

CONSTRUCTION OF ENERGY FUNCTIONS FOR LATTICE HETEROPOLYMER MODELS: EFFICIENT ENCODINGS FOR CONSTRAINT SATISFACTION PROGRAMMING AND QUANTUM ANNEALING

RYAN BABBUSH,¹ ALEJANDRO PERDOMO-ORTIZ,^{1,2} BRYAN
O'GORMAN,¹ WILLIAM MACREADY,³ and ALAN ASPURU-GUZI¹

¹*Department of Chemistry and Chemical Biology, Harvard University,
12 Oxford Street, Cambridge, MA 02138, USA*

²*NASA Ames Quantum Laboratory, Ames Research Center,
Moffett Field, CA 94035, USA*

³*D-Wave Systems, Inc., 100-4401 Still Creek Drive, Burnaby,
British Columbia V5C 6G9, Canada*

- I. Introduction
 - A. Motivation and Background
 - B. Overview of Mapping Procedure
- II. The ‘‘Turn’’ Encoding of Self-Avoiding Walks
 - A. Embedding Physical Structure
 - B. ‘‘Turn Ancilla’’ Construction of $E(q)$
 - 1. Construction of $E_{\text{back}}(q)$
 - 2. Construction of $E_{\text{overlap}}(q)$ with Ancilla Variables
 - 3. Construction of $E_{\text{pair}}(q)$ with Ancilla Variables
 - C. ‘‘Turn Circuit’’ Construction of $E(q)$
 - 1. Sum Strings
 - 2. Construction of $E_{\text{overlap}}(q)$ with Circuit
 - 3. Construction of $E_{\text{pair}}(q)$ with Circuit
- III. The ‘‘Diamond’’ Encoding of SAWs
 - A. Embedding Physical Structure
 - B. Natively 2-Local $E(q)$
 - 1. Construction of $E_{\text{one}}(q)$
 - 2. Construction of $E_{\text{connect}}(q)$

- 3. Construction of $E_{\text{overlap}}(q)$
- 4. Construction of $E_{\text{pair}}(q)$
- IV. Pseudo-Boolean Function to W-SAT
 - A. MAX-SAT and W-SAT
 - B. Constructing WCNF Clauses
 - C. Solving SAT Problems
- V. W-SAT to Integer-Linear Programming
 - A. Mapping to ILP
 - B. Solving ILP Problems
- VI. Locality Reductions
- VII. Quantum Realization
 - A. Previous Experimental Implementation
 - B. Six-Unit Miyazawa-Jernigan Protein
 - 1. $E_{\text{back}}(q)$ for Six-Unit SAW on 2D Lattice
 - 2. $E_{\text{overlap}}(q)$ for Six-Unit SAW on 2D Lattice
 - 3. $E_{\text{pair}}(q)$ for MJ Model PSVKMA
 - 4. Setting λ Penalty Values
 - 5. Reduction to 2-Local
 - 6. QUBO Matrix and Solutions
- VIII. Conclusions
- References

I. INTRODUCTION

A. Motivation and Background

Optimization problems associated with the interaction of linked particles are ubiquitous in the physical sciences. For example, insights into a problem of biological relevance, such as the protein folding problem, can be obtained from trying to solve the optimization problem of finding the lowest energy configuration of a given sequence of amino acids in space [1–8]. Among other examples of biologically relevant polymers, DNA and RNA chains also fold into complicated structures that can be challenging to predict.

The number of possible configurations (in fact, the number of local minima) for a protein with N amino acids is exponential in N [9]. Even the simplest model for lattice folding [10] was proved to be an NP-hard problem [11,12]. This implies that the scaling of the worst-case scenario for arbitrary protein sequences is exponential with the size of the system. This scaling imposes limitations on the exhaustive search in lattice models for proteins with as few as 36 amino acids in even the most coarse-grained protein models [13].

An alternative route to exhaustive search or the development of new heuristics is to map these problems into the form of other, more general problems that have been extensively studied for decades. For instance, the NP-complete problem

known as MAX-SAT has central importance to practical technologies such as artificial intelligence, circuit design, automated theorem proving, cryptography, and electronic verification [14–16]. The study of this particular problem is central to computer science. There are several journals, conferences, and competitions every year dedicated entirely to solving SAT problems [17]. Another widely studied constraint satisfaction problem is linear programming that has many applications including logistics scheduling, operations research, company management, and economic planning [18]. Some applications of linear programming, that is, multicommodity flow problems, are considered important enough that entire fields of research exist to develop specialized algorithms for their solution [19].

Once cast as one of these canonical constraint satisfaction problems, one can leverage decades of progress in these fields to solve lattice heteropolymer problems. Though it has received relatively little attention until recently, the idea that constraint programming can help solve problems of this type has at least appeared in protein folding and computer science literature [20]. Other relevant papers include Refs. [21–25].

Another intriguing option is to study these problems using a computer that takes advantage of quantum mechanical effects to drastically reduce the time required to solve certain problems. For combinatorial optimization problems, perhaps the most intuitive quantum computing paradigm is quantum annealing [26–32], also known as adiabatic quantum computation [27,33,34]. In quantum annealing, the presence of quantum fluctuations (tunneling) allows the system to efficiently traverse potential energy barriers that have a tendency to trap classical optimization algorithms.

Motivated by the experimental realization of studying biologically interesting optimization problems with quantum computation, in this chapter we present a general construction of the free energy function for the two-dimensional lattice heteropolymer model widely used to study the dynamics of proteins. While the authors have already demonstrated some of these techniques in Ref. [35], the encoding strategies discussed here are more general and also more efficient than what we have explained previously. The reduction in resources achieved with these methods allowed for the first experimental implementation of lattice folding on a quantum device [36], where we employed up to 81 superconducting qubits to solve a six-amino acid problem in the Miyazawa–Jernigan (MJ) model [37].

The goal of this chapter is to explain the mapping used in Ref. [36], to discuss the strengths and weaknesses of this mapping with respect to other strategies, and to demonstrate how to map the lattice heteropolymer problem into forms that can be solved by using different types of technology and algorithms. While the focus of this chapter will be on lattice protein folding, the methods introduced here have

very general relevance to discrete and combinatorial optimization problems in science. Whether one decides to use a classical or a quantum (annealing) device, the mappings and techniques presented here emphasize the importance of three key considerations: energy function locality, coupler/coefficient resolution, and efficiency of encoding.

In this context, the “locality” of an expression refers to the order of the largest many-body expansion term. For instance, QUBO (quadratic unconstrained binary optimization) problems, which are a binary version of the Ising model, are said to be “2-local” because QUBO expressions never contain terms with more than two variables. This is a relevant consideration because an expression that is 3-local cannot be programmed into a quantum device with only pairwise couplings. A similar consideration applies to classical solvers. Coefficient resolution refers to the ability of a quantum device or classical solver to program coupler values to the degree of precision required for the problem. Finally, the efficiency of the encoding refers to the number of bits required to encode the problem. A sketch of how one might weigh these considerations to determine an encoding is shown in Fig. 1.

B. Overview of Mapping Procedure

The embedding strategies presented here apply to many discrete optimization problems. Mapping these problems to a constraint programming problem is a three-step process. In this section, we provide a brief description of the process and expand upon each step as it applies to lattice folding in later sections.

1. *Encode Solution Space in Computational Basis.* Define a one-to-one mapping between possible valid assignments of the problem and a bit string encoding this information. Let us denote the bit string by $\mathbf{q} \equiv q_1 q_2 \cdots q_n$. The way information is encoded at this point can drastically alter the nature of the following three steps, so one must take care to choose a mapping that will ultimately make the best use of resources; in many cases, the most compact mapping will have a high-order energy function or require many ancillary bits. Regardless of how information is encoded, the bit string must uniquely enumerate each element of the low-energy solution space.
2. *Constrain Energy Landscape with Pseudo-Boolean Expression.* Construct a pseudo-Boolean energy function $E(\mathbf{q}) = E(q_1, q_2, \dots, q_n)$ that takes \mathbf{q} as input and correctly reproduces the relative energies in the low-energy subspace of the original problem so that the optimal solution to $E(\mathbf{q})$ encodes the solution to the original problem. The construction of this function is not trivial and will depend largely on how information is encoded in \mathbf{q} . At this point, it may be necessary to increase the dimensionality of the solution space by adding ancillary bits. In a previous contribution, we provided a specific

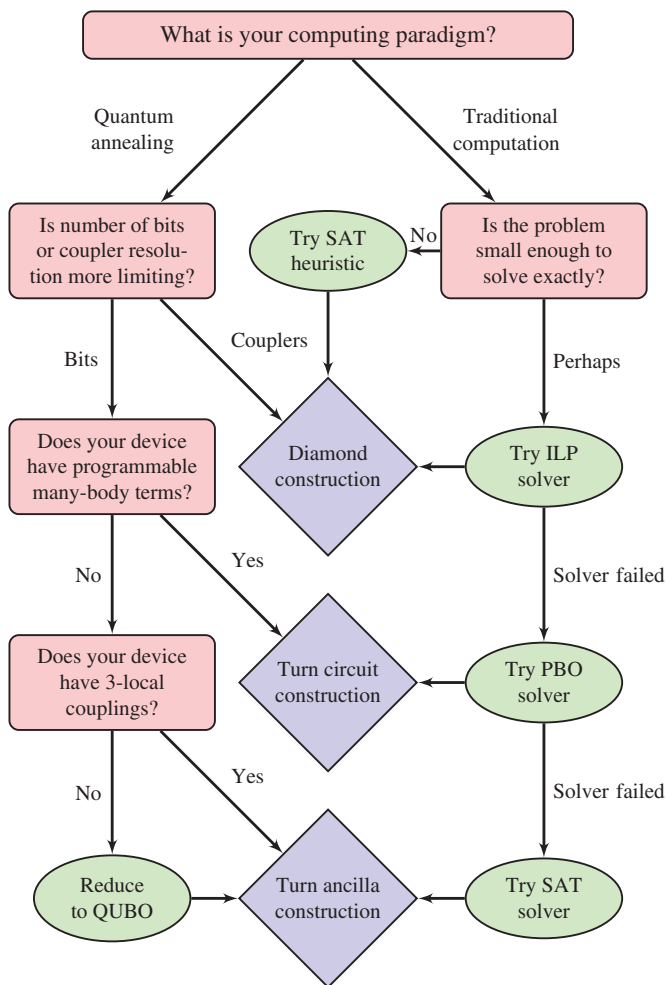


Figure 1. Flow chart describing how one might choose between the three problem encodings discussed in this chapter based on available computing resources. The “diamond encoding” is not very efficient but produces a sparse QUBO matrix without requiring reductions that increase the required coupler resolution. This makes it a natural choice for classical integer–linear programming (ILP) and heuristic satisfiability (SAT) solvers that perform best on underconstrained problems. The “turn circuit” representation is an overconstrained, but highly efficient, mapping that works best for methods designed to solve high-local expressions such as many-body ion trap simulators or pseudo-Boolean optimization (PBO) solvers. The “turn ancilla” encoding represents a balance of these benefits as it is relatively efficient and can easily collapse to 2-local without extremely high term coefficients.

technique to construct the energy function for particles interacting in a lattice [35]. The purpose of this contribution is to introduce the reader to several different types of mappings that have distinct advantages or disadvantages depending on problem size, complexity, and available resources.

3. *Map Boolean Representation to Desired Constraint Programming.* In most cases, one can take advantage of significantly more powerful solvers by making a final transformation from pseudo-Boolean function to weighted maximum satisfiability (W-SAT), ILP or QUBO. When cast as a W-SAT problem, one can take advantage of both heuristic and exact W-SAT solvers that have been developed by the computer science community and tested every year in annual “SAT Competitions.” When represented as an ILP problem, one can use commercial logistics scheduling software such as IBM’s CPLEX. If one wishes to implement the energy expression on a quantum device, it may be necessary to manipulate the energy expression so that it contains only local fields and two-body couplings. So the final step is often to reduce the dimensionality of the pseudo-Boolean expression to 2-local so that the problem can be implemented as QUBO on currently existing architectures for adiabatic quantum computing as was done in Ref. [36].

II. THE “TURN” ENCODING OF SELF-AVOIDING WALKS

A. Embedding Physical Structure

Let us use the term “fold” to denote a particular self-avoiding walk (SAW) assumed by the ordered chain of beads or “amino acids” on a square lattice. These configurations include amino acid chains that might intersect at different points due to amino acids occupying the same lattice sites. Even though overlapping folds will exist in the solution space of our problem, these folds are unphysical and therefore we need to construct energy functions to penalize such configurations. Such functions will be discussed in detail below.

A fold of an N -amino acid protein is represented in what we refer to as the “turn” mapping by a series of $N - 1$ turns. We use this name to distinguish the encoding from other (spatial) representations that encode the possible folds by explicitly encoding the grid location of each amino acid. The square lattice spatial representation discussed in Ref. [35] has the advantage of being general for the problem of N particles interacting in a lattice (which need not be connected) but we can do much better in terms of the number of variables needed; bit efficiency is the main advantage of the turn mapping.

In the turn mapping, one saves bits by taking advantage of the connectivity of a valid SAW to store information about where each amino acid is relative to the

previous amino acid instead of encoding explicit amino acid locations. Therefore, instead of encoding the positions of the j amino acids in the lattice, we encode the j th turn taken by the $(j + 1)$ th amino acid in the chain. For pedagogical purposes, we concentrate on the case of a two-dimensional (2D) lattice SAW; the extension to a three-dimensional lattice requires a straightforward extension of the same techniques described here for the 2D case.

