequalities we conclude that

$$\begin{aligned} a_C &> 0 \\ a_B &> 0 \\ a_A &> 0 \quad . \end{aligned}$$

Combining the fourth inequality with the observation made above that $a_{i,j} = -a_i - a_j$ results in the inequality

$$a_A + a_B + a_C + (-a_A - a_B) + (-a_A - a_C) + (-a_B - a_C) > 0$$

and therefore that

$$a_A + a_B + a_C < 0 \quad .$$

But this results in a contradiction: We saw above that a_A, a_B, and a_C must each be greater than zero, but we now see that their sum must be less than zero. This proves that a quadratic pseudo-Boolean function for XOR cannot be constructed with three variables.

To construct a pseudo-Boolean function for XOR we introduce an ancillary variable, D, whose purpose is to increase the degrees of freedom when solving for the Equation 1 coefficients but whose value is ultimately ignored. Essentially, we extend Table I with an additional column to produce Table II. It is difficult in the general case to determine values with which to populate the ancillary columns—more than one may be required for a given truth table—in order to make the system of equalities and the system of inequalities

A	B	C	D
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

TABLE II: The truth table for $A \oplus B = C$ (XOR) with ancillary variable D. The addition of column D makes it possible to express this truth table as a quadratic pseudo-Boolean function.

solvable [10], [11]. The specific values used in Table II lead to the quadratic pseudo-Boolean function

$$\begin{aligned} f(A,B,C,D) = A + B + C + 4D + 2AB - 2AC - 4AD \\ - 2BC - 4BD + 4CD \,, \end{aligned} \quad (4)$$

which is minimized on any row of Table II. As a QUBO, we search for the A, B, C, and D that minimize the function but disregard the value of D that is found.

This example illustrates why finding a QUBO for even a simple problem can be quite difficult, much more so than constructing an NchooseK constraint. It also shows how difficult it can be to determine the function of a QUBO relative to the function of an NchooseK constraint: it is non-obvious that Equation 4 corresponds to an XOR or that one of its variables is not part of the solution being sought.

B. Exact Cover and Map Coloring

Let us next consider the exact-cover and map-coloring problems, which we investigated previously (Section III), from the perspective of a QUBO. Both of these problems are relatively simple to solve using QUBOs. The exact cover problem can be divided into parts based on the elements of E. Because only one subset in the cover is allowed to contain any given elements, we can set up a QUBO for each element $e \in E$ of the form

$$\left(\left(\sum_i v_i \right) - 1 \right)^2 \quad ,$$

where $v_i = 1$ if and only if subset s_i is part of the cover. This ensures that exactly one of the s_i will be included in the cover. (An expression $(x-1)^2$ is minimized when $x = 1$.) Aggregating all of these per-element QUBOs results in a single QUBO that describes the problem. This is a straightforward problem setup but essentially required constructing an NchooseK problem—identifying that exactly one of the sets of v_i must be *true*—before turning it into a series of QUBOs.

The map-coloring problem is also straightforward to express as a QUBO and has been well explored in the context of QUBO problems and quantum annealing [12], [13]. The corresponding top-level QUBO is slightly more complicated than that for the exact-cover problem in that it comprises two different types of QUBOs. One type ensures that each territory has exactly one color, and the other ensures that two territories sharing a border have different colors. Let us denote a variable $T_{i,j}$ for each territory i and color j.

Authorized licensed use limited to: University of Maryland College Park. Downloaded on July 20,2022 at 17:32:28 UTC from IEEE Xplore. Restrictions apply.

The first type of QUBO resembles the expressions in the exact cover problem, where we ensure that each territory can have only one color. Aggregating these results gives

$$\sum_i \left(\left(\sum_j T_{i,j} \right) - 1 \right)^2 .$$

The second type of QUBO is the one involving borders. To ensure that the two adjacent territories do not share a color, we add a QUBO for each pair of territories i and k with a common border. This QUBO is simply $T_{i,j}T_{k,j}$, which has a nonzero value—and is therefore not minimized—if and only if both $T_{i,j}$ and $T_{k,j}$ are *true* (i.e., both have color j). Aggregating all QUBOs of both types results in a QUBO expression of the map-coloring problem.

