Construction of Energy Functions for Lattice Heteropolymer Models: Efficient Encodings for Constraint Satisfaction Programming and Quantum Annealing

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-Guzik[1]†

November 16, 2012

Abstract

Optimization problems associated with the interaction of linked particles are at the heart of polymer science, protein folding and other important problems in the physical sciences. In this review we explain how to recast these problems as constraint satisfaction problems such as linear programming, maximum satisfiability, and pseudo-boolean optimization. By encoding problems this way, one can leverage substantial insight and powerful solvers from the computer science community which studies constraint programming for diverse applications such as logistics, scheduling, artificial intelligence, and circuit design. We demonstrate how to constrain and embed lattice heteropolymer problems using several strategies. Each strikes a unique balance between number of constraints, complexity of constraints, and number of variables. In addition, each strategy has distinct advantages and disadvantages depending on problem size and available resources. Finally, we show how to reduce the locality of couplings in these energy functions so they can be realized as Hamiltonians on existing adiabatic quantum annealing machines.

[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

*Corresponding author. Electronic address: babbush@fas.harvard.edu.
†Corresponding author. Electronic address: alan@aspuru.com.

Contents

1 Introduction

1.1 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 (Sali, Shakhnovich & Karplus 1994, Pande 2010, Dill, Ozkan, Shell & Weikl 2008, Mirny & Shakhnovich 2001, Pande, Grosberg & Tanaka 2000, Dill 1995, Gruebele & Wolynes 1998, Shakhnovich 1994). Among other examples of biologically relevant polymers, DNA and RNA chains also fold into complicated structures which 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 (Hart & Istrail 1997). Even the simplest model for lattice folding (Lau & Dill 1989) was proved to be an NP-hard problem (Berger & Leighton 1998, Crescenzi, Goldman, Papadimitriou, Piccolboni & Yannakakis 1998). 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 (Schram, Barkema & Bisseling 2011).

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 which 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 (Hansen & Jaumard 1990, Hansen, Jaumard & De Aragao 1998, Soos, Nohl & Castelluccia 2009). 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 (Marques-Silva & Sakallah 2007). Another widely studied constraint satisfaction problem is linear programming which has many applications including logistics scheduling, operations research, company management, and economic planning (Hemmecke, Köppe, Lee & Weismantel 2009). Some applications of linear programming, i.e. multi-commodity flow problems, are considered important enough that entire fields of research exist to develop specialized algorithms for their solution (Even, Itai & Shamir 1976).

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 since (Yue & Dill 1995). Other relevant papers include (Ullah & Steinhöfel 2010, Dal Palù, Dovier & Fogolari 2004, Krippahl & Barahona 1999, Backofen 1998, Backofen & Will 2006).

Another intriguing option is to study these problems using a computer which 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 paradigms is quantum annealing (Finnila, Gomez, Sebenik, Stenson & Doll 1994, Kadowaki & Nishimori 1998, Santoro & Tosatti 2006, Das & Chakrabarti 2008, Apolloni, Cesa-Bianchi & De Falco 1988, Apolloni, Carvalho & De Falco 1989, Smelyanskiy, Rieffel, Knysh, Williams, Johnson, Thom, Macready & Pudenz 2012), also known as adiabatic quantum computation (Farhi, Goldstone, Gutmann & Sipser 2000, Farhi, Goldstone, Gutmann, Lapan, Lundgren & Preda 2001, Kadowaki & Nishimori 1998). In quantum annealing, the presence of quantum fluctuations (tunneling) allows the system to efficiently traverse potential energy barriers which have a tendency to trap classical optimizations algorithms.

Motivated by the experimental realization of studying biologically interesting optimization problems with quantum computation, in this contribution 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 (Perdomo, Truncik, Tubert-Brohman, Rose & Aspuru-Guzik 2008), 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 (Perdomo-Ortiz, Dickson, Drew-Brook, Rose & Aspuru-Guzik 2012) where we employed up to 81 superconducting qubits to solve a 6 amino-acid problem in the Miyazawa-Jernigan (MJ) model (Miyazawa & Jernigan 1996).