Because the location of an amino acid in the turn mapping is specified by its location relative to the previous acid in the primary sequence, the solution space consists only of paths, or “worms,” embedded in the lattice. The resulting energy function is invariant under translation, rotation, and reflection with respect to the embedding in physical space as long as the local structure of the relative locations is kept intact. More specifically, each of the $N - 1$ turns in 2D space requires 2 bits so that each of the four directions (up, down, left, and right) has a unique representation. This assumes a rectilinear lattice, but the method is equally valid, though with slight modification, for other lattices, for example, triangular. The convention or “compass” used in this chapter is presented in the upper-left part of Fig. 2. Furthermore, we can fix the first three bits to obtain only solutions that are rotationally invariant. Under this convention, the bit string q is written as

$$q = 01 \underbrace{0q_1}_{\text{turn 2}} \underbrace{q_2q_3}_{\text{turn 3}} \cdots \underbrace{q_{2(N-1)-4}q_{2(N-1)-3}}_{\text{turn } (N-1)} \quad (1)$$

We have chosen to fix the first three bits as 010 so that the walk always turns first to the right and then either right or down. This does not affect the structure of the solution space and leaves only $N - 2$ turns to be specified; an example is provided in Eq. (1). Since every turn requires 2 bits, the turn mapping requires

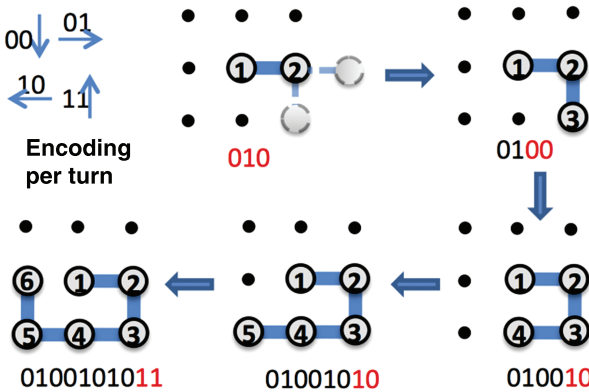


Figure 2. Step-by-step construction of the binary representation of a particular six-unit lattice protein in the turn encoding. Two qubits per bond are needed and the turn “compass” (bond directions) are denoted as “00” (downward), “01” (rightward), “10” (left), and “11” (upward). This image has been reproduced from Ref. [36] with permission from the authors.

only $2(N - 2) - 1 = 2N - 5$ bits to represent a fold. This can be compared with the $(2N - 4) \log_2 N$ required for the spatial mapping in Ref. [35]. To clearly demonstrate how this mapping works, an example of the turn encoding for a short SAW is shown in Fig. 2.

B. “Turn Ancilla” Construction of $E(q)$

Now that we have a mapping function that translates a length $2N - 5$ bit string into a specific fold in the 2D lattice, we can construct $E(q)$ as a function of these binary variables. For the case of lattice folding, we need to penalize folds where two amino acids overlap; that is, the chain must be self-avoiding. This penalty will be given by the energy function, $E_{\text{overlap}}(q)$, that returns an extremely high value if and only if amino acids overlap. While it is possible to construct a single function $E_{\text{overlap}}(q)$ that penalizes all potential overlaps, we will show that less ancillary bits are needed if we introduce the function $E_{\text{back}}(q)$ that penalizes the special case of overlaps that happen because the chain went directly backward on itself. In this scheme, $E_{\text{overlap}}(q)$ will apply to all other potential overlaps.

Finally, we must consider the interaction energy among the different amino acids. This will ultimately determine the structure of the lowest energy fold. The energy given by the pairwise interaction of beads in our chain will be given by $E_{\text{pair}}(q)$. In some lattice protein models such as the hydrophobic–polar (HP) protein folding model, there is only one stabilizing interaction; however, the construction we present here applies for an arbitrary interaction matrix among the different amino acids (or particles to be even more general). One of the advantages of the turn representation over the spatial representation is that we do not need to worry about having the amino acids linked in the right order (primary sequence), since this is guaranteed by design. The construction of the energy function

$$E(q) = E_{\text{back}}(q) + E_{\text{overlap}}(q) + E_{\text{pair}}(q) \quad (2)$$

involves a series of intermediate steps that we outline next.

1. Construction of $E_{\text{back}}(q)$

In order to have a valid SAW, we need to guarantee that our “worm” does not turn left and then immediately turn right or vice versa, or turn up and then immediately turn down or vice versa. In order to program this constraint into the energy function, we will introduce several simple logic circuits. Looking at the compass provided in Fig. 2, it should be clear the circuits in Figs. 3–6 return TRUE if and only if a particular turn (encoded $q_1 q_2$) went right, left, up, or down, respectively.

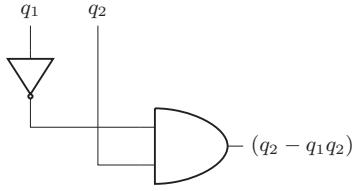


Figure 3. A logic circuit representing “right” consisting of a NOT gate after the first bit and an AND gate. Evaluates to TRUE if and only if $q_1, q_2 = 0, 1$.

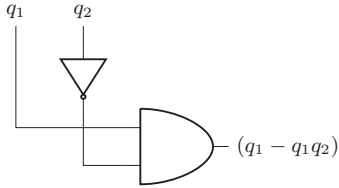


Figure 4. A logic circuit representing “left” consisting of a NOT gate after the second bit and an AND gate. Evaluates to TRUE if and only if $q_1, q_2 = 1, 0$.

Using these circuits we can generalize the concept of “up,” “down,” “left,” and “right” functions to precise directional strings. In two dimensions (as prescribed by Fig. 2), we have the functions for the j th turn:

$$d_{x+}^j = q_{2j}(1 - q_{2j-1}) = q_{2j} - q_{2j}q_{2j-1} \quad (3)$$

$$d_{x-}^j = (1 - q_{2j})q_{2j-1} = q_{2j-1} - q_{2j}q_{2j-1} \quad (4)$$

$$d_{y+}^j = q_{2j}q_{2i-1} \quad (5)$$

$$d_{y-}^j = (1 - q_{2j})(1 - q_{2j-1}) = 1 - q_{2j} - q_{2j-1} + q_{2j}q_{2j-1} \quad (6)$$

which evaluate to TRUE if and only if the j th turn is to the right, left, up, or down, respectively. Having defined these circuits, we can construct a more complicated circuit that takes two turns (the 4 bits $q_i q_{i+1} q_{i+2} q_{i+3}$) as input and returns TRUE if and

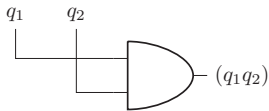


Figure 5. A logic circuit representing “up.” Only TRUE if $q_1, q_2 = 1, 1$.

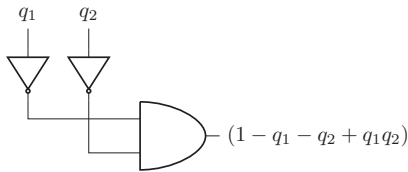
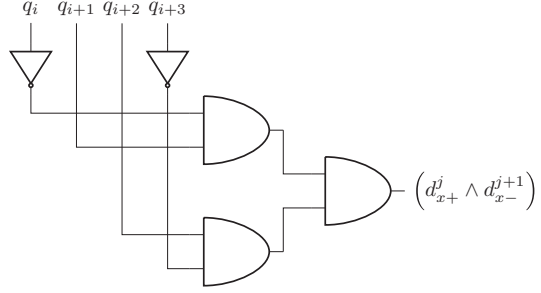


Figure 6. A logic circuit representing “down.” Only TRUE if $q_1, q_2 = 0, 0$.

Figure 7. A logic circuit that returns TRUE if and only if $(d_{x+}^j \wedge d_{x-}^{j+1})$, that is, the turn sequence $q_i q_{i+1} q_{i+2} q_{i+3} = 0110$, meaning that it went right and then left.



only if the second turn went backward, that is, $(d_{x+}^j \wedge d_{x-}^{j+1}) \vee (d_{x-}^j \wedge d_{x+}^{j+1}) \vee (d_{y+}^j \wedge d_{y-}^{j+1}) \vee (d_{y-}^j \wedge d_{y+}^{j+1})$. An example of these conjunctions $(d_{x+}^j \wedge d_{x-}^{j+1})$ is shown in Fig. 7.

The other three conjunctions are also trivially constructed by combining the appropriate circuits using AND gates that simply multiply together the directional strings. The utility of these circuits is that they produce terms in a pseudo-Boolean function. Specifically we get the terms,

$$(d_{x+}^j \wedge d_{x-}^{j+1}) = q_{i+1}q_{i+2} - q_i q_{i+1}q_{i+2} - q_{i+1}q_{i+2}q_{i+3} + q_i q_{i+1}q_{i+2}q_{i+3} \quad (7)$$

$$(d_{x-}^j \wedge d_{x+}^{j+1}) = q_i q_{i+3} - q_i q_{i+1}q_{i+3} - q_i q_{i+2}q_{i+3} + q_i q_{i+1}q_{i+2}q_{i+3} \quad (8)$$

$$(d_{y+}^j \wedge d_{y-}^{j+1}) = q_i q_{i+1} - q_i q_{i+1}q_{i+2} - q_i q_{i+1}q_{i+3} + q_i q_{i+1}q_{i+2}q_{i+3} \quad (9)$$

$$(d_{y-}^j \wedge d_{y+}^{j+1}) = q_{i+2}q_{i+3} - q_i q_{i+2}q_{i+3} - q_{i+1}q_{i+2}q_{i+3} + q_i q_{i+1}q_{i+2}q_{i+3} \quad (10)$$

It might seem logical to finish this circuit by combining all four backward overlap circuits with OR gates; however, this is not an advisable strategy as it is sure to produce many high-ordered terms. Because exactly one or none of these circuits will be TRUE, we can accomplish the same result by summing the four circuits. Accordingly, for the two turns $q_i q_{i+1} q_{i+2} q_{i+3}$, the pseudo-Boolean expression

$$(d_{x+}^j \wedge d_{x-}^{j+1}) + (d_{x-}^j \wedge d_{x+}^{j+1}) + (d_{y+}^j \wedge d_{y-}^{j+1}) + (d_{y-}^j \wedge d_{y+}^{j+1}) \quad (11)$$

evaluates to TRUE if and only if $q_i q_{i+1} q_{i+2} q_{i+3}$ represents a backward turn and evaluates to FALSE otherwise. Our goal is to construct a pseudo-Boolean expression that returns a penalty whenever a backward turn is made; therefore, we must multiply this expression by a constant to be determined later, known as λ_{overlap} . After substituting Eqs. (7)–(10) into Eq. (11), factoring the terms, and adding in λ_{overlap} , we can write

$$E_{\text{back}}(q_i q_{i+1} q_{i+2} q_{i+3}) = \lambda_{\text{overlap}} (2q_i q_{i+2} - q_i - q_{i+2}) (2q_{i+1} q_{i+3} - q_{i+1} - q_{i+3}) \quad (12)$$

To construct the entire $E_{\text{back}}(\mathbf{q})$, we need to sum together bits from each pair of adjacent turns. Keeping in mind that we fix the first three bits at 010, we write the final expression for $E_{\text{back}}(\mathbf{q})$ as,

$$E_{\text{back}}(\mathbf{q}) = \lambda_{\text{overlap}} (q_1 q_2 + q_2 q_3 - 2q_1 q_2 q_3) + \lambda_{\text{overlap}} \sum_{i=2}^{2N-8} (2q_i q_{i+2} - q_i - q_{i+2}) (2q_{i+1} q_{i+3} - q_{i+1} - q_{i+3}) \quad (13)$$

In this expression, the first three terms come from ensuring that the second turn (which begins with a bit fixed at 0) does not overlap with the third turn. Notice that in this expression, the first physical bit with an unknown value is labeled “ q_1 ” despite the fact that the first three information bits are fixed at 010. This formalism will be consistent throughout our chapter.

It is important to point out that while the decision to use a separate $E_{\text{back}}(\mathbf{q})$ instead of a more general $E_{\text{overlap}}(\mathbf{q})$ has the disadvantage of introducing 3- and 4-local terms, it has the advantage of construction without any ancillary bits. Furthermore, even if one needs an entirely 2-local expression, this strategy may still be preferable because the same reductions needed to collapse this expression to 2-local will be needed in collapsing the pairwise energy function to 2-local by construction. For more on reductions, see Section VI.

2. Construction of $E_{\text{overlap}}(\mathbf{q})$ with Ancilla Variables

The overlap energy function $E_{\text{overlap}}(\mathbf{q})$ penalizes configurations in which any two amino acids share the same lattice point. The penalty energy associated with any pair of amino acids overlapping must be large enough to guarantee that it does not interfere with the spectrum of the valid configurations (we return to the topic of choosing penalty values later on). We begin by defining a function that specifies the x and y grid positions of each amino acid. Because the directional strings we defined earlier in Eqs. (7)–(10) keep track of the direction of every step, we can

define these functions as

$$x_n = 1 + q_1 + \sum_{k=2}^{n-1} (d_{x+}^k - d_{x-}^k) \quad (14)$$

$$y_n = q_1 - 1 + \sum_{k=2}^{n-1} (d_{y+}^k - d_{y-}^k) \quad (15)$$

where the position of the n th amino acid in the sequence is a function of the preceding $n - 1$ turns iterated through with index k . Note that the terms in front of the sum are determined by the first three (fixed) bits: 010. With these definitions, we can make an extremely useful function that will return the square of the grid distance between any two amino acids (denoted i and j):

$$g_{ij} = (x_i - x_j)^2 + (y_i - y_j)^2 \quad (16)$$

g_{ij} has several extremely useful properties worth pointing out now. First, g_{ij} is zero if and only if two amino acids overlap; otherwise, g_{ij} is always positive. Additionally, g_{ij} has the very surprising property of being natively 2-local when constructed using the compass that we defined in Fig. 2 (therefore the decision to encode directions in that fashion was not arbitrary). This is surprising because the directional strings are 2-local, so we might naïvely expect something that involves the square of these to be 4-local; however, this turns out not to be the case because x_n and y_n are 1-local by construction.