Once again, to set up this problem as a QUBO we first construct the problem in such a way that it would have been trivial to create an NchooseK problem from it: one color per territory must be *true*; and for each color, zero or one of a pair of adjacent territories must have that color. We noted the two requirements and created a different type of QUBOs to handle each, just as we did with NchooseK constraints in Section III-B.

These examples indicate that expressing a problem with NchooseK tends to be simpler than expressing the same problem as a QUBO. In fact, establishing NchooseK-like constraints is sometimes the first step in constructing a QUBO. An NchooseK problem also tends to be easier to interpret than the corresponding QUBO problem because the need for ancillary variables is hidden from the programmer and because NchooseK constraints directly express the number of variables that must be *true* rather than indirectly encoding such tallies in terms of sums of squared differences, sums of sums, and other formulations.

VI. Results

We ran a variety of exact-cover and map-coloring problems on one of IBM's gate-based machines, ibmq_guadalupe, and one of D-Wave's annealing machines, Advantage 1.1. We observed the correct final results each time we ran any of the problems. In the case of the IBM machine, running the problem includes running multiple circuits 1024 times each, calculating a single result. The D-Wave machine runs a single circuit multiple times, in this case 100. The result which occurs most often is returned, but the energy for each result is calculated. This can be inspected to find multiple correct solutions if such exist, or to check to ensure that the most common result also has the lowest energy. In the map-coloring problems, the D-Wave machine found multiple correct results while QAOA on the IBM machine terminates after a single result is found.

Some results on the gate-based machine are shown in Table III. QAOA alternates submitting a job to the quantum computer and feeding the measured output to a classical optimizer, which prepares the next job to submit to the quantum computer. The process repeats until a convergence criterion is met. We found no significant trend in the relationship between the complexity of the problem and the number of individual jobs needed to be run on IBM's machine until convergence was reached, not even when the machine was using all 16 of its qubits.

Type	Vars	Cons	Qubits	Jobs	Depth	CNOTs
Exact	6	7	6	31	70	61
Exact	6	8	6	28	52	48
Exact	8	8	8	33	130	171
Exact	10	10	14	30	122	256
Map	8	6	8	31	90	112
Map	12	16	15	36	130	281
Map	16	20	16	33	168	403
Map	16	24	16	31	173	388

TABLE III: Problem setups and results on IBM's ibmq_guadalupe 16-qubit gate-model machine. The table indicates the problem type (exact cover/map coloring), the number of variables of interest, the number of NchooseK constraints, the number of qubits used for them, the number of jobs run as part of the QAOA, the depth of the circuits within the QAOA, and the number of CNOT gates per circuit. Each job comprised 1024 shots (quantum circuit executions).

The data plotted in Figure 4 shows how the circuit depth of the circuits and the number of CNOT gates used both rise as the number of NchooseK variables increases. The circuit

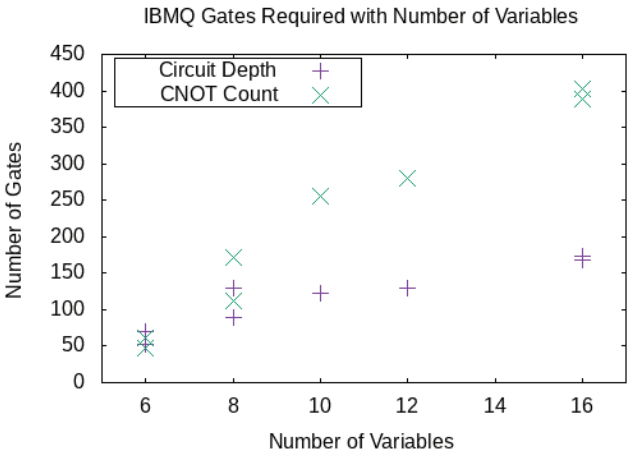

Fig. 4: Graph of the circuit depth (+) and the number of CNOT gates (×) as a function of the number of NchooseK variables in the program.

depth—the number of time steps needed for the circuit to complete—is an important metric because it indicates how long the qubits will need to remain active, which correlates with both execution time and susceptibility to errors. The CNOT count is important because (two-qubit) CNOT gates are an order of magnitude more susceptible to errors than single-qubit gates. For these results, we considered every job except the last in a QAOA iteration sequence. All jobs but the last one are run to find the circuit parameters (rotational angles) that minimize the corresponding QUBO, and they make up

Authorized licensed use limited to: University of Maryland College Park. Downloaded on July 20,2022 at 17:32:28 UTC from IEEE Xplore. Restrictions apply.

the bulk of the work. The final job represents only a post-processing step.