The goal of this review is to explain the mapping used in (Perdomo-Ortiz et al. 2012), 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 which can be solved by using different types of technology and algorithms. While the focus of this paper 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 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 which 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. 1.1.

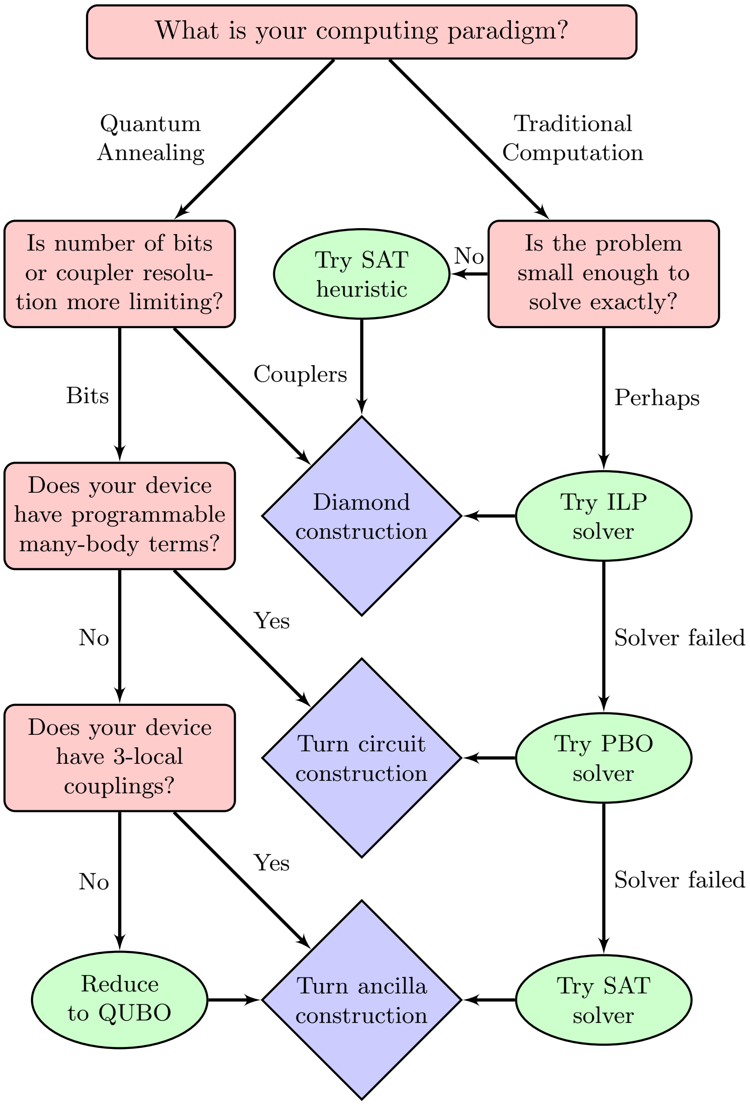

What is your computing paradigm?
Quantum Annealing
Traditional Computation
Is number of bits or coupler resolution more limiting?
Try SAT heuristic
No
Is the problem small enough to solve exactly?
Bits
Couplers
Perhaps
Does your device have programmable many-body terms?
Diamond construction
Try ILP solver
No
Yes
Solver failed
Does your device have 3-local couplings?
Turn circuit construction
Try PBO solver
No
Yes
Solver failed
Reduce to QUBO
Turn ancilla construction
Try SAT solver

Figure 1: Flow chart describing how one might choose between the three problem encodings discussed in this review based 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 which 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 pseudoboolean optimization (PBO) solvers. The "turn ancilla" encoding represents a balance of these benifits as it is relatively efficient and can easily collapse to 2-local without extremely high term coefficients.

1.2 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 $\boldsymbol{q} \equiv q_1 q_2 \cdots q_n$. 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 which 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 $E(\boldsymbol{q}) = E(q_1, q_2, \cdots, q_n)$ which takes $\boldsymbol{q}$ as input and correctly reproduces the relative energies in the low energy subspace of the original problem so that the optimal solution to $E(\boldsymbol{q})$ 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 $\boldsymbol{q}$. 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 technique to construct the energy function for particles interacting in a lattice (Perdomo et al. 2008). The purpose of this contribution is to introduce the reader to several different types of mappings which 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), integer-linear programming (ILP), or quadratic unconstrained binary optimization (QUBO). When cast as a W-SAT problem one can take advantage of both heuristic and exact W-SAT solvers which 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 (Perdomo-Ortiz et al. 2012).