In order to use g_{ij} to construct $E_{\text{overlap}}(q)$, we need a function that takes g_{ij} as input and returns a penalty if and only if $g_{ij} = 0$. First, we note the bounds on g_{ij} :

$$0 \leq g_{ij} \leq (i - j)^2 \quad (17)$$

To help enforce the constraint that $g_{ij} \geq 1$, we introduce a free parameter, α_{ij} . In the optimization literature, such a variable is called a “slack variable” and is used to convert an inequality into an equality. In our case,

$$0 \leq \alpha_{ij} \leq (i - j)^2 - 1 \quad (18)$$

This implies that

$$\forall g_{ij} \geq 1 \exists \alpha_{ij} : (i - j)^2 - g_{ij} - \alpha_{ij} = 0 \quad (19)$$

Furthermore, if and only if $g_{ij} = 0$,

$$(i - j)^2 - g_{ij} - \alpha_{ij} \geq 1 \forall \alpha_{ij} \quad (20)$$

In order to introduce a slack variable such as α_{ij} into the construction of our pseudo-Boolean function, we must encode it using ancilla bits. Ancilla bits are real, unconstrained bits used in the calculation that have no physical significance to the particular problem mapping (i.e., ancilla bits do not tell us anything about a particular protein fold). In using ancilla, we increase the dimensionality of the solution space of our problem by introducing extra variables but gain the ability to use those bits in our energy function.

Every pair of amino acids that could possibly overlap will need unique bits to form an α for use in the $E_{\text{overlap}}(\mathbf{q})$ term corresponding to that pair. Only amino acids that are an even number of turns apart can possibly overlap and we are already preventing amino acids that are two turns apart from overlapping with $E_{\text{back}}(\mathbf{q})$; thus, the number of amino acid pairs that require a slack variable is calculated as

$$\sum_{i=1}^{N-4} \sum_{j=i+4}^N [(1 + i - j) \bmod 2] \quad (21)$$

Each α_{ij} can be represented in binary using the corresponding ancilla bits. Using Eq. (18) we see that the α_{ij} corresponding to amino acid pair i, j can be represented in μ_{ij} ancilla bits, where

$$\mu_{ij} = \lceil 2 \log_2 (i - j) \rceil [(1 + i - j) \bmod 2] \quad (22)$$

Therefore, the total number of ancilla bits required to form $E_{\text{overlap}}(\mathbf{q})$ is

$$\sum_{i=1}^{N-4} \sum_{j=i+4}^N \mu_{ij} \quad (23)$$

Finally, we can write the formula for a given α_{ij} as

$$\alpha_{ij} = \sum_{k=0}^{\mu_{ij}} q_{c_{ij}+k} 2^k \quad (24)$$

where c_{ij} denotes a pointer to the first ancilla bit corresponding to a particular amino acid pair. For instance, if the $E_{\text{overlap}}(\mathbf{q})$ ancilla are in sequential order from lowest index pair to highest index pair and come immediately after the information

bits, then we could write

$$c_{ij} = \sum_{m=1}^i \left(\sum_{n=m+4}^N \mu_{mn} \right) - \sum_{n=j}^N \mu_{in} \quad (25)$$

However, there are still several problems we must address before we can construct $E_{\text{overlap}}(\mathbf{q})$. To begin with, we originally wanted an α_{ij} that was specifically restricted to the domain given in Eq. (18) but since we cannot constrain the physical bits in any fashion, Eqs. (22) and (24) suggest that our slack variable is actually in the domain given by

$$0 \leq \alpha_{ij} \leq 2^{\mu_{ij}} - 1 \quad (26)$$

We should adjust Eqs. (19) and (20) so that

$$\forall g_{ij} \geq 1 \exists \alpha_{ij} : 2^{\mu_{ij}} - g_{ij} - \alpha_{ij} = 0 \quad (27)$$

Furthermore, if and only if $g_{ij} = 0$,

$$2^{\mu_{ij}} - g_{ij} - \alpha_{ij} \geq 1 \forall \alpha_{ij} \quad (28)$$

Finally, there is the question of how to guarantee that α_{ij} is the particular α_{ij} that gives 0 in Eq. (27) whenever $g_{ij} \geq 1$. Even though there exist α_{ij} such that Eq. (27) evaluates to 0, it is also possible to have α_{ij} such that Eq. (27) evaluates to a negative value. Negative values would incentivize overlaps instead of penalizing them, so to ensure that the lowest energy solution always has $E_{\text{overlap}}(\mathbf{q}) = 0$ we square the expression to obtain the following formula:

$$\gamma_{ij} = \lambda_{\text{overlap}} [2^{\mu_{ij}} - g_{ij} - \alpha_{ij}]^2 \quad (29)$$

The expression γ_{ij} is effective for our purposes because α_{ij} 's restricted domain given by Eq. (26) promises that γ_{ij} can only equal zero if $g_{ij} \geq 1$. γ_{ij} is zero only if $g_{ij} \geq 1 \wedge \alpha_{ij} = 2^{\mu_{ij}} - g_{ij}$; thus, the goal is to make λ_{overlap} a sufficiently large penalty that all low-energy solutions must have no overlaps, that is, $g_{ij} \geq 1$ for all ij , and $\alpha_{ij} = 2^{\mu_{ij}} - g_{ij}$. Finally, we can write the final expression

$$E_{\text{overlap}}(\mathbf{q}) = \sum_{i=1}^{N-4} \sum_{j=i+4}^N [(1+i-j) \bmod 2] \gamma_{ij} \quad (30)$$

Again, we include the term $[(1+i-j) \bmod 2]$ because only amino acids that are an even number apart have the possibility of overlapping. Furthermore, because overlaps between adjacent amino acids are impossible and overlaps between amino acids two turns apart are prevented by $E_{\text{back}}(\mathbf{q})$, we start the second sum at $j = i + 4$. Accordingly, one should only create ancillary bits for pairs in which

$(i - j) \bmod 2 = 0 \wedge i - j \geq 4$. It should now be clear that the reason we introduced $E_{\text{back}}(\mathbf{q})$ was that we used fewer ancillary bits in this step.

3. Construction of $E_{\text{pair}}(\mathbf{q})$ with Ancilla Variables

Finally, we need to construct the pairwise interaction energy function. To do this, we need to make an interaction matrix, \mathbf{J} , that contains all of the pairwise interactions that lower the energy when two amino acids are adjacent on the lattice (thus all elements of \mathbf{J} are negative or zero). Note that this interaction matrix must contain many zero-valued elements as many amino acid pairs cannot possibly be adjacent. For instance, only amino acids that are at least three turns apart and an odd number of turns apart can ever be adjacent. Furthermore, depending on the interaction model, many of these amino acids might not “interact”; for instance, in the HP model only H–H pairs can interact whereas in the MJ model all amino acids can interact.

For each potential interaction, we must introduce one ancillary bit denoted ω_{ij} , where i and j denote the amino acids involved in the interaction. ω_{ij} is essentially a switch that is only “on” without incurring an energy penalty if two amino acids are interacting (i.e., if $g_{ij} = 1$). We can now write the pairwise interaction term:

$$\varphi_{ij} = \omega_{ij} J_{ij} (2 - g_{ij}) \quad (31)$$

This simple function does everything we need to write the pair function. Because $E_{\text{overlap}}(\mathbf{q})$ ensures that $g_{ij} \geq 1$, we see that φ_{ij} is positive only if both J_{ij} and ω_{ij} are nonzero and g_{ij} is greater than 2. Such solutions will never be part of the low-energy landscape for our problem because the energy could be made lower by trivially flipping the ω_{ij} ancillary bit. On the other hand, $\varphi_{ij} = J_{ij}$ if and only if $g_{ij} = 1 \wedge \omega_{ij} = 1$, which means that the pair is adjacent! Thus, the final form of $E_{\text{pair}}(\mathbf{q})$ is

$$E_{\text{pair}}(\mathbf{q}) = \sum_{i=1}^{N-1} \sum_{j=i+3}^N \omega_{ij} J_{ij} (2 - g_{ij}) \quad (32)$$

C. “Turn Circuit” Construction of $E(\mathbf{q})$

The turn ancilla construction has the advantage of providing an energy expression with relatively few many-body terms, but it does so at the cost of introducing ancilla bits. If one intends to use a pseudo-Boolean solver or a quantum device with adjustable many-body couplings, bit efficiency is much more important than the particular structure of the energy expression. This section explains the so-called “circuit” construction that provides optimal efficiency at the cost of

introducing high-ordered many-body terms. The turn circuit construction (along with reductions explained in Section VI) was used to encode problems into a quantum annealing machine in Ref. [36].

1. Sum Strings

The circuit construction works by keeping track of the turns in between amino acids to determine if the amino acids overlap or not. To do this, we keep track of the turns in every direction using the directional strings defined in Eqs. (7)–(10). Using these directional strings, we introduce ancillary bits referred to as “sum strings.” Sum strings are strings of $\lceil \log_2(j - i) \rceil$ bits for each segment of the chain between amino acids i and j , with $1 \leq i < j \leq N$ and $i + 1 < j$. As in the case of the directional strings, we require one “sum string” per direction per pair of amino acids to be compared. Each represents, in binary, the number of total turns in a particular direction within the segment.

As in the ancilla construction, whether or not two amino acids interact or overlap depends on the sequence of turns between them. To determine this, for each segment of the directional strings we construct a string that is the sum, in binary, of the bits between two amino acids, that is, the total number of turns in that direction. This process is most straightforwardly described using a circuit model. Consider, a single half-adder (HA) gate consisting of an AND and a XOR gate, as shown in Fig. 8. The output of a half-adder can be interpreted as the 2-bit sum of its two input bits. Accordingly, if we wanted to add 3 bits we could add two of them, and then add the resultant 2-bit number to the third bit, as shown in Fig. 9.

In general, to add a single bit to an n -bit number, we simply apply n half-adders. First, a half-adder applied to the single bit and the least significant bit of the augend gives the least significant bit of the sum. Next, we use a second half-adder to add the carry bit of the first addition and the second least significant bit of the augend

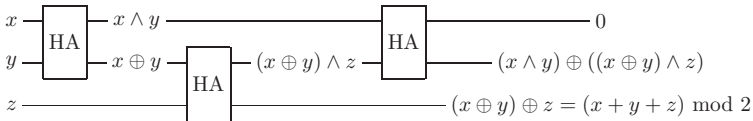


Figure 9. A circuit to sum 3 bits.

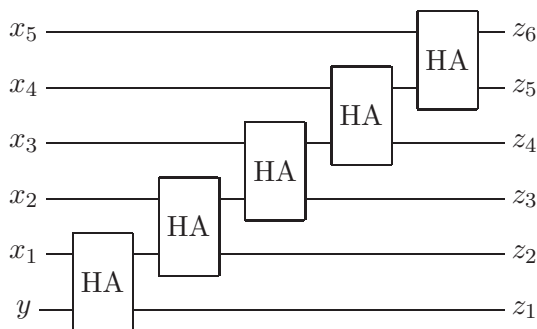


Figure 10. Circuit for the addition of a single bit y to the 5-bit $x = x_5x_4x_3x_2x_1$ to form the 6-bit sum $x + y = z_6z_5z_4z_3z_2z_1$.

to give the second least significant bit of the sum. This process is repeated until the $(n + 1)$ -bit sum is computed. For an example of this, see Fig. 10.

Thus, given an arbitrary number of bits we can find their sum, in binary, by successively combining the strategies shown in Fig. 11, that is, first adding the first three bits (see the first three HA gates from left to right) and then adding the next bit to the resulting 3-bit number that carries the previous sum. This is accomplished by the next three HA gates. From then and on, one adds a simple bit to each of the resulting n -bit number by using n HA gates until all bits in the string are added.

We can use the circuit shown in Fig. 12 to compute the binary digits of a particular sum that will be very useful to us:

$$s_{k\pm}^r(i, j) = r^{\text{th}} \text{ digit of } \sum_{p=i}^{j-1} d_{k\pm}^p \quad (33)$$

This sum tells us how many turns our protein has taken in the $\pm k$ direction between any two amino acids. For instance, $s_{x-}^1(3, 9)$ would tell us the value of the first binary digit of an integer representing the number of times that the protein turned in the negative x direction (i.e., left) between amino acids 3 and 9. While the size of the output of the circuit given in Fig. 12 scales exactly with the size of the

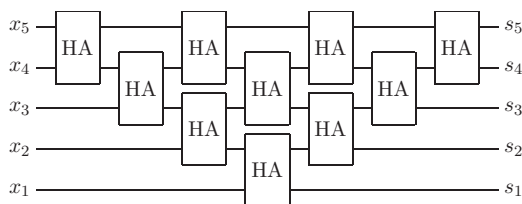


Figure 11. The circuit for the sum, $s_1s_2s_3s_4s_5$, of 5 bits $x_1 + x_2 + x_3 + x_4 + x_5$.

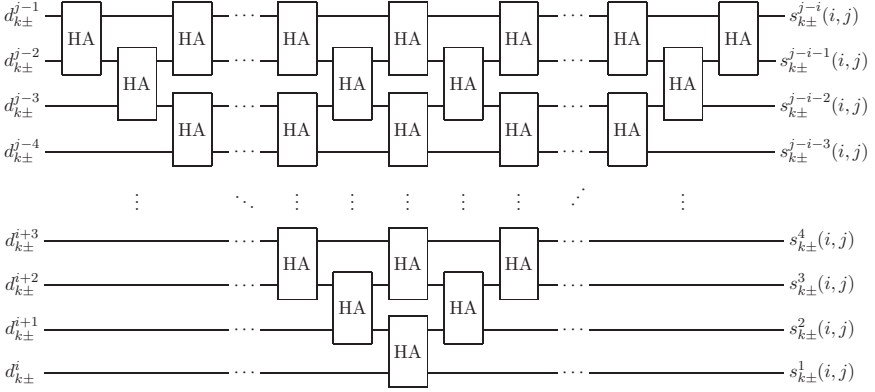


Figure 12. The circuit for the number $s_{k\pm}(i, j)$ of turns between amino acids i and j in the $\pm k$ direction.

input, the maximum number of bits needed to represent the sum of a set of bits scales logarithmically; therefore, many of the bits representing higher places in the sequence are zero. Specifically, the sum of n bits requires at most $\lceil \log_2 n \rceil$ bits to represent in binary.