The increases in depth and CNOT count indicate an increase in complexity. Not only do qubits likely need to interact via CNOTs if they share any constraints, but they also likely need to be swapped in order to affect each other due to the hardware interconnect topology. These swaps are constructed from CNOT gates, also contributing to the increase in CNOT count. Any increase in gate count has the potential to increase the circuit depth; CNOTs are especially likely to do so, as gates which affect multiple qubits can force some qubits to wait for others to finish other operations before interacting.

Results for the quantum annealer are shown in Table IV. Both the IBM and D-Wave quantum computers provide only

Type	Vars	Cons	Correct (%)	Qubits
Exact	6	7	57	7
Exact	6	8	71	7
Exact	8	8	53	12
Exact	10	10	44	16
Map	8	6	100	8
Map	12	16	91	17
Map	16	20	91	22
Map	16	24	70	25

TABLE IV: Problem set-ups and results on a D-Wave Advantage quantum-annealing machine. The tables indicates the problem type (exact cover/map coloring), the number of variables of interest, the number of NchooseK constraints, the percentage of runs which returned the correct results, and the number of qubits used. Each problem was run 100 times. In all cases, the statistical mode corresponds to the correct result, even when the overall percentage of runs returning the correct result was relatively small.

sparse qubit connectivity. Entangling qubits that are non-adjacent in the hardware topology requires extra time on a gate-based quantum computer such as IBM's, which is achieved via a sequences of swap operations. However, it requires extra space on an annealing-based quantum computer such as D-Wave's, which comes in the form of "chaining" multiple physical qubits into a logical qubit to increase effective connectivity. This effect is visible in Table IV as the required number of qubits increases not only with the number of NchooseK variables, as was the case with the gate-based system, but also with the number of constraints, as seen in particular in the final two rows of the table.

The map-coloring problems used in the final two rows of the table both involve four territories and four colors. In the first one, with 20 constraints, each territory shares a border with two others, and they could be arranged in a ring. The other is similar, but with one additional border added between two of the territories. This increased connectivity of the territories corresponds to increased connectivity needed within the annealer, leading in turn to more qubits being used to represent the problem.

Both the QAOA algorithm and quantum-annealing hardware typically run each problem many times to gain statistical validity. (Remember, quantum computation is fundamentally stochastic.) While Qiskit's QAOA implementation returns the single best solution, the Ocean library returns a histogram of solutions so that one may select a solution (or, if desired, multiple solutions) to consider. If the problem is correctly formulated, the minimal-energy solution should (within statistical error) be the correct one, but this may not be the most frequently occurring solution. When we inspect the success probability of running our eight problems (Figure 5), an interesting trend appears. Taken individually, the accuracy of the different problems falls when the number of qubits used increases, but the first-order effect is the problem type. The best exact-cover problem observes a worse success probability—by a full percentage point—than the worst map-coloring problem, despite the fact that the exact-cover problems use significantly fewer qubits than the map-coloring ones. One possible explanation of this is that the exact cover problems had, in these examples, exactly one right answer. The map coloring problems, on the other hand, always have multiple solutions simply by virtue of color permutations, to say nothing of different correct arrangements of the colors.

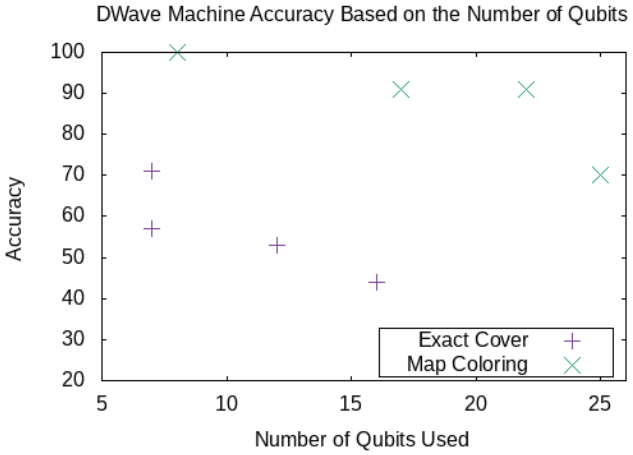

Fig. 5: The trend in accuracy on D-Wave systems with respect to the number of qubits used. Overall, the exact-cover problems (+) observe a lower success probability than the map-coloring problems (×).