2 The "Turn" Encoding of Self-Avoiding Walks

2.1 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 which encode the possible folds by explicitly encoding the grid location of each amino acid. The square lattice spatial representation discussed in (Perdomo et al. 2008) 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 jth amino acids in the lattice, we encode the jth turn taken by the $j+1$ 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 translationally and rotationally invariant 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 two 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, e.g. triangular. The convention or "compass" used in this paper is presented in the upper-left part of Fig. 2. Furthermore, we can fix the first three bits to obtain only solutions which are rotationally invariant. Under this convention, the bit-string $\boldsymbol{q}$ is written as,

$$\boldsymbol{q} = 01\ \underbrace{0q_1}_{turn2}\ \underbrace{q_2q_3}_{turn3} \cdots \underbrace{q_{2(N-1)-1}q_{2(N-1)}}_{turn(N-1)} \tag{1}$$

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 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 (Perdomo et al. 2008). To clearly demonstrate how this mapping works, an example of the turn encoding for a short SAW is shown below in Fig. 2.

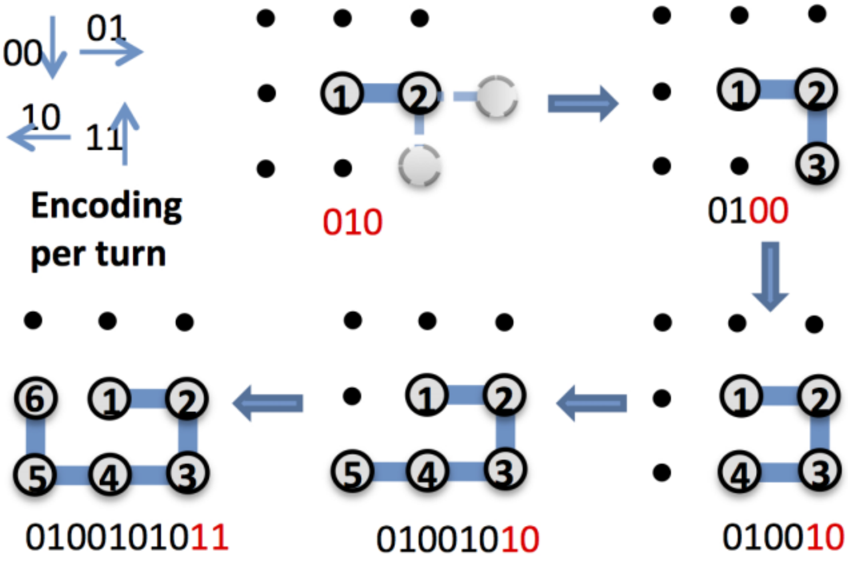


Figure 2: Step-by-step construction of the binary representation of a particular six unit lattice protein in the turn encoding. Two qubits per bond are needed and the turn "compass" (bond directions) are denoted as "00" (downwards), "01" (rightwards), "10" (left), and "11" (upwards). This image has been reproduced from (Perdomo-Ortiz et al. 2012) with permission from the authors.

2.2 "Turn ancilla" construction of $E(\boldsymbol{q})$

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

$$E(\boldsymbol{q}) = E_{back}\,(\boldsymbol{q}) + E_{overlap}(\boldsymbol{q}) + E_{pair}(\boldsymbol{q}), \tag{2}$$

involves a series of intermediate steps which we outline next.

2.2.1 Construction of $E_{back}(\boldsymbol{q})$

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_1q_2) went right, left, up, or down respectively.