2. Construction of $E_{\text{overlap}}(q)$ with Circuit

The overlap penalty should be positive if any two amino acids are at the same lattice point. For a pair i, j , this occurs when the number of turns between them in each direction $k\pm$ is equal to those in the opposite direction $k\mp$ or, equivalently, when the bit strings representing those numbers, $s_{k+}^{j-i} \dots s_{k+}^1$ and $s_{k-}^{j-i} \dots s_{k-}^1$, are the same. As discussed above, since only the first $\lceil \log_2(j-i) \rceil$ digits of $s_{k\pm}$ are nonzero, the overlap penalty function for amino acids i, j is

$$E_{\text{overlap}}(i, j) = \prod_{k=1}^D \left(\prod_{r=1}^{\lceil \log(j-i) \rceil} \text{XNOR}(s_{k+}^r(i, j), s_{k-}^r(i, j)) \right) \quad (34)$$

where

$$\text{XNOR}(p, q) = 1 - p - q + 2pq \quad (35)$$

is the exclusive NOR function that evaluates to TRUE if and only if the 2 bits have the same value. Furthermore, we need not consider every pair of amino acids in the sequence because in order for the number of turns in opposite directions to be equal, there must be an even number of total turns. The total on-site penalty

function is

$$E_{\text{overlap}} = \lambda_{\text{overlap}} \sum_{i=1}^{N-2} \left(\sum_{j=1}^{\lfloor (N-i)/2 \rfloor} E_{\text{overlap}}(i, i+2j) \right) \quad (36)$$

3. Construction of $E_{\text{pair}}(\mathbf{q})$ with Circuit

To determine if a pair of amino acids is adjacent on the lattice without using ancilla bits is more involved. Two amino acids are adjacent if and only if the number of turns between them in opposite directions is the same in all but one dimension and the numbers of turns in the other dimension have a difference of 1. The construction of the equality condition is the same as for the overlap function; to construct the latter condition, consider the set of 4-bit numbers, as shown in Fig. 13.

Note that when the first of two sequential binary is even, the Hamming distance between those bit strings is the same except for the least significant bit, for example, 0000 and 0001, 1000 and 1001, and 1110 and 1111. On the other hand, sequential numbers for which the lesser one is odd differ in at least two places, depending on where the rightmost 0 is in the lesser number, that is,

$$\begin{array}{r} 0000000000 \dots 01 \\ + \quad ** \dots ** \quad 011 \dots 11 \\ \hline ** \dots ** \quad 100 \dots 00 \end{array} \quad (37)$$

as in 0011 and 0100, 0111 and 1000, and 1011 and 1100.

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Figure 13. All 4-bit binary numbers and their decimal representations.

Let us use p to denote the position of the rightmost 0 in the odd, lesser number of this comparison. There are three portions of the bit strings that need attention when comparing adjacency in this case. First, all digits from the least significant and up to p need to be different. Second, all digits after p need to be the same. Third, within each possible adjacency direction ($k+$ or $k-$) there needs to be a change from $p-1$ to p . Finally, all the digits from the least significant up to the $(p-2)$ th digit need to be the same. Using these conditions, for both cases when the lesser number is either even or odd, results in the adjacency terms for each of the two dimensions and all of the possible amino acid pairs, $a_k(i, j)$:

$$\begin{aligned}
 a_k(i, j) = & \left[\prod_{w \neq k} \left(\prod_{r=1}^{\lceil \log(j-i) \rceil} \text{XNOR}(s_{w+}^r(i, j), s_{w-}^r(i, j)) \right) \right] \\
 & * \left[\text{XOR}(s_{k+}^1(i, j), s_{k-}^1(i, j)) \prod_{r=2}^{\lceil \log(j-i) \rceil} \text{XNOR}(s_{k+}^r(i, j), s_{k-}^r(i, j)) \right. \\
 & + \sum_{p=2}^{\lceil \log(j-i) \rceil} \left(\text{XOR}(s_{k+}^{p-1}(i, j), s_{k+}^p(i, j)) \prod_{r=1}^{p-2} \text{XNOR}(s_{k+}^r(i, j), s_{k+}^{r+1}(i, j)) \right. \\
 & \left. \left. * \prod_{r=1}^p \text{XOR}(s_{k+}^r(i, j), s_{k-}^r(i, j)) \prod_{r=p+1}^{\lceil \log(j-i) \rceil} \text{XNOR}(s_{k+}^r(i, j), s_{k-}^r(i, j)) \right) \right]
 \end{aligned} \tag{38}$$

Thus, total contribution of the interaction between two amino acids to the total energy function is given by

$$E_{\text{pair}}(i, j) = J_{ij} [a_x(i, j) + a_y(i, j)] \tag{39}$$

where J_{ij} is the adjacency matrix giving the energy of pairwise interactions that we used earlier. As was the case with the overlap penalty function, we need not consider all pairs of amino acids. In order for two amino acids to be adjacent, there must be an odd number of turns between them, excluding the trivial case of amino acids that are adjacent in the primary sequence. Accordingly, the total pairwise interaction function is

$$E_{\text{pair}} = \sum_{i=1}^{N-3} \left(\sum_{j=1}^{\lceil (N-i-1)/2 \rceil} E_{\text{pair}}(i, 1+i+2j) \right) \tag{40}$$

III. THE “DIAMOND” ENCODING OF SAWs

There are many different ways in which one could encode a SAW into binary. Of all the alternatives to the “turn” encoding that we have considered, one stands out for a number of reasons: the so-called “diamond encoding” lends itself to an energy function that is natively 2-local (without any reductions) and that has a very sparse QUBO matrix. Despite the fact that the diamond encoding requires no ancillary bits whatsoever, the encoding is still less bitwise efficient than the “turn encoding.” In the language of constraint satisfaction programming, this means that the clause:variable ratio is significantly lower when compared to the clause:variable in the turn encoding.

A. Embedding Physical Structure

The diamond encoding can be thought of as a more sophisticated and subtle version of the “spatial” encoding used in Ref. [35]. The key insight behind the diamond encoding is that if the first amino acid is fixed then each subsequent amino can only occupy a very restricted set of lattice points that can be enumerated independent of any knowledge of the particular fold. To clarify this point and elucidate why we refer to this as the “diamond” encoding, see Fig. 14.

Figure 14 illustrates what the “diamond” of valid lattice sites looks like for the first four amino acids in a SAW. In the diamond encoding, each bit refers to a specific lattice site that could be occupied by an amino acid in that part of the sequence. In Fig. 14, we notice that the second amino acid may occupy 4 positions,

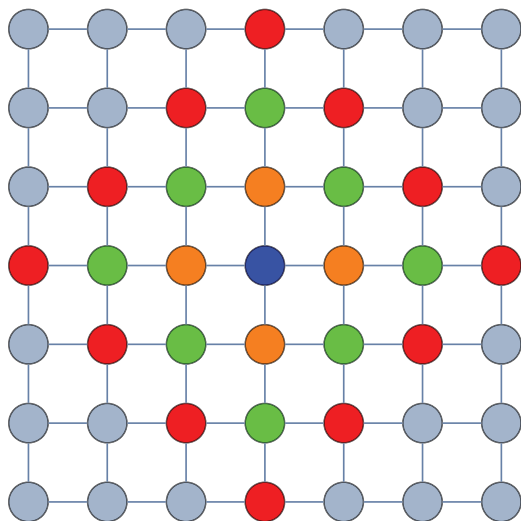


Figure 14. A “map” of the diamond encoding in 2D. If the first amino acid is fixed to the blue lattice point in the center, then the second amino acid must be on an orange lattice point, the third must be on a green lattice point, and the fourth must be on either an orange or red lattice point.

the third may occupy 8, and the fourth may occupy 16. Accordingly, we need this many bits for each amino acid:

$$\mathbf{q} = \underbrace{q_1 q_2 q_3 q_4}_{\text{2nd acid}} \underbrace{q_5 q_6 q_7 q_8 q_9 q_{10} q_{11} q_{12} \cdots}_{\text{3rd acid}} \quad (41)$$

Though very straightforward to encode, this representation makes significantly less efficient use of bits than does the turn representation. However, there are a few tricks that we can use to improve the situation for this encoding. While the “diamond” of possible lattice locations for each amino acid grows quadratically with the length of the chain, we can simultaneously save bits and drastically reduce the solution space without discarding the global minimum by deciding to set a hard limit on the size of the diamond. For instance, if a protein has length 100, then we would expect that the diamond for the 100th amino acid will have a radius of 99 lattice points at each corner. However, we can use the observation that proteins always fold into very compact structures to justify a substantial restriction on the solution space of our problem.

The very fact that these problems are typically called “protein folding” suggests that low-energy solutions involve dense conformations. Indeed, almost all heuristic methods for folding proteins take advantage of the compact nature of low-energy folds to constrain search procedures [38–40]. A large part of the reason why lattice heteropolymer problems such as protein folding are so difficult and poorly suited to heuristic algorithms is because the low-energy solutions are always very compact and thus frustrated, which makes it very unlikely that compact folds will be found efficiently via stochastic searches [41–44]. Therefore, for any interesting problem it is reasonable to assume that the protein will not stretch out further than a certain limit. To estimate this limit, one must have familiarity with the types of solutions expected of the particular problem. An examination of several publications holding current records for lowest energy folds in canonical problems suggests that for a 100-unit instance in 2D a reasonable cutoff radius would be around 20–30 lattice points. The cutoff radius could reasonably be made shorter for lattice models in higher dimensions as folds are expected to be even more compact on higher dimensional lattices. The number of bits required for the diamond encoding can be expected to grow cubically up to a limit and then linearly after that limit if a cutoff is imposed. Because the number of bits required for the turn ancilla grows quadratically, for large proteins or proteins on higher dimensional lattices the diamond encoding would actually be more efficient in bit resources.

B. Natively 2-Local $E(\mathbf{q})$

The major advantages of the diamond encoding become evident as soon as one starts to construct $E(\mathbf{q})$. The breakdown of the energy function looks different for

the diamond encoding than for the turn encoding because the diamond encoding has different strengths and weaknesses. The first difference is that the diamond encoding will require a constraint $E_{\text{one}}(\mathbf{q})$, which makes sure that each amino acid will have only one bit flipped to “on” so that each amino acid can only occupy one lattice position. Furthermore, the diamond encoding does not hardcode a primary structure constraint, so we will need a term $E_{\text{connect}}(\mathbf{q})$ to guarantee that each sequential amino acid is adjacent. Like the turn encoding, the diamond encoding will also require $E_{\text{overlap}}(\mathbf{q})$ and $E_{\text{pair}}(\mathbf{q})$ terms. The overall energy function looks like

$$E(\mathbf{q}) = E_{\text{one}}(\mathbf{q}) + E_{\text{connect}}(\mathbf{q}) + E_{\text{overlap}}(\mathbf{q}) + E_{\text{pair}}(\mathbf{q}). \quad (42)$$

1. Construction of $E_{\text{one}}(\mathbf{q})$

Each amino acid is encoded by flipping a bit in the part of the total bit-string sequence that represents that amino acid. Thus, we need to make sure that exactly one bit is flipped “on” for each amino acid. The most efficient way to guarantee this is to lower the energy whenever a bit is flipped on but introduce extremely high penalties if any two are flipped on for the same amino acid. For instance, if \mathbf{q}^k is the binary vector that represents the k th amino acid and n_k represents the length of this vector, then we can write

$$E_{\text{one}}(\mathbf{q}) = \lambda_{\text{one}} \sum_{k=2}^N \sum_{i=1}^{n_k-1} \sum_{j>i}^{n_k} q_i^k q_j^k \quad (43)$$

λ_{one} in Eq. (43) yields terms that impose a very large penalty if any two (or more) bits are flipped at once. As written, this function allows for the possibility that no bits are flipped on at once (and clearly one must be flipped on). However, the terms introduced in $E_{\text{connect}}(\mathbf{q})$ will guarantee that the low-energy solutions all have one bit flipped on. Thus, this function only needs to make sure that no more than one bit is flipped for each amino acid.

2. Construction of $E_{\text{connect}}(\mathbf{q})$

To form $E_{\text{connect}}(\mathbf{q})$, we take a very similar approach to how we formed $E_{\text{one}}(\mathbf{q})$. To guarantee that the low-energy solution space contains only amino acid chains that connect in the desired order, we couple every bit representing amino acid k to each of the $n_{k-1} \leq 4$ bits representing a lattice position adjacent to that amino acid from the previous amino acid $k-1$ and multiply by a reward as follows [using the same notation as was used in Eq. (43)]:

$$E_{\text{connect}}(\mathbf{q}) = N - 2 - \lambda_{\text{connect}} \sum_{k=2}^N \sum_{i=1}^{n_k-1} \sum_{j=1}^{n_{k-1}} q_i^k q_j^{k-1} \quad (44)$$

Note a subtle difference between the second and third sums here is that the “ -1 ” in the upper limit of the sum is subscripted in the latter but not in the former equation. Another important caveat is that $\lambda_{\text{connect}} \ll \lambda_{\text{one}}$ so that the system cannot overcome the λ_{one} penalty by having multiple λ_{connect} couplings. Finally, we put the constant factor of $N - 2$ into the equation to adjust the energy back to zero overall for valid solutions that contain $N - 2$ connections.

3. Construction of $E_{\text{overlap}}(\mathbf{q})$

It is much easier to prevent amino acids from overlapping in the diamond mapping than in the turn mapping. The only way that amino acids could overlap in the diamond mapping is for amino acids that have an even number of bonds between them to flip bits corresponding to the same lattice location. For instance, in Fig. 14 it is clear that the fourth amino acid could overlap with second amino acid since the orange lattice points are possibilities for both amino acids. Assuming that the diamond lattice positions are encoded so that the inner diamond bits come first in the bit string for each amino acid and that bits are enumerated in some consistent fashion (e.g., starting at the top and going clockwise around the diamond), we can write the following:

$$E_{\text{overlap}}(\mathbf{q}) = \lambda_{\text{overlap}} \sum_{k=2}^{N-1} \sum_{h>k}^N \sum_{i=1}^{n_k} [(1+k-h) \bmod 2] q_i^k q_i^h \quad (45)$$

This expression would perfectly sum over all the possible overlaps as the first two sums iterate through all possible overlapping pairs and the third sum iterates through all of the diamond points up to the last point they both share, n_k .