VII. Future Work

Work is currently being done to expand the capabilities of the NchooseK model to enable it to tackle a greater variety of optimization problems. On the evaluation side, we are also planning experiments to investigate the results of NchooseK problems more fully, notably measuring the time spent on both the quantum computation proper and the time spent in classical problem preparation and, for QAOA, optimization.

We are also interested in finding new ways to prepare NchooseK problems for gate-based machines. One approach is to prepare custom mixers for QAOA [14], rather than using Qiskit's defaults.

Authorized licensed use limited to: University of Maryland College Park. Downloaded on July 20,2022 at 17:32:28 UTC from IEEE Xplore. Restrictions apply.

Once we have the framework of NchooseK more solidly in place, we may look into expanding into more general ways of programming, such as a more general ILP interface.

VIII. Conclusions

NchooseK is a constraint-based programming model designed to be sufficiently powerful to express a variety of problems while working at a level of abstraction that enables the same program to run on classical computers, quantum annealers, and circuit-based quantum computers. Although not a typical programming model, NchooseK has a classical semantics in that programmers work with bits ($\mathbb{Z}_2$) rather than qubits ($\mathbb{C}^2$) and do not have to reason about quantum effects such as superpositioning and entanglement, e.g., via unitary matrix transformations. Our intention is that NchooseK's simple semantics will help non-experts exploit the power of quantum computing.

NchooseK as we have implemented it converts each problem to a QUBO, which is subsequently converted to a form more suitable for the target architecture. QUBOs make a suitable intermediate representation because they are supported natively by quantum annealers, can be converted to QAOA circuits for circuit-based machines, and can be solved using a variety of classical solver types, such as an SMT solver. While programs are converted internally to QUBOs, it is often easier for a programmer to write NchooseK programs than it is to calculate QUBO coefficients, and NchooseK problems are more human-readable than QUBOs.

Acknowledgements

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, and MPS-2120757.

References

[1] H. Khetawat, A. Atrey, G. Li, and F. Mueller, "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.

[2] 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.

[3] K. Appel and W. Haken, "Every planar map is four colorable. Part I: Discharging," *Illinois Journal of Mathematics*, vol. 21, no. 3, pp. 429–490, Sep. 1977.

[4] K. Appel, W. Haken, and J. Koch, "Every planar map is four colorable. Part II: Reducibility," *Illinois Journal of Mathematics*, vol. 21, no. 3, pp. 491–567, Sep. 1977.

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

[6] 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.

[7] R. Wille, R. V. Meter, and Y. Naveh, "IBM's Qiskit tool chain: Working with and developing for real quantum computers," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Florence, Italy, Mar. 25–29, 2019, pp. 1234–1240.

[8] M. J. D. Powell, *A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation*. Dordrecht, Netherlands: Springer, 1994, pp. 51–67.

[9] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. New York, New York, USA: ACM, 1996, pp. 212–219.

[10] Z. Bian, F. Chudak, R. Israel, B. Lackey, W. G. Macready, and A. Roy, "Discrete optimization using quantum annealing on sparse Ising models," *Frontiers in Physics*, vol. 2, Sep. 18, 2014.

[11] S. Pakin, "A simple heuristic for expressing a truth table as a quadratic pseudo-boolean function," in *IEEE International Conference on Quantum Computing and Engineering (QCE 21)*, Oct. 17–22, 2021.

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

[13] E. D. Dahl, "Programming with D-Wave: Map coloring problem," D-Wave Systems, Burnaby, British Columbia, Canada, White Paper, 2013. [Online]. Available: https://www.dwavesys.com/media/htfgw5bk/map-coloring-wp2.pdf

[14] S. Hadfield, Z. Wang, B. O'Gorman, E. Rieffel, D. Venturelli, and R. Biswas, "From the quantum approximate optimization algorithm to a quantum alternating operator ansatz," *Algorithms*, vol. 12, no. 2, p. 34, Feb 2019. [Online]. Available: http://dx.doi.org/10.3390/a12020034

Authorized licensed use limited to: University of Maryland College Park. Downloaded on July 20,2022 at 17:32:28 UTC from IEEE Xplore. Restrictions apply.