Mapping

Introduction

In this lesson, we will focus on the first and often most challenging step in defining a quantum program: mapping a problem to be run on a quantum computer. This step covers how a user starts with a computational problem and translates it to something that can be solved on a quantum computer.

In Lessons 2 and 3 of this course, we mentioned that the mapping stage is the first of four total steps in the Qiskit patterns framework. From these

lessons, you may remember that the goal of mapping is to translate or rewrite a computational problem into a cost function or expectation value that we can evaluate using a quantum computer.

In Lesson 3 we discussed one concrete example with Max-Cut, a computationally hard, but very common, problem in combinatorial optimization. In that example, we went through several steps to translate the initial graph problem into one that could be solved on a quantum computer. We transformed the problem of finding the maximum number of cuts in the graph into a cost function, rewrote that cost function as a Hamiltonian, and then prepared a trial quantum state whose ground state corresponded to the maximum cut. Finally, we constructed a quantum circuit representing the trial quantum state of interest, and then added the specific gates to allow the state to evolve over time. This sequence of steps was all part of mapping. While the exact steps were unique to the Max-Cut problem, the same general procedure can applied to many other applications, such as quantum chemistry and quantum simulations.

Mapping can be difficult. There isn't a one-size-fits all strategy for every single problem, so it can be intimidating. In this lesson, we'll look at some general considerations for mapping, and then dive into some representative example problems to demonstrate the various ways to map a problem to a quantum computer.

General considerations

While the exact strategy one uses to map a problem to a quantum computer depends on the problem at hand, they generally accomplish a couple main things.

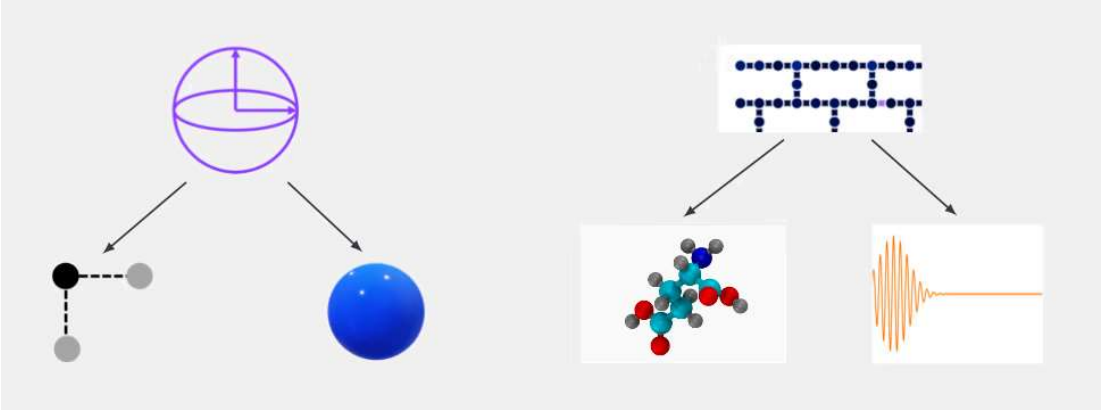
First, you might need to simplify the problem to make it feasible. This is not unique to quantum – all scientific disciplines use simplified models to study phenomena they're interested in, while ignoring irrelevant details. In physics, there's a famous expression that takes this principle to an extreme: "assume a spherical cow." It is often too hard to describe a system exactly as it appears, but we can instead make reasonable simplifications that can still lead to helpful solutions. Some examples of how we might do this in quantum computing are to limit the size or the circuit depth by choosing a hardware-efficient ansatz, truncating complex time evolutions, or neglecting

Hamiltonian terms that contribute little to the final energy of a quantum state.

Second, mapping involves writing the problem in such a way that a quantum computer can make sense of it. Often that involves asking these three questions:

1. What will our qubits represent in our model?
2. Is our problem continuous? Since quantum computers are digital, if the problem is continuous, we will need to find a way to discretize it.
3. Lastly, does the topology of the problem align with the topology of the hardware? If not, we might need to implement some tricks to make it work.

Let's address the first question, which is at the heart of many of the difficulties with understanding mapping: What can a qubit represent?



Qubits can be used to represent many things. The first, and perhaps simplest, is a node on a graph. Graphs are used to show connectivity in many different types of mathematical problems, and the nodes are a fundamental element representing a point or entity within a network. Depending on what the whole network represents, that could be a city, a person, or a ferromagnet in a lattice.