4. Construction of $E_{\text{pair}}(\mathbf{q})$

To form the pairwise interaction term, we simply couple each bit to the possible adjacent lattice locations that could be occupied by other amino acids. The strength of the coupling will depend on the interaction matrix element between the two amino acids coupled by the term. Additionally, we note that amino acids are only able to be adjacent if there are an even number of amino acids (2 or greater) in between the two. Thus, the formula is as follows:

$$E_{\text{pair}}(\mathbf{q}) = \sum_{k=2}^{N-1} \sum_{h=k+2}^N \sum_{\langle ij \rangle} J_{hk} [(k-h) \bmod 2] q_i^k q_j^h \quad (46)$$

where the sum over $\langle ij \rangle$ is understood as a sum over bits corresponding to adjacent lattice sites. There is no straightforward way to write the function $\langle ij \rangle$ in analytical terms. Nevertheless, for large problems it is trivial to write a program that iterates through bits in the second amino acid with a for-loop and evaluate the sum on those bits if the first amino acid bit and the second amino acid bit have a grid distance of 1.

IV. PSEUDO-BOOLEAN FUNCTION TO W-SAT

In order to take advantage of state-of-the-art SAT solvers to optimize our pseudo-Boolean function, it is necessary to map the problem to W-SAT. The most general form of the generic SAT problem is known as K -SAT. In K -SAT, the problem is to find a vector of Boolean-valued variables that satisfies a list of clauses, each containing up to K variables, which constrain the solution. When K -SAT has a solution it is known as “satisfiable” and for $K \leq 2$ the problem is tractable in polynomial time. However, for $K > 2$ the problem is known to be NP-complete; in fact, 3-SAT was the first problem proved to be NP-complete [45].

A. MAX-SAT and W-SAT

MAX-SAT is a more difficult version of the canonical SAT problem that is relevant when K -SAT is either “unsatisfiable” or at least not known to be satisfiable. In MAX-SAT, the goal is not necessarily to find the solution string that satisfies all clauses (such a solution string may not even exist); rather, the goal is to find the solution string that satisfies the maximum number of clauses.

An extension of MAX-SAT known as W-SAT is what will be most relevant to us. In W-SAT, each clause is given a positive integer-valued “weight” that is added to a sum only if the clause evaluates to FALSE. Accordingly, in W-SAT the goal is to minimize this sum rather than the total number of FALSE clauses as in canonical MAX-SAT [46,47]. We can more succinctly state the problem as follows: given m number of clauses (y) each with a weight of w , minimize

$$W = \sum_{i=1}^m w_i y_i \quad : \quad y_i = \begin{cases} 1 & \text{if the } i\text{th clause is FALSE} \\ 0 & \text{otherwise} \end{cases} \quad (47)$$

The same approximation schemes and exact solver algorithms that work for MAX-SAT also work for W-SAT [48,49]. In order to use these solvers, one must first translate their pseudo-Boolean function into a W-SAT problem articulated in what is known as weighted conjunctive normal form (WCNF). In WCNF, the W-SAT problem is stated as a list of weights followed by a clause with each clause stated as an OR statement between integers representing the index of the corresponding Boolean variable in the solution vector. In WCNF, a negative integer denotes a negation. For instance, the WCNF clause “4000 9 -1 82” means $x_9 \vee \neg x_1 \vee x_{82}$ with penalty of 4000 if clause evaluates to FALSE. Figure 15 shows this clause as a logic circuit.

B. Constructing WCNF Clauses

To prepare the WCNF input file from a pseudo-Boolean function, one will need to write a short script that transforms each term in the pseudo-Boolean function into a WCNF clause. There is more than one way to accomplish this transformation

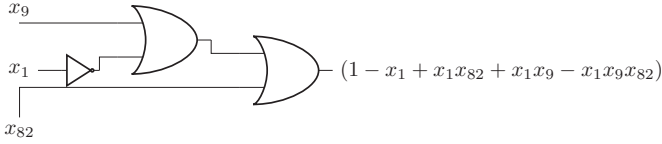


Figure 15. A logic circuit representation of the CNF clause: “9 -1 82”.

and we will only discuss one method here. For a more complete review of this topic, see Ref. [50].

It will be very useful to think of CNF clauses as logic circuits that involve only OR gates and NOT gates as in Fig. 15. Weights in WCNF notation always represent a positive value. Because pseudo-Boolean functions are treated as cost functions to minimize and the goal of W-SAT is to minimize the sum of weights on FALSE clauses, terms in the pseudo-Boolean function with a positive weight are very easy to translate in WCNF notation. To achieve this, one needs only to pass all variables in the clause through a NOT gate and then a series of OR gates (effectively making a NAND gate that takes all variables as input). This circuit is illustrated in Fig. 16 for the case of a five-variable clause.

Representing a negative weighted pseudo-Boolean term in CNF is less trivial but follows a simple pattern. To make the CNF clause positive (corresponding to negative Boolean term), one needs to construct the same circuit as in the case when the Boolean term is positive but remove one of the NOT gates. An example comprising three variables is shown in Fig. 17. However, this circuit alone does not accomplish our goal as it produces a 2-local term with negative weight in addition to the 3-local term with positive weight. Consequentially, after using the circuit in Fig. 17 to get rid of the 3-local term “ $x_1x_2x_3$,” we must subtract the term “ x_1x_2 ” multiplied by its weight from the pseudo-Boolean expression we are converting into CNF. At first glance, it is not obvious that this procedure will

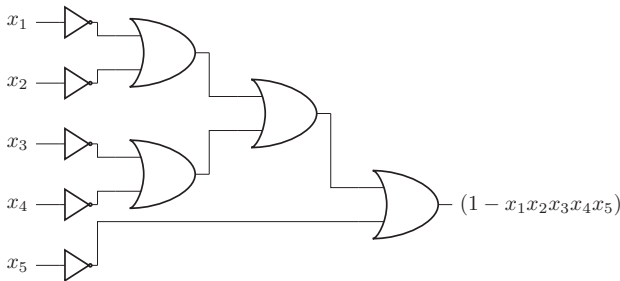


Figure 16. A logic circuit that shows that any pseudo-Boolean term with positive weight is equivalent (up to a constant) to a CNF clause with each variable negated. The term produced here is negative because the weight is added only when the clause evaluates to FALSE.

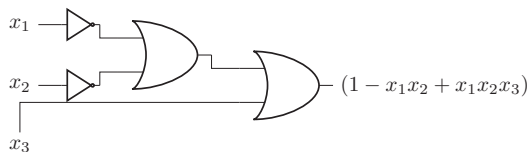


Figure 17. A logic circuit on three variables that gives a positive-valued 3-local CNF term.

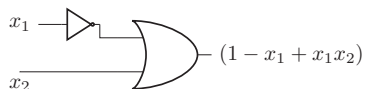


Figure 18. A logic circuit on three variables that gives a positive-valued 2-local CNF term.

get us anywhere—we turned a term into CNF only to introduce a new term into the pseudo-Boolean that we must convert back into CNF. However, the auxiliary terms produced by this circuit are of one degree less than number of variables in the term; thus, we can iterate this procedure until only the constant term remains. The next CNF clause (this time 2-local) is shown in Fig. 18.

C. Solving SAT Problems

While MAX-SAT is known to be NP-hard, there exist heuristic algorithms that are guaranteed to satisfy a fixed fraction of the clauses of the optimal solution in polynomial time. In general, oblivious local search will achieve at least an approximation ratio of $k/(k+1)$, Tabu search achieves a ratio of at least $(k+1)/(k+2)$, and nonoblivious local search achieves an approximation ratio of $(2^k - 1)/2^k$, where k is the “ K ” in K -SAT. For the special case of MAX-2-SAT, the best possible algorithm is theoretically capable of satisfying at least $(21/22) + \epsilon \approx 0.955 + \epsilon$ [9] in polynomial time [49,51]. Additionally, there are a large number of exact MAX-SAT solvers that run in superpolynomial time but in many cases can find the solution to MAX-SAT in a very short amount of time, even for problems containing hundreds of variables and clauses [17,52].

V. W-SAT TO INTEGER-LINEAR PROGRAMMING

ILP is a subset of linear programming problems in which some variables are restricted to integer domains. In general, ILP is an NP-hard problem but the importance of ILP problems (particularly for logistics scheduling) has produced many extremely good exponential-time exact solvers and polynomial-time heuristic solvers [46]. Pseudo-Boolean optimization is an even more specific case of ILP sometimes known as 0–1 ILP, where the integer variables are Boolean [47]. The mapping between W-SAT and ILP is very straightforward.

A. Mapping to ILP

In ILP, the goal is to minimize an objective function of integer-valued variables subject to a list of inequality constraints that must be satisfied. The inequality constraints come directly from the clauses in W-SAT. As described in Section IV.A, the logical clause from the WCNF clause “4000 9 \neg 1 82” (which again means $x_9 \vee \neg x_1 \vee x_{82}$ with penalty of 4000 if clause evaluates to False) can be represented as $x_9 + (1 - x_1) + x_{82} \geq 1$ s.t. $x_n \in \{0, 1\}$. In ILP, all constraints must be satisfied but in W-SAT clauses are sometimes not satisfied; to accommodate this, we introduce an auxiliary binary variable y_1 into the equation and get $y_1 + x_9 + (1 - x_1) + x_{82} \geq 1$. Thus, if the original equation is False, y_1 will have a value of True, which satisfies the inequality. We can take advantage of this auxiliary variable to construct the optimization function W . Since the clause in our example has a weight of 4000, we can write $W = 4000y_1$ s.t. $y_1 + x_9 + (1 - x_1) + x_{82} \geq 1$. Thus, the mapping between ILP and W-SAT is extremely trivial: all WCNF clauses are rewritten as linear equalities that are $\geq 1 - y_i$ by adding together the variables (or their negations) where i is the index of the clause and the objective function is written as $W = \sum_{i=1}^N w_i y_i$ where N is the number of clauses and w_i is the weight of that clause [46].

B. Solving ILP Problems

Commercial logistic scheduling software such as IBM ILOG CPLEX Optimization Studio (also known as CPLEX) is designed to solve integer programming, linear programming, and mixed integer-linear programming problems on a very large scale [53]. Constraint satisfaction problems that are sometimes very difficult to solve using conventional SAT techniques can be easier to solve using ILP techniques and vice versa. In particular, SAT solvers and specialized pseudo-Boolean optimizers seem to outperform ILP solvers when a problem is overconstrained [54]. On the other hand, for problems that are underconstrained and have a large number of variables, ILP solvers are the natural choice. In some cases, 0–1 ILP optimizers such as Pueblo will outperform both SAT solvers and commercial ILP solvers [55–57].

VI. LOCALITY REDUCTIONS

The practical ability to either exactly or approximately solve random instances of constraint satisfaction optimization such as pseudo-Boolean optimization or MAX-SAT seems to depend very sensitively on the variable to clause ratio and degree of constraint expressions [46,49,58]. In fact, the degree of constraints determines the complexity class of certain constraint satisfaction problems; for example, 2-SAT is proven to be in P whereas 3-SAT is in NP-complete [45]. Clearly for instances such as this, there can be no efficient method that reduces the degree

of constraints. Fortunately, reducing the degree of constraints in general pseudo-Boolean optimization (i.e., reducing the polynomial order of pseudo-Boolean terms) can be done efficiently.

Constraint degree reduction is particularly important if we wish to solve our problem using existing architectures for adiabatic quantum computation because available devices tend to be very limited in their ability to realize arbitrary variable couplings (especially high-ordered couplings). For instance, the D-Wave One device used for pseudo-Boolean optimization in Ref. [36] is only able to implement 2-local qubit couplings and has limited coupler resolution. To encode functions of higher locality in such setups, we must introduce ancilla bits that replace 2-local terms to reduce locality. Because these ancilla become free parameters of the system, it is also necessary to introduce penalty functions to account for the possibility that their value may be incorrect. All of this is accomplished with the function $E_{\wedge}(q_i, q_j, \tilde{q}_n; \delta_n)$ in Eq. (48) that introduces the ancillary bit q_n in order to collapse the 2-local term $q_i q_j$ with energy penalty of δ_n if $q_n \neq q_i q_j$. For a further discussion, see Refs. [35,59,60].

$$E_{\wedge}(q_i, q_j, \tilde{q}_n; \delta_n) = \delta_n(3\tilde{q}_n + q_i q_j - 2q_i \tilde{q}_n - 2q_j \tilde{q}_n) \quad (48)$$