Qubits can also be used to represent bosons and fermions, although I will caution here that one single qubit does not exactly equal to one boson or one fermion – it's a bit trickier than that, as we will discuss further in the lesson.

Now we're getting examples that are a bit more complicated. For these models, it doesn't make sense anymore to speak in terms of single qubits,

instead we need groups of them to make up something physical. For example, a group of qubits, here represented on a heavy hexagonal topology, can be used to represent geometrical locations of amino acids; polymer chains. Another example is the simulation of hadron scattering in high energy physics models, which can be done by simulating the time evolution of a hadronic wavepacket. In this case, a register of qubits can be used to encode different states of a quantum field; the vacuum state of that field, or a wavepacket that propagates on top of that vacuum.

But at this point we have spoken abstractly enough about the challenge before us. Let's look into these examples in detail.

Mapping Examples

Max-Cut

Let's begin with our first example. One of the most straightforward mapping problems is the one we have already covered in some depth: the Max-Cut example. In that problem the mapping was pretty easy for us because one qubit was equivalent to one node on our graph.

Recall that, to map the Max-Cut problem, we expressed the cost function as a Hamiltonian using the QUBO formulation. A Hamiltonian cost function is a function that encodes the optimal solution of the problem in the ground state of the Hamiltonian. To construct the cost Hamiltonian, we used the `SparsePauliOp` class in Qiskit to specify the connectivity of our graph, and the mapping stage to the quantum operators was done. And the quantum circuit was simply the QAOA ansatz. And if this isn't ringing a bell, please go back and check out Lesson 3 where we walk through this all in much more detail.

In that lesson, even in the 100-qubit, utility-scale example, the graph connectivity already matched the topology of our superconducting hardware. So we didn't need to concern ourselves with how to deal with different topologies. But that's not always the case. If we had a more complicated or densely connected graph than the examples we highlighted so far, we would need to implement a series of SWAP gates to modify the effective connectivity of the hardware. This is handled at the second step of

Qiskit patterns, transpilation, but should be kept in mind even in the mapping step.

Protein folding

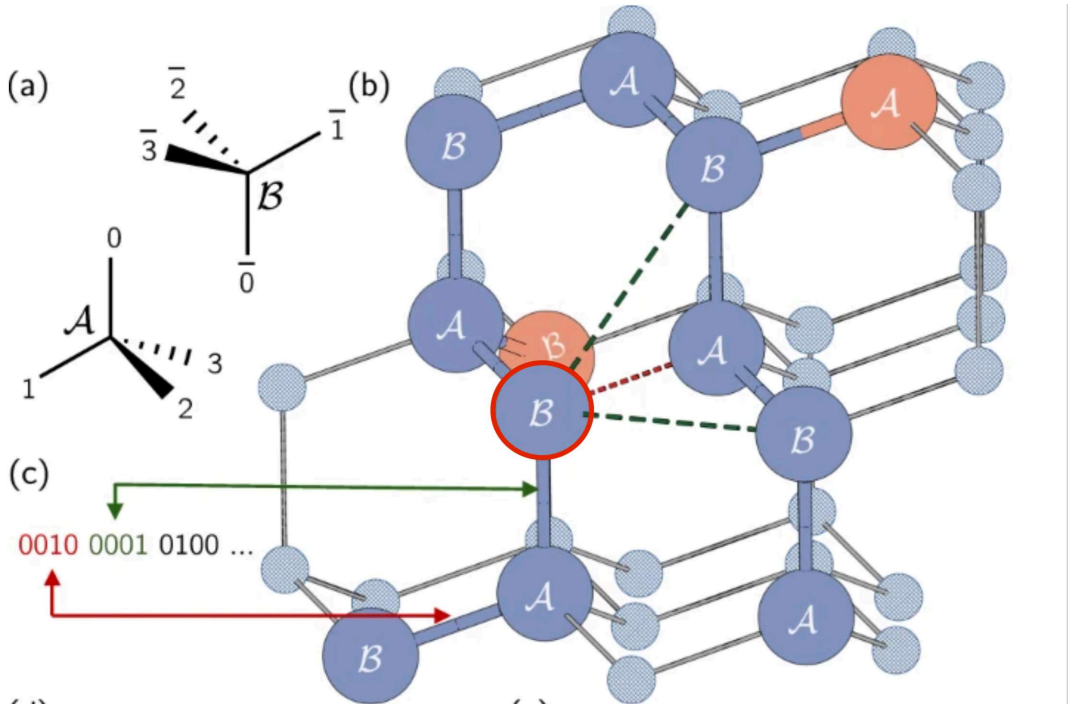
Next, let's explore an example modeled in the paper called "Resource-efficient quantum algorithm for protein folding," written by IBM and collaborators at University of New South Wales.

A little biochemistry background: Proteins are macromolecules composed of long chains of amino acids. These chains fold into complex structures that perform a wide variety of biological functions. Determining a protein's structure in three-dimensional space, and understanding the relationships between structure and function, are among the most challenging problems in biochemistry today. Proteins fold into useful structures due to interactions between amino acids. As a structure twists and folds, amino acids that are far from each other along the chain may end up right next to each other and may interact strongly.

To model this on a quantum computer, we need a Hamiltonian describing all these interactions between amino acids. Then, we can predict the final structure by finding the state that will minimize the energy of our Hamiltonian. Here, we will focus on how amino acid chains can be modeled on a quantum computer and how we can obtain inter-amino-acid distances for calculation of interaction energies. With this, we will have gathered all the necessary contributions to the Hamiltonian that is necessary to simulate it on a quantum computer.

In real proteins, amino acids can occupy a continuum of possible locations. However, we are going to use a simplification and restrict them using a lattice model, where each amino acid occupies a point on a grid. Here, the authors used a tetrahedral lattice. Quick note: here, we are encoding the direction of the edges, not the nodes themselves like in the Max-Cut problem. Each qubit represents a possible single-step path along the tetrahedral grid. Note that adjacent sites have been labeled as A or B because of their different orientations in the lattice.

The protein chain is represented as a series of turns or directions on this lattice. Each turn between amino acids can be in one of four directions, corresponding to edges of the tetrahedron. These four possible turns are encoded using four qubits into the states `0001`, `0010`, `0100`, or `1000`.



Let's look at an example in the figure above. Let's place our first amino acid on the point labelled "B" circled in red in our tetrahedral lattice. The direction to the first amino acid to the second is arbitrary because the system can always be rotated to make that edge point in any direction we like. So, we can place our second amino acid on the point below the first labelled "A". It's not as easy to see, but the path from the second to the third is also arbitrary. All three choices would result in our having two edges with an angle of approximately 109.5 degrees between them. Picking this second edge simply determines the orientation of our protein in space. So, without loss of generality, we can choose the first two turns to just be `0001` and `0010`.

With these simplifications, the configuration of the amino acid chain is given by this expression:

$$(0001)(0010)(q_9 q_{10} q_{11} q_{12}) \cdots (q_{4N-3} q_{4N-2} q_{4N-1} q_{4N})$$

So far, we have mapped the tetrahedron edges to qubits, and our quantum circuit will be an ansatz. Now we need to consider how to encode the energy of the problem into a Hamiltonian, so that its ground state gives us the optimal folding pattern.

For any given configuration, there will be an associated energy due to interactions between the amino acids. These interactions are strongest

when the two amino acids are close to each other. Obviously, amino acids that are adjacent in the backbone of the chain will always interact with each other. But because the protein can twist and fold, other amino acid pairs can also interact. Amino acids 10 and 20 might end up being on adjacent sites after the protein has folded, for example. So, we need a formula to describe the distance d between amino acids i and j using the information encoded on the configuration qubits. That way we can use their separation distance to determine how strongly they are interacting.

First, let's introduce a function $f_a(i)$ that indicates whether or not an edge a is used for the turn at amino acid i. Here a could take on any of the four possible directions. Based on the configuration we started with above, we can write these functions:

$$
\begin{aligned}
f_0(i) &= q_{4i-3} \\
f_1(i) &= q_{4i-2} \\
f_2(i) &= q_{4i-1} \\
f_3(i) &= q_{4i}
\end{aligned}
$$

Then we can define a difference in the number of a-labeled turns on A and B lattices from index i to index j as Δn:

$$
\Delta n_a(i,j) = \sum_{k=i}^{j} (-1)^k f_a(k)
$$

Why would we do this? Well, it turns out that a turn of a from lattice site A to B is exactly cancelled out by a turn of a from lattice site B to A. So to know the distance the amino acid on site i is from the one on site j, we just need to find the difference between steps made along a edges from A sites and B sites. Since A and B sites necessarily alternate along the protein backbone, this is captured in the $(-1)^k$. This same argument applies to all four edge types. So, the total distance between amino acids in tetrahedral lattice steps can be calculated by this expression:

$$
\begin{aligned}
d(i,j) &= \sum_a ||\Delta n_a(i,j)||^2 \\
&= (\Delta n_0(i,j))^2 + (\Delta n_1(i,j))^2 + (\Delta n_2(i,j))^2 + (\Delta n_3(i,j))^2
\end{aligned}
$$