If one desires an entirely 2-local energy function, then many $E_{\wedge}(q_i, q_j, \tilde{q}_n; \delta_n)$'s may be necessary to collapse all high-local terms. For instance, consider the complete energy function for the HP model protein H P P H P when coded in the turn ancilla mapping:

$$\begin{aligned} E = & -4q_2q_6\lambda_1 + 4q_1q_3q_6\lambda_1 + 3q_6\lambda_1 + 28q_1\lambda_2 + 25q_1q_2\lambda_2 + 108q_2\lambda_2 \\ & -56q_1q_3\lambda_2 - 50q_1q_2q_3\lambda_2 + 26q_2q_3\lambda_2 + 28q_3\lambda_2 + 24q_1q_4\lambda_2 - 16q_1q_2q_4\lambda_2 \\ & -56q_2q_4\lambda_2 - 48q_1q_3q_4\lambda_2 + 32q_1q_2q_3q_4\lambda_2 - 18q_2q_3q_4\lambda_2 + 25q_3q_4\lambda_2 \\ & + 108q_4\lambda_2 - 56q_1q_5\lambda_2 - 48q_1q_2q_5\lambda_2 + 25q_2q_5\lambda_2 + 48q_1q_3q_5\lambda_2 \\ & -50q_2q_3q_5\lambda_2 - 56q_3q_5\lambda_2 - 48q_1q_4q_5\lambda_2 + 32q_1q_2q_4q_5\lambda_2 - 18q_2q_4q_5\lambda_2 \\ & + 36q_2q_3q_4q_5\lambda_2 - 50q_3q_4q_5\lambda_2 + 25q_4q_5\lambda_2 + 28q_5\lambda_2 - 32q_1q_7\lambda_2 \\ & -96q_2q_7\lambda_2 + 64q_1q_3q_7\lambda_2 - 32q_3q_7\lambda_2 + 64q_2q_4q_7\lambda_2 - 96q_4q_7\lambda_2 \\ & + 64q_1q_5q_7\lambda_2 + 64q_3q_5q_7\lambda_2 - 32q_5q_7\lambda_2 - 32q_7\lambda_2 - 16q_1q_8\lambda_2 - 48q_2q_8\lambda_2 \\ & + 32q_1q_3q_8\lambda_2 - 16q_3q_8\lambda_2 + 32q_2q_4q_8\lambda_2 - 48q_4q_8\lambda_2 + 32q_1q_5q_8\lambda_2 \\ & + 32q_3q_5q_8\lambda_2 - 16q_5q_8\lambda_2 + 64q_7q_8\lambda_2 - 32q_8\lambda_2 - 8q_1q_9\lambda_2 - 24q_2q_9\lambda_2 \\ & + 16q_1q_3q_9\lambda_2 - 8q_3q_9\lambda_2 + 16q_2q_4q_9\lambda_2 - 24q_4q_9\lambda_2 + 16q_1q_5q_9\lambda_2 \\ & + 16q_3q_5q_9\lambda_2 - 8q_5q_9\lambda_2 + 32q_7q_9\lambda_2 + 16q_8q_9\lambda_2 - 20q_9\lambda_2 - 4q_1q_{10}\lambda_2 \\ & -12q_2q_{10}\lambda_2 + 8q_1q_3q_{10}\lambda_2 - 4q_3q_{10}\lambda_2 + 8q_2q_4q_{10}\lambda_2 - 12q_4q_{10}\lambda_2 \\ & + 8q_1q_5q_{10}\lambda_2 + 8q_3q_5q_{10}\lambda_2 - 4q_5q_{10}\lambda_2 + 16q_7q_{10}\lambda_2 + 8q_8q_{10}\lambda_2 \\ & + 4q_9q_{10}\lambda_2 - 11q_{10}\lambda_2 + 36\lambda_2 \end{aligned} \quad (49)$$

In order to reduce this function to 2-local, we will need to collapse some of the 2-local terms inside of the 3-local terms to a single bit. We enumerate all of the 3-local terms and their corresponding 2-local terms that we could use to reduce each 3-local term in Eq. (50).

$$\begin{pmatrix} q_1 & q_2 & q_3 \\ q_1 & q_2 & q_4 \\ q_1 & q_3 & q_4 \\ q_2 & q_3 & q_4 \\ q_1 & q_2 & q_3 \\ q_1 & q_2 & q_5 \\ q_1 & q_3 & q_5 \\ q_2 & q_3 & q_5 \\ q_1 & q_4 & q_5 \\ q_2 & q_4 & q_5 \\ q_1 & q_2 & q_4 \\ q_3 & q_4 & q_5 \\ q_2 & q_3 & q_4 \\ q_1 & q_3 & q_6 \\ q_1 & q_3 & q_7 \\ q_2 & q_4 & q_7 \\ q_1 & q_5 & q_7 \\ q_3 & q_5 & q_7 \\ q_1 & q_3 & q_8 \\ q_2 & q_4 & q_8 \\ q_1 & q_5 & q_8 \\ q_3 & q_5 & q_8 \\ q_1 & q_3 & q_9 \\ q_2 & q_4 & q_9 \\ q_1 & q_5 & q_9 \\ q_3 & q_5 & q_9 \\ q_1 & q_3 & q_{10} \\ q_2 & q_4 & q_{10} \\ q_1 & q_5 & q_{10} \\ q_3 & q_5 & q_{10} \end{pmatrix} \iff \begin{pmatrix} q_1 q_2 & q_1 q_3 & q_2 q_3 \\ q_1 q_2 & q_1 q_4 & q_2 q_4 \\ q_1 q_3 & q_1 q_4 & q_3 q_4 \\ q_2 q_3 & q_2 q_4 & q_3 q_4 \\ q_1 q_2 & q_1 q_3 & q_2 q_3 \\ q_1 q_2 & q_1 q_5 & q_2 q_5 \\ q_1 q_3 & q_1 q_5 & q_3 q_5 \\ q_2 q_3 & q_2 q_5 & q_3 q_5 \\ q_1 q_4 & q_1 q_5 & q_4 q_5 \\ q_2 q_4 & q_2 q_5 & q_4 q_5 \\ q_1 q_2 & q_1 q_4 & q_2 q_4 \\ q_3 q_4 & q_3 q_5 & q_4 q_5 \\ q_2 q_3 & q_2 q_4 & q_3 q_4 \\ q_1 q_3 & q_1 q_6 & q_3 q_6 \\ q_1 q_3 & q_1 q_7 & q_3 q_7 \\ q_2 q_4 & q_2 q_7 & q_4 q_7 \\ q_1 q_5 & q_1 q_7 & q_5 q_7 \\ q_3 q_5 & q_3 q_7 & q_5 q_7 \\ q_1 q_3 & q_1 q_8 & q_3 q_8 \\ q_2 q_4 & q_2 q_8 & q_4 q_8 \\ q_1 q_5 & q_1 q_8 & q_5 q_8 \\ q_3 q_5 & q_3 q_8 & q_5 q_8 \\ q_1 q_3 & q_1 q_9 & q_3 q_9 \\ q_2 q_4 & q_2 q_9 & q_4 q_9 \\ q_1 q_5 & q_1 q_9 & q_5 q_9 \\ q_3 q_5 & q_3 q_9 & q_5 q_9 \\ q_1 q_3 & q_1 q_{10} & q_3 q_{10} \\ q_2 q_4 & q_2 q_{10} & q_4 q_{10} \\ q_1 q_5 & q_1 q_{10} & q_5 q_{10} \\ q_3 q_5 & q_3 q_{10} & q_5 q_{10} \end{pmatrix} \quad (50)$$

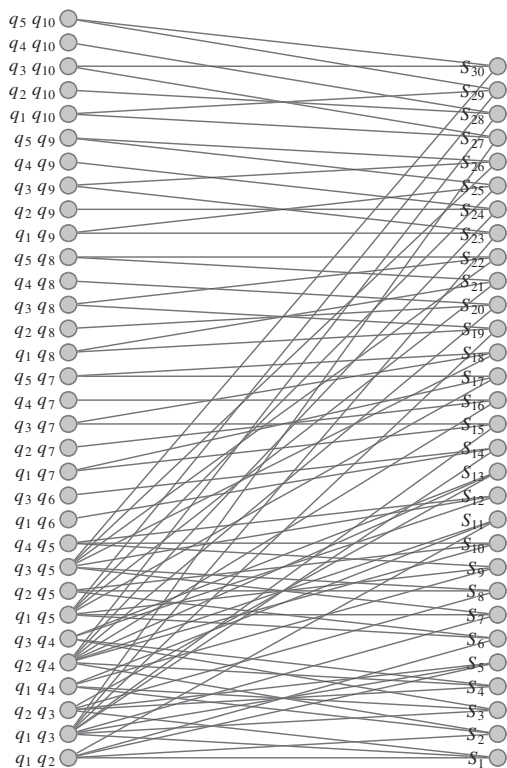


Figure 19. A bipartite graph connecting the 3-local terms (S_n) in Eq. (49) to the 2-local terms ($q_i q_j$) that collapse them.

Equation (50) shows that there are thirty 3-local terms in Eq. (49) and three different ways to collapse each of those 3-local terms. In general, the problem of choosing the most efficient 2-local terms to collapse this function is NP-complete. This becomes evident if we represent our problem as an element cover on a bipartite graph. Suppose we relabel each 3-local term on the left as “set” 1–30, denoted as $S_1 S_2 \cdots S_{30}$. We can then make the following bipartite graph that connects the 3-local terms to the 2-local terms that collapse them.

Figure 19 shows that we can now restate the problem in the following way: “choose the fewest number of 2-local terms (on the left) that cover all 3-local terms (on the right) with at least one edge.” In general, this problem is isomorphic to the canonical “hitting set” problem that is equivalent to set cover, one of Karp’s 21 NP-complete problems [61–63]. However, we have specifically kept this issue in mind when creating the turn ancilla representation in such a way as to guarantee that it is easy to find a relatively efficient solution to this problem. Accordingly, our experience has been that a greedy local search algorithm performs very well.

The explanation for this is simple: each 3- or 4-local term will contain no more than one ancillary bit; thus, to cover all 3- and 4-local terms we can focus entirely on the physical bits (in this case, bits 1–5). In alternative mappings not presented here, we have frequently encountered extremely difficult instances of the hitting set problem during the reduction process. In these situations, one should see Ref. [64] for a very efficient algorithm that can exactly solve hitting cover in $O(1.23801^n)$.

VII. QUANTUM REALIZATION

A primary goal of this chapter is to elucidate an efficient process for encoding chemical physics problems into a form suitable for quantum computation. In addition to providing the alternatives for the solution of the lattice heteropolymer problem in quantum devices, we seek to provide a general explanation of considerations for constructing energy functions for these devices. These have many possible applications for solving problems related to statistical mechanics on the device. In this section, we will complete our chapter by demonstrating the final steps required to embed a small instance of a particular lattice protein problem into a QUBO Hamiltonian.

The Hamiltonians and the number of resources presented here correspond to the minimum amount of resources needed assuming the device can handle many-body interactions as is the case for NMR quantum computers or trapped ions. The hierarchical experimental proposals presented here work for lattice folding under no external constraints, that is, amino acid chains in “free space.”¹ As a final step, we will reduce these Hamiltonians to a 2-local form specifically designed for the D-Wave One used in Refs. [36,65–67]. The final Hamiltonian we present is more efficient than that used in Ref. [36] as we have since realized several tricks to make the energy function more compact.

A. Previous Experimental Implementation

Throughout this chapter, we have referred to an experimental implementation of quantum annealing to solve lattice heteropolymer problems in Ref. [36]. The quantum hardware employed consists of 16 units of a recently characterized eight-qubit unit cell [67,68]. Post-fabrication characterization determined that only 115 qubits out of the 128-qubit array can be reliably used for computation. The array of coupled superconducting flux qubits is, effectively, an artificial Ising spin system with programmable spin–spin couplings and transverse magnetic fields. It is designed to solve instances of the following (NP-hard) classical optimization problem: given a set of local longitudinal fields (h_i) and an interaction matrix

¹ External interactions could also be included as presented and verified experimentally in Ref. [36].

(J_{ij}), find the assignment $\mathbf{s} = \mathbf{s}_1 \mathbf{s}_2 \mathbf{s}_3 \cdots \mathbf{s}_N$ that minimizes the objective function $E(\mathbf{s})$, where

$$E(\mathbf{s}) = \sum_{1 \leq i \leq N} h_i s_i + \sum_{1 \leq i < j \leq N} J_{ij} s_i s_j \quad (51)$$

and $s_i \in -1, 1$. Thus, the solution to this problem, \mathbf{s} , can be encoded into the ground-state wavefunction of the quantum Hamiltonian

$$\mathcal{H}_p = \sum_{1 \leq i \leq N} h_i \sigma_i^z + \sum_{1 \leq i < j \leq N} J_{ij} \sigma_i^z \sigma_j^z \quad (52)$$

Quantum annealing exploits the adiabatic theorem of quantum mechanics, which states that a quantum system initialized in the ground state of a time-dependent Hamiltonian remains in the instantaneous ground state, as long as it is driven sufficiently slowly. Since the ground state of \mathcal{H}_p encodes the solution to the optimization problem, the idea behind quantum annealing is to adiabatically prepare this ground state by initializing the quantum system in some easy-to-prepare ground state, \mathcal{H}_b . In this case, \mathcal{H}_b corresponds to a superposition of all states of the computational basis. The system is driven slowly to the problem Hamiltonian, $\mathcal{H}(\tau = 1) \approx \mathcal{H}_p$. Deviations from the ground state are expected due to deviations from adiabaticity, as well as thermal noise and imperfections in the implementation of the Hamiltonian.

Using the encoding methods discussed here, the authors were able to encode and to solve the global minima solution for small tetrapeptide and hexapeptide chains under several experimental schemes involving 5 and 8 qubits for four-amino acid sequence (HP model) and 5, 27, 28, and 81 qubits for the six-amino acid sequence under the MJ model for general pairwise interactions.

B. Six-Unit Miyazawa–Jernigan Protein

The example we will present here is a different encoding of the largest problem performed in Ref. [36]: the MJ protein proline–serine–valine–lysine–methionine–alanine (PSVKMA) on a 2D lattice. We will use the pairwise nearest-neighbor MJ interaction energies presented in Table 3 of Ref. [37] and shown in Fig. 20. We will use the turn ancilla construction for our energy function and






Amino acid sequence	Interaction	ΔE
		−1
		−2
		−3
		−4

Figure 20. Interaction matrix for our protein in the MJ model.

constrain the first three virtual bits to 010, as before. Recall that the turn ancilla construction requires $2N - 5$ physical information bits; thus, our six-unit MJ protein will be encoded into 7 bits.

1. $E_{\text{back}}(\mathbf{q})$ for Six-Unit SAW on 2D Lattice

Using Eq. (13), we find that our six-unit protein has the backward energy function,

$$\begin{aligned} E_{\text{back}}(\mathbf{q}) = & \lambda_{\text{back}}(q_1q_2 - 2q_1q_3q_2 + 2q_3q_2 - 2q_3q_4q_2 - 2q_3q_5q_2 + 4q_3q_4q_5q_2 \\ & - 2q_4q_5q_2 + q_5q_2 + q_3q_4 - 2q_3q_4q_5 + 2q_4q_5 - 2q_4q_5q_6 + q_5q_6 \\ & + q_4q_7 - 2q_4q_5q_7 - 2q_4q_6q_7 + 4q_4q_5q_6q_7 - 2q_5q_6q_7 + q_6q_7) \quad (53) \end{aligned}$$

Soon, we will discuss how to choose the appropriate value for λ_{back} but for now we simply note that λ_{back} and λ_{overlap} penalize the same illegal folds; thus, we realize that $\lambda_{\text{back}} = \lambda_{\text{overlap}}$.

2. $E_{\text{overlap}}(\mathbf{q})$ for Six-Unit SAW on 2D Lattice

Using Eq. (30), we calculate the overlap energy function as

$$\begin{aligned} E_{\text{overlap}}(\mathbf{q}) = & \lambda_{\text{overlap}}(96q_2q_1 - 96q_2q_3q_1 - 64q_3q_1 - 64q_2q_4q_1 + 64q_2q_3q_4q_1 \\ & - 96q_3q_4q_1 + 96q_4q_1 - 96q_2q_5q_1 + 64q_2q_4q_5q_1 - 96q_4q_5q_1 \\ & - 64q_5q_1 - 48q_2q_6q_1 + 32q_2q_3q_6q_1 - 48q_3q_6q_1 + 32q_3q_4q_6q_1 \\ & - 48q_4q_6q_1 + 32q_2q_5q_6q_1 + 32q_4q_5q_6q_1 - 48q_5q_6q_1 + 72q_6q_1 \\ & - 48q_2q_7q_1 - 48q_3q_7q_1 + 32q_2q_4q_7q_1 - 48q_4q_7q_1 + 96q_3q_5q_7q_1 \\ & - 48q_5q_7q_1 + 32q_2q_6q_7q_1 + 32q_4q_6q_7q_1 - 48q_6q_7q_1 - 8q_7q_1 \\ & - 8q_3q_{10} + 64q_3q_8q_1 + 64q_5q_8q_1 - 32q_8q_1 + 32q_3q_9q_1 \\ & + 32q_5q_9q_1 - 16q_9q_1 + 16q_3q_{10}q_1 + 16q_5q_{10}q_1 - 8q_{10}q_1 \\ & + 8q_3q_{11}q_1 + 8q_5q_{11}q_1 - 4q_{11}q_1 + 64q_3q_{12}q_1 + 64q_5q_{12}q_1 \\ & + 64q_7q_{12}q_1 - 96q_{12}q_1 + 32q_3q_{13}q_1 + 32q_5q_{13}q_1 + 32q_7q_{13}q_1 \\ & - 48q_{13}q_1 + 16q_3q_{14}q_1 + 16q_5q_{14}q_1 + 16q_7q_{14}q_1 - 24q_{14}q_1 \\ & + 8q_3q_{15}q_1 + 8q_5q_{15}q_1 + 8q_7q_{15}q_1 - 12q_{15}q_1 + 64q_1 + 144q_2 \\ & + 96q_2q_3 + 64q_3 - 64q_2q_4 - 64q_2q_3q_4 + 96q_3q_4 + 144q_4 \\ & + 96q_2q_5 - 96q_2q_3q_5 - 64q_3q_5 - 64q_2q_4q_5 + 64q_2q_3q_4q_5 \\ & - 96q_3q_4q_5 + 96q_4q_5 + 64q_5 - 8q_2q_6 - 48q_2q_3q_6 + 72q_3q_6 \\ & - 48q_2q_4q_6 - 48q_3q_4q_6 - 8q_4q_6 - 48q_2q_5q_6 + 32q_2q_3q_5q_6 \\ & - 48q_3q_5q_6 + 32q_3q_4q_5q_6 - 48q_4q_5q_6 + 72q_5q_6 + 36q_6 \\ & + 72q_2q_7 - 48q_2q_3q_7 - 8q_3q_7 - 48q_2q_4q_7 + 32q_2q_3q_4q_7 \\ & - 48q_3q_4q_7 + 72q_4q_7 - 48q_2q_5q_7 - 48q_3q_5q_7 + 32q_2q_4q_5q_7 \end{aligned}$$

$$\begin{aligned}
& -48q_4q_5q_7 - 8q_5q_7 - 48q_2q_6q_7 + 32q_2q_3q_6q_7 - 48q_3q_6q_7 \\
& + 32q_3q_4q_6q_7 + 32q_2q_5q_6q_7 + 32q_4q_5q_6q_7 - 48q_5q_6q_7 + 72q_6q_7 \\
& + 36q_7 - 96q_2q_8 - 32q_3q_8 + 64q_2q_4q_8 - 96q_4q_8 - 32q_5q_8 \\
& - 32q_8 - 48q_2q_9 - 16q_3q_9 + 32q_2q_4q_9 - 48q_4q_9 + 32q_3q_5q_9 \\
& - 16q_5q_9 + 64q_8q_9 - 32q_9 - 24q_2q_{10} + 16q_2q_4q_{10} - 24q_4q_{10} \\
& + 16q_3q_5q_{10} - 8q_5q_{10} + 32q_8q_{10} + 16q_9q_{10} - 20q_{10} - 12q_2q_{11} \\
& - 4q_3q_{11} + 8q_2q_4q_{11} - 12q_4q_{11} + 8q_3q_5q_{11} - 4q_5q_{11} + 16q_8q_{11} \\
& + 8q_9q_{11} + 4q_{10}q_{11} - 11q_{11} - 96q_2q_{12} - 96q_3q_{12} + 64q_2q_4q_{12} \\
& - 96q_4q_{12} + 64q_3q_5q_{12} - 96q_5q_{12} + 64q_2q_6q_{12} + 64q_4q_6q_{12} \\
& - 96q_6q_{12} + 64q_3q_7q_{12} + 64q_5q_7q_{12} - 96q_7q_{12} + 64q_{12} \\
& - 48q_2q_{13} - 48q_3q_{13} + 32q_2q_4q_{13} - 48q_4q_{13} + 32q_3q_5q_{13} \\
& - 48q_5q_{13} + 32q_2q_6q_{13} + 32q_4q_6q_{13} - 48q_6q_{13} + 32q_3q_7q_{13} \\
& + 32q_5q_7q_{13} - 48q_7q_{13} + 64q_{12}q_{13} + 16q_{13} - 24q_2q_{14} - 24q_3q_{14} \\
& + 16q_2q_4q_{14} - 24q_4q_{14} + 16q_3q_5q_{14} - 24q_5q_{14} + 16q_2q_6q_{14} \\
& + 16q_4q_6q_{14} - 24q_6q_{14} + 16q_3q_7q_{14} + 16q_5q_7q_{14} - 24q_7q_{14} \\
& + 32q_{12}q_{14} + 16q_{13}q_{14} + 4q_{14} - 12q_2q_{15} - 12q_3q_{15} + 8q_2q_4q_{15} \\
& - 12q_4q_{15} + 8q_3q_5q_{15} - 12q_5q_{15} + 8q_2q_6q_{15} + 8q_4q_6q_{15} \\
& - 12q_6q_{15} + 8q_3q_7q_{15} + 8q_5q_7q_{15} - 12q_7q_{15} + 16q_{12}q_{15} \\
& + 8q_{13}q_{15} + 4q_{14}q_{15} + q_{15} - 48q_4q_6q_7 + 64q_3q_5q_8) \tag{54}
\end{aligned}$$

We notice that as discussed in Section VI, all the 3-local terms here contain at least two physical information qubits (i.e., q_1 through q_7).

3. $E_{\text{pair}}(q)$ for MJ Model PSVKMA

Using the J matrix as defined in Eq. (32), we calculate the pairwise energy function as

$$\begin{aligned}
E_{\text{pair}}(q) = & -4q_2q_{16} + 4q_1q_3q_{16} + 3q_{16} - 8q_1q_{17} - 16q_2q_{17} + 8q_1q_3q_{17} \\
& - 8q_3q_{17} + 8q_2q_4q_{17} - 16q_4q_{17} + 8q_1q_5q_{17} + 8q_3q_5q_{17} - 8q_5q_{17} \\
& + 8q_2q_6q_{17} + 8q_4q_6q_{17} - 16q_6q_{17} + 8q_1q_7q_{17} + 8q_3q_7q_{17} \\
& + 8q_5q_7q_{17} - 8q_7q_{17} + 30q_{17} - 12q_1q_{18} - 12q_2q_{18} + 12q_1q_3q_{18} \\
& - 12q_3q_{18} + 12q_2q_4q_{18} - 12q_4q_{18} + 12q_1q_5q_{18} + 12q_3q_5q_{18} \\
& - 12q_5q_{18} + 21q_{18} - 16q_2q_{19} - 16q_3q_{19} + 16q_2q_4q_{19} - 16q_4q_{19} \\
& + 16q_3q_5q_{19} - 16q_5q_{19} + 16q_2q_6q_{19} + 16q_4q_6q_{19} - 16q_6q_{19} \\
& + 16q_3q_7q_{19} + 16q_5q_7q_{19} - 16q_7q_{19} + 28q_{19} \tag{55}
\end{aligned}$$

4. Setting λ Penalty Values

Finally, we will discuss how one chooses the correct penalty values for the energy function. This is a crucial step if one wishes to implement the algorithm experimentally as all currently available architectures for adiabatic quantum annealing have limited coupler resolution. That is, quantum annealing machines cannot realize arbitrary constant values for the QUBO expression. Thus, it is very important that one chooses the lowest possible penalty values that still impose the correct constraints. In our problem, we choose the value of λ_{overlap} by asking ourselves the following question: What is the greatest possible amount that any overlap could *lower* the system energy? In general, a very conservative upper bound can be obtained by simply summing together every J matrix element (which would mean that a single overlap allowed every single possible interaction to occur); in our problem, this upper bound would be -10 . Thus, we can set $\lambda_{\text{overlap}} = +10$.

5. Reduction to 2-Local

Using a standard greedy search algorithm, we find that an efficient way to collapse this energy function to 2-local is to make ancilla with the qubit pairs,

$$\begin{aligned}
 q_2 q_4 &\rightarrow q_{20} \\
 q_1 q_3 &\rightarrow q_{21} \\
 q_3 q_5 &\rightarrow q_{22} \\
 q_1 q_5 &\rightarrow q_{23} \\
 q_2 q_6 &\rightarrow q_{24} \\
 q_4 q_6 &\rightarrow q_{25} \\
 q_3 q_7 &\rightarrow q_{26} \\
 q_5 q_7 &\rightarrow q_{27} \\
 q_1 q_7 &\rightarrow q_{28}
 \end{aligned} \tag{56}$$

There is one issue left to discuss—the value of δ_n in Eq. (48). The purpose of δ_n is to constrain the reductions in Eq. (56) so that the value of the ancillary bit actually corresponds to the product of the two bits it is supposed to represent. To understand how Eq. (48) accomplishes this, see Table I. In order for Eq. (48) to work, we must choose δ_n that is large enough so that a violation of the reduction we desire will always raise the system energy. Thus, we must ensure that δ_n is large enough so that configurations that do not conform to the reduction are penalized by an amount higher than the largest penalty they could avoid and larger in magnitude than the largest energy reduction they could achieve with the illegal move. Of course, finding the exact minimum value of $E(\mathbf{q})$ is as difficult as minimizing $E(\mathbf{q})$ (our goal). Instead, we can simply make an upper bound for the penalty by setting it equal to one plus either the sum of the absolute value of

TABLE I
Truth Table for the Function $E_{\wedge}(q_i, q_j, \tilde{q}_n; \delta_n)$ from Eq. (48).

q_n	q_i	q_j	$E_{\wedge}(q_i, q_j, \tilde{q}_n; \delta_n)$
0	0	0	0
0	0	1	0
0	1	0	0
1	1	1	0
1	0	0	3δ
1	0	1	δ
1	1	0	δ
0	1	1	δ

all pseudo-Boolean coefficients corresponding to the variables being collapsed in $E(\mathbf{q})$ (whichever sum is larger).

6. QUBO Matrix and Solutions

After reduction of the energy function to 2-local, we arrive at the final pseudo-Boolean energy function. Instead of writing out the entire pseudo-Boolean expression, we will instead provide a matrix containing all of the coefficients of 1-local terms on the diagonal and 2-local terms in the upper triangular portion of this matrix. This representation is known as the QUBO matrix and contains all of the couplings needed for experimental implementation and is shown in Eq. (57). Note that the full pseudo-Boolean expression contains one constant term that we drop in the matrix representation. This constant has a value of $C = 180$ for this particular problem.

320	485	42962	480	42962	360	42962	-160	-80	-40	-20	-480	-240	-120	-60	0	-8	-12	0	-320	-85924	0	-85924	-240	-240	0	0	-85924	
0	720	490	42962	485	42962	360	-480	-240	-120	-60	-480	-240	-120	-60	-4	-16	-12	-16	-85924	-490	-490	-480	-85924	0	-240	-240	-240	
0	0	320	485	42962	360	42962	-160	-80	-40	-20	-480	-240	-120	-60	0	-8	-12	-16	-330	-85924	-85924	0	-240	-240	-85924	0	0	
0	0	0	720	490	42962	365	-480	-240	-120	-60	-480	-240	-120	-60	0	-16	-12	-16	-85924	-480	-490	-480	0	-85924	-240	-250	-240	
0	0	0	0	320	365	42962	-160	-80	-40	-20	-480	-240	-120	-60	0	-8	-12	-16	-330	0	-85924	-85924	-240	-250	0	-85924	0	
0	0	0	0	0	180	365	0	0	0	0	-480	-240	-120	-60	0	-16	0	-16	-240	-240	-240	-240	-85924	-85924	-240	-250	-240	
0	0	0	0	0	0	180	0	0	0	0	-480	-240	-120	-60	0	-8	0	-16	-240	-240	-240	-240	-240	-240	-250	-85924	-85924	
0	0	0	0	0	0	0	-160	320	160	80	0	0	0	0	0	0	0	0	0	320	320	320	320	0	0	0	0	
0	0	0	0	0	0	0	0	-160	80	40	0	0	0	0	0	0	0	0	0	160	160	160	160	0	0	0	0	
0	0	0	0	0	0	0	0	0	-100	20	0	0	0	0	0	0	0	0	0	80	80	80	80	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	-55	0	0	0	0	0	0	0	0	0	40	40	40	40	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	320	320	160	80	40	0	0	320	320	320	320	320	320	320	320	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	80	80	40	0	0	0	160	160	160	160	160	160	160	160	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	20	0	0	0	0	80	80	80	80	80	80	80	80	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	40	40	40	40	40	40	40	40	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	4	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	0	0	0	8	8	8	8	8	8	8	8	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21	0	12	12	12	12	0	0	0	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28	16	0	16	0	16	16	16	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	128566	320	340	320	0	0	160	160	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	128566	0	0	160	160	0	480	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	128566	0	160	160	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	128566	160	160	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	128846	0	160	160	160	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	128846	160	180	160	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	128846	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	128846	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	128846	

(57)

Taking the matrix in Eq. (57) as Q , we can write the total energy of a given solution (denoted by q) as

$$E(q) = qQq \quad (58)$$

The problem is now ready for its implementation on a quantum device. For our particular problem instance, the solution string is given by the bit string

$$0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0 \quad (59)$$

The energy given by Eq. (58) is -186 . In the original expression, this corresponds to an energy of $C - 186 = 180 - 186 = -6$. Let us confirm that this is accurate to the MJ model. Looking only at the physical information bits and prepending the first three constant bits (010), we see that the bit string prescribes the following fold:

$$q = \underbrace{01}_{\text{right}} \underbrace{00}_{\text{down}} \underbrace{00}_{\text{down}} \underbrace{10}_{\text{left}} \underbrace{11}_{\text{up}} \quad (60)$$

which corresponds to the fold shown in Fig. 21,

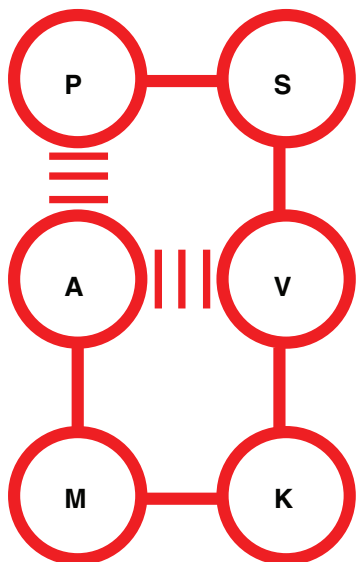


Figure 21. The solution to our example problem for MJ protein PSVKMA.