But how do we get the Hamiltonian from this long equation for the total distance between the amino acids? First, we can convert from the distance

in lattice steps to Euclidean space with some simple geometry:

$$
\begin{aligned}
d(i,j) = 0 &\rightarrow r_{ij} = 0 \\
d(i,j) = 1 &\rightarrow r_{ij} = 1 \\
d(i,j) = 2 &\rightarrow r_{ij} = 2\sqrt{\frac{2}{3}} \approx 1.63 \\
d(i,j) = 3 &\rightarrow r_{ij} = \sqrt{\frac{11}{3}} \approx 1.91 \\
d(i,j) = 4 &\rightarrow r_{ij} = \frac{4}{\sqrt{3}} \approx 2.31 \\
d(i,j) = 5 &\rightarrow r_{ij} = \sqrt{\frac{19}{3}} \approx 2.52
\end{aligned}
$$

Then, these distances will be used in calculating the energy of protein configuration. Depending on our purposes, we might dictate a cutoff distance below which you consider the pair to be interacting, or you might do something more complicated.

It may not be obvious, but we're actually done with the mapping stage by doing this. the states of the qubits indicate the "turn" of the protein at each lattice site, and the collection of turns determines the distance between any pair of amino acids. Pairs of various species of amino acids have different interactions, some attractive, some repulsive. If you are using the configuration and distances to merely determine if known amino acid interactions are "on" or "off", the strengths of these have already been worked out and can simply be looked up in a table like this:

(c)	H_6	P_7	F_8	H_9	L_{10}
D_1	-2.32		-3.48		-3.4
R_2		-1.7		-2.16	
V_3			-6.29		-6.48
Y_4				-3.52	
V_5					-6.48

In summary, in this example, the qubits are used to mark steps in a path along a lattice, which together, form a chain of amino acids. By simulating how they bend and fold we can hopefully find better outcomes in medical research. We skipped how to calculate a few terms of this Hamiltonian because they were hyper-specific to this problem, while defining directions on a lattice can be applied more generally. Now once you have a general Hamiltonian, you are always going to want to translate that into Pauli operators, which are native to the quantum computer. That's what we'll discuss next.

Jordan-Wigner Transformation

Now let's explore how to translate a system of subatomic particles into Pauli operators.

Subatomic particles are broken down into two categories – bosons and fermions. Bosons, like photons or the Higgs, obey a certain set of statistical rules. Fermions, like electrons or neutrinos, obey another. The key difference between them is that bosons are allowed to occupy the same state – there's no limit to how many bosons can be in the ground state or any excited state. Fermions, on the other hand, are selfish – they demand that every single particle has its own quantum state.

Bosons also have integer spins while fermions have half-integer spin, like the spin-1/2 electron, and more exotic spin-3/2 particles. To describe a system of particles, we need a description of their energy. Let's focus on fermions. We can start with a Hamiltonian that is written in terms of what we call *c operators*. These are basically mathematical objects that correspond to creating or annihilating a fermion from a state in the system. These are often written as $f_i^\dagger$ and f_j, where $f_i^\dagger$ is the operator that creates a fermion in state i and f_j is the operator that destroys a fermion in state j.

But remember, quantum computers usually operate in a qubit basis with specific rules for representing fermionic systems; they do not inherently work in the language of fermionic operators. To bridge this gap, we need to map this fermionic notation onto Pauli operators, which naturally correspond to quantum gates.

There are several such transformations for fermions. A common choice is the Jordan-Wigner transformation. The Bravyi-Kitaev and parity mappings are more recent fermionic encoding. Bosonic operators can be transformed by using the Holstein-Primakoff transformation or directly mapping the Fock states to a sub-basis of the available bosonic modes, among other options. Other encodings are also actively being researched. For now, we are going to just focus on the Jordan-Wigner transform.

The Jordan-Wigner transformation involves mapping a single fermion to many qubits. Why can't we just assign one qubit to represent each electron? This has to do with the distinguishability of identical fermions. Qubits are distinguishable and electrons are not. For instance, we can easily label and identify individual qubits on any device. But the indistinguishability of electrons means we can't label them at all. Thus, we need to actually label the operators according to occupied orbitals, like 1s, 2p, 2p, etc., instead of specific electrons. So, each qubit plays approximately the role of an orbital in the molecule which is either occupied or unoccupied. But how we do that is a bit more complicated. The Jordan-Wigner mapping keeps track of anti-symmetry and ensures the correct statistics of the overall fermionic system. The Jordan-Wigner mapping expresses fermionic operators in terms of Pauli operators using these relations:

$$f_j^\dagger = \Big(\prod_{k<j}(-Z_k)\Big)\Big(\frac{X_j + iY_j}{2}\Big)$$

$$f_j = \Big(\prod_{k<j}(-Z_k)\Big)\Big(\frac{X_j - iY_j}{2}\Big)$$

The Jordan-Wigner mapping is conceptually simple because of the one-to-one correspondence between orbitals and qubits. There are other mappings that accomplish a similar goal, including the parity mapping. The calculation of the parity of a state requires consideration of multiple qubits. In the parity mapping (and some others) the interpretation of one qubit corresponding to one orbital does not hold. Now let's walk through a simple example. Suppose we want to calculate the single qubit interaction $f_0^\dagger f_0$. We start by plugging in our definitions for the creation and annihilation operators.

$$\begin{aligned} f_0^\dagger f_0 &= \left(\prod_{k<0}(-Z_k)^2\right)\left(\frac{X_0 + iY_0}{2}\right)\left(\frac{X_0 - iY_0}{2}\right) \\ &= \frac{1}{4}(X_0^2 + iY_0X_0 - X_0Y_0 + Y_0^2) \\ &= \frac{1}{4}(2I - i[X_0, Y_0]) \end{aligned}$$

The commutator $[X_0, Y_0] = 2iZ_0$. So, the final expression becomes:

$$f_0^\dagger f_0 = \frac{1}{2}(I + Z_0)$$

So, we have successfully rewritten a fermionic expression in terms of Pauli operators that our quantum computer will be able to understand. Let's quickly discuss how we would implement the Jordan-Wigner mapping in Qiskit. It's important to understand how these types of transforms work, but it would be impractical to manually work them out every time – especially for large size systems. Luckily, Qiskit makes this easy for us with the `SparsePauliOp` function.

At a high level, the steps are:

1. Use SparsePauliOp's `from_list` function to create an identity operator corresponding to the size of the parameter space to be mapped.
2. Following the definition of the creation and destruction operators shown previously, use the SparsePauliOp `from_list` function to define X, Y, Z Pauli operators. This will map the fermionic creation and annihilation operators to qubit spin operators, encoding fermionic occupation number into the computation basis of qubits.
3. Generate the desired Hamiltonian in the Pauli basis by applying the above operators to the orbitals of interest. This usually corresponds to the creation of an identity matrix which represents the core (non-interacting) orbitals and then applying the creation and annihilation

operators to the active space, with weights that correspond to specifics of the problem at hand.

Now that we fully understand Jordan-Wigner mapping scheme, let's see a more complicated example. You might remember the paper titled, "Scalable Circuits for Preparing Ground States on Digital Quantum Computers: The Schwinger Model Vacuum on 100 Qubits" from the previous lesson. We won't go through the paper in detail again—we'll just focus in on the Jordan-Wigner mapping, which is used to express the Hamiltonian of lattice sites with L sites for the Schwinger model in the absence of an electric field.

Here, it's a lot harder to specifically pinpoint what one qubit represents in this model because it's only the collection of qubits together that make something physical, in this case, a wave packet. Instead, you can roughly think of the qubits as discretized pieces of space. Here, L is the lattice volume in which each element (unit cell) comprises two qubits. The fermionic operators that we saw in the previous slide describe the amplitude of the wavefunction on a particular site. So, our Hamiltonian contains these fermionic creation and annihilation operators. So, we use the Jordan-Wigner transformation to map these operators into the Pauli operators.

$$\begin{aligned}\mathcal{H} &= \mathcal{H}_m + \mathcal{H}_{kin} + \mathcal{H}_{el} \\ &= \frac{m}{2}\sum_{j=0}^{2L-1}[(-1)^j Z_j + I] + \frac{1}{2}\sum_{j=0}^{2L-2}(\sigma_j^+\sigma_{j+1}^- + h.c.) + \frac{g^2}{2}\sum_{j=0}^{2L-2}\Big(\sum_{k\leq j} Q\end{aligned}$$

where σ_+ is the Pauli operator $X + iY$ and σ_- is $X - iY$. Once we have a Hamiltonian written in this format, the hard part of the mapping stage is over, and it can now be easily written to a circuit in Pauli operators.

Conclusion

Sign in to track your progress

We have discussed four examples of how specific mapping techniques have been used lately in the quantum computing field, starting from simplest and