VIII. CONCLUSIONS

As both traditional and quantum computer science continue to advance as fields, domain scientists from all disciplines need to develop new ways of representing problems in order to leverage state-of-the-art computational tools. In this chapter, we discussed strategies and techniques for solving lattice heteropolymer problems with some of these tools. While the lattice heteropolymer model is widely applicable to many problems, the general principles used to optimally encode and constrain this particular application are fairly universal for discrete optimization problems in the physical sciences.

We focused on three mappings: “turn ancilla,” “turn circuit,” and “diamond.” The turn ancilla mapping is the best mapping in terms of the scaling of the number of resources for large instances, thus making it ideal for benchmark studies of lattice folding using (heuristic) solvers for pseudo-Boolean minimization. Additionally, this method shows how one can use ancilla variables to construct a fitness function with relatively few constraints per clause (i.e., low-locality). With ancilla variables, even an extremely simple encoding, such as the turn encoding, can be used to construct a complicated energy function. While some of the particular tricks employed to optimize the efficiency of this mapping, such as introducing the backward penalty, are specific to lattice heteropolymers, the general logic behind these tricks is much more universal.

The turn circuit mapping is the most compact of all three mappings. The extremely efficient use of variables (qubits) makes it ideal for benchmark experiments on quantum devices that can handle many-body couplings. Moreover, the turn circuit method demonstrates how one can construct an elaborate energy function by utilizing logic circuits to put together a high-local fitness function of arbitrary complexity without ancilla variables. While different problems may involve different circuits, the underlying strategy is very broadly applicable.

The diamond encoding illustrates a strategy for producing an extremely underconstrained optimization problem. Furthermore, this method demonstrates that even fairly complex energy functions can be represented as natively 2-local functions if one is willing to sacrifice efficiency. Many quantum devices can only couple bits pairwise; thus, this is a very important quality of the diamond encoding. Finally, if one uses another, more efficient encoding, we explain how reductions can be used to replace high-local terms with 2-local terms in an optimally efficient fashion but at the cost of needing very high coupler resolution. The relatively few constraints in the diamond encoding make it a natural choice for exact or heuristic ILP and W-SAT solvers.

These three strategies elucidate many of the concepts that we find important when producing problems suitable for the D-Wave device utilized in Ref. [36]. Accordingly, as quantum information science continues to develop, we hope that

the methods discussed in this chapter will be useful to scientists wishing to leverage similar technology for the solution of discrete optimization problems.

Acknowledgments

The authors thank Joseph Goodknight for editing this chapter. This research was sponsored by United States Department of Defense. The views and conclusions contained in this chapter are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the U.S. Government.

REFERENCES

1. A. Sali, E. Shakhnovich, and M. Karplus, *Nature* **369**(6477), 248–251 (1994).
2. V. S. Pande, *Phys. Rev. Lett.* **105**(19), 198101 (2010).
3. K. A. Dill, S. B. Ozkan, M. S. Shell, and T. R. Weikl, *Ann. Rev. Biophys.* **37**(1), 289–316 (2008).
4. L. Mirny and E. Shakhnovich, *Ann. Rev. Biophys. Biomol. Struct.* **30**, 361–396 (2001).
5. V. Pande, A. Grosberg, and T. Tanaka, *Rev. Mod. Phys.* **72**(1), 259–314 (2000).
6. K. A. Dill, *Polym. Prepr. Am. Chem. Soc. Div. Polym. Chem.* **36**(1), 635 (1995).
7. M. Gruebele and P. G. Wolynes, *Nat. Struct. Biol.* **5**(8), 662–665 (1998).
8. E. I. Shakhnovich, *Phys. Rev. Lett.* **72**(24), 3907–3910 (1994).
9. W. E. Hart, and S. Istrail, *J. Comput. Biol.* **4**(1), 1–22 (1997).
10. K. F. Lau and K. A. Dill, *Macromolecules* **22**(10), 3986–3997 (1989).
11. B. Berger and T. Leighton, *J. Comput. Biol.* **5**(1), 27–40 (1998).
12. P. Crescenzi, D. Goldman, C. Papadimitriou, A. Piccolboni, and M. Yannakakis, *J. Comput. Biol.* **5**(3), 423–465 (1998).
13. R. D. Schram, G. T. Barkema, and R. H. Bisseling, *J. Stat. Mech.* P06019 (2011).
14. P. Hansen and B. Jaumard, *Computing* **7**(656–666), 279–303 (1990).
15. P. Hansen, B. Jaumard, and M. P. De Aragao, *Eur. J. Oper. Res.* **108**(3), 671–683 (1998).
16. M. Soos, K. Nohl, and C. Castelluccia, in, *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing, Lecture Notes in Computer Science*, Vol. 5584, 2009, pp. 244–257.
17. J. A. Marques-Silva and K. A. Sakallah, in *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing* Lisbon, Portugal, May 28–31, 2007, *Lecture Notes in Computer Science*, Vol. 4501, J. A. Marques-Silva and K. A. Sakallah, eds., Springer, 2007, p. 384.
18. R. Hemmecke, M. Köppe, J. Lee, and R. Weismantel, in *50 Years of Integer Programming 1958–2008*, Springer, 2009, p. 57.
19. S. Even, A. Itai, and A. Shamir, *SIAM J. Comput.* **5**(4), 691–703 (1976).
20. K. Yue and K. A. Dill, *Proc. Nat. Acad. Sci. USA* **92**(1), 146–150 (1995).
21. A. D. Ullah, and K. Steinhöfel, *BMC Bioinform.* **11**(Suppl. 1), S39 (2010).
22. A. Dal Palù, A. Dovier, and F. Fogolari, *BMC Bioinform.* **5**(1), 186 (2004).
23. L. Krippahl and P. Barahona, in *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP99)* Vol. 1713, 1999, pp. 289–302.

24. R. Backofen, *Energy* **3**, 389–400 (1998).
25. R. Backofen and S. Will, *Constraints* **11**(1), 5–30 (2006).
26. A. B. Finnila, M. A. Gomez, C. Sebenik, C. Stenson, and J. D. Doll, *Chem. Phys. Lett.* **2614**(March), 343–348 (1994).
27. T. Kadowaki and H. Nishimori, *Phys. Rev. E* **58**(5), 15 (1998).
28. G. E. Santoro, and E. Tosatti, *J. Phys. A* **39**(36), R393–R431 (2006).
29. A. Das, and B. K. Chakrabarti, *Rev. Mod. Phys.* **80**(3), 22 (2008).
30. A. Apolloni, N. Cesa-Bianchi, and D. De Falco, in *Stochastic Processes, Physics and Geometry, Proceedings of the Ascona-Locarno Conference*, 1988, pp. 97–111.
31. B. Apolloni, C. Carvalho, and D. De Falco, *Stoc. Proc. Appl.* **33**, 233–244 (1989).
32. V. N. Smelyanskiy, E. G. Rieffel, S. I. Knysh, C. P. Williams, M. W. Johnson, M. C. Thom, W. G. Macready, and K. L. Pudenz, arXiv:quant-ph/1204.2821 (2012).
33. E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, arXiv:quant-ph/0001106 (2000).
34. E. Farhi, J. Goldstone, S. Gutmann, J. Lapan, A. Lundgren, and D. Preda, *Science* **292**(5516), 472–475 (2001).
35. A. Perdomo, C. Truncik, I. Tubert-Brohman, G. Rose, and A. Aspuru-Guzik, *Phys. Rev. A* **78**(1), 35 (2008).
36. A. Perdomo-Ortiz, N. Dickson, M. Drew-Brook, G. Rose, and A. Aspuru-Guzik, *Nat. Sci. Rep.* **2**, 571 (2012).
37. S. Miyazawa and R. L. Jernigan, *J. Mol. Biol.* **256**(3), 623–644 (1996).
38. D. Baker, *Nature* **405**(6782), 39–42 (2000).
39. M. T. Oakley, D. J. Wales, and R. L. Johnston, *J. Phys. Chem. B* **115**(August), 11525–11529 (2011).
40. E. I. Shakhnovich, *Fold. Des.* **1**(3), R50–R54 (1996).
41. K. A. Dill, *Curr. Opin. Struct. Biol.* **3**(1), 99–103 (1993).
42. J.-E. Shea, J. N. Onuchic, and C. L. Brooks, *J. Chem. Phys.* **113**(17), 7663–7671 (2000).
43. C. J. Camacho, *Phys. Rev. Lett.* **77**(11), 4 (1995).
44. H. Nymeyer, A. E. García, and J. N. Onuchic, *Proc. Nat. Acad. Sci. USA* **95**(11), 5921–5928 (1998).
45. S. A. Cook, in *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC 71)*, 1971, pp. 151–158.
46. Z. Xing and W. Zhang, *Artif. Intell.* **164**(1–2), 47–80 (2005).
47. E. Boros and P. L. Hammer, *Discrete Appl. Math.* **123**(1–3), 155–225 (2002).
48. D. Boughaci and H. Drias, in *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC 04)*, 2004, p. 35.
49. D. Pankratov and A. Borodin, in *Theory and Applications of Satisfiability Testing (SAT 2010), Lecture Notes in Computer Science*, Vol. 6175, O. Strichman and S. Szeider eds., Springer, 2010, pp. 223–236.
50. N. Eén, and N. Sörensson, *J. Satisfiability Boolean Model. Comput.* **2**(3–4), 1–26 (2006).
51. A. Choi, T. Standley, and A. Darwiche, in *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming*, 2009, pp. 211–225.
52. J. Larrosa, F. Heras, and S. De Givry, *Artif. Intell.* **172**(2–3), 204–233 (2006).
53. IBM, *IBM ILOG CPLEX V12.1: User's Manual for CPLEX*, 2009.

54. F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah, in *Proceedings of the ACM/IEEE International Conference on Computer-Aided Design (ICCAD)*, 2002, pp. 450–457.
55. K. A. Sakallah, *Electr. Eng.* **2**, 155–179 (2006).
56. H. M. Sheini, and K. A. Sakallah, in *Proceedings of the Design, Automation and Test in Europe*, 2005, pp. 684–685.
57. P. Manolios and V. Papavasileiou, in *Proceedings of the Formal Methods in Computer-Aided Design (FMCAD'11)*, 2011.
58. F. Kahl and P. Strandmark, in *IEEE International Conference on Computer Vision*, 2011.
59. J. D. Biamonte, *Phys. Rev. A* **77**(5), 1–8 (2008).
60. R. Babbush, B. O’Gorman, and A. A. Aspuru-Guzik, *Ann. Phys.* **525**(10–11), 877–888 (2013).
61. K. Chandrasekaran, R. Karp, E. Moreno-Centeno, and S. Vempala, in *ACM-SIAM Symposium on Discrete Algorithms*, 2011, pp. 614–629.
62. V. Chvatal, *Math. Oper. Res.* **4**(3), 233–235 (1979).
63. S. Laue, in, *Annual Symposium on Theoretical Aspects of Computer Science (STACS 2008)* 2008, pp. 479–490.
64. L. S. L. Shi and X. C. X. Cai, in *Proceedings of the Third Joint International Conference on Computational Science and Optimization*, 2010, pp. 64–67.
65. H. Neven, G. Rose, and W. G. Macready, *Computer* **3**, 7 (2008).
66. V. S. Denchev, N. Ding, S. V. N. Vishwanathan, and H. Neven, in *Proceedings of the International Conference on Machine Learning*, 2012, p. 1205.1148.
67. M. W. Johnson, M. H. S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, E. M. Chapple, C. Enderud, J. P. Hilton, K. Karimi, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, C. Rich, M. C. Thom, E. Tolkacheva, C. J. S. Truncik, S. Uchaikin, J. Wang, B. Wilson, and G. Rose, *Nature* **473**(7346), 194–198 (2011).
68. R. Harris, M. W. Johnson, T. Lanting, A. J. Berkley, J. Johansson, P. Bunyk, E. Tolkacheva, E. Ladizinsky, N. Ladizinsky, T. Oh, F. Cioata, I. Perminov, P. Spear, C. Enderud, C. Rich, S. Uchaikin, M. C. Thom, E. M. Chapple, J. Wang, B. Wilson, M. H. S. Amin, N. Dickson, K. Karimi, B. Macready, C. J. S. Truncik, and G. Rose, *Phys. Rev. B* **82**(2), 16 (2010).