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

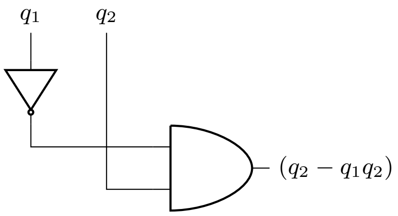


Figure 4: A logical circuit representing "left" consisting of a NOT gate after the second bit and an AND gate. Evaluates to TRUE if and only if $q_1, q_2 = 1, 0$.

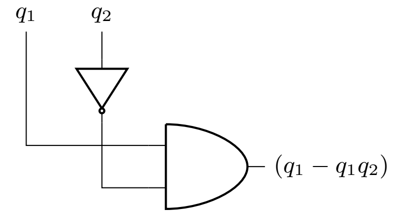


Figure 5: A logical circuit representing "up". Only TRUE if $q_1, q_2 = 1, 1$.

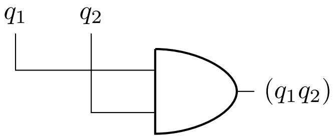
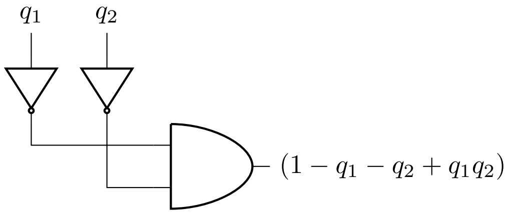


Figure 6: A logical circuit representing "down". Only TRUE if $q_1, q_2 = 0, 0$.



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 jth turn,

$$\begin{aligned} d^j_{x+} &= q_{2j}(1 - q_{2j-1}) = q_{2j} - q_{2j}q_{2j-1} && (3) \\ d^j_{x-} &= (1 - q_{2j})q_{2j-1} = q_{2j-1} - q_{2j}q_{2j-1} && (4) \\ d^j_{y+} &= q_{2j}q_{2i-1} && (5) \\ d^j_{y-} &= (1 - q_{2j})(1 - q_{2j-1}) = 1 - q_{2j} - q_{2j-1} + q_{2j}q_{2j-1}, && (6) \end{aligned}$$

which evaluate to TRUE if and only if the jth turn is to be right, left, up or down respectively. Having defined these circuits we can construct a more complicated circuit which takes two turns (the 4 bits $q_iq_{i+1}q_{i+2}q_{i+3}$) as input and returns TRUE if and only if the second turn went backwards, i.e. $\left(d^j_{x+} \wedge d^{j+1}_{x-}\right) \vee \left(d^j_{x-} \wedge d^{j+1}_{x+}\right) \vee \left(d^j_{y+} \wedge d^{j+1}_{y-}\right) \vee \left(d^j_{y-} \wedge d^{j+1}_{y+}\right)$. An example of these conjunctions, $\left(d^j_{x+} \wedge d^{j+1}_{x-}\right)$ is shown in Fig. 7.

The other three conjunctions are also trivially constructed by combining the appropriate circuits using AND gates which 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,

$$\begin{aligned} \left(d^j_{x+} \wedge d^{j+1}_{x-}\right) &= q_{i+1}q_{i+2} - q_iq_{i+1}q_{i+2} - q_{i+1}q_{i+2}q_{i+3} + q_iq_{i+1}q_{i+2}q_{i+3} && (7) \\ \left(d^j_{x-} \wedge d^{j+1}_{x+}\right) &= q_iq_{i+3} - q_iq_{i+1}q_{i+3} - q_iq_{i+2}q_{i+3} + q_iq_{i+1}q_{i+2}q_{i+3} && (8) \\ \left(d^j_{y+} \wedge d^{j+1}_{y-}\right) &= q_iq_{i+1} - q_iq_{i+1}q_{i+2} - q_iq_{i+1}q_{i+3} + q_iq_{i+1}q_{i+2}q_{i+3} && (9) \\ \left(d^j_{y-} \wedge d^{j+1}_{y+}\right) &= q_{i+2}q_{i+3} - q_iq_{i+2}q_{i+3} - q_{i+1}q_{i+2}q_{i+3} + q_iq_{i+1}q_{i+2}q_{i+3}. && (10) \end{aligned}$$

It might seem logical to finish this circuit by combining all four backwards overlap circuits with OR gates; however, this is not an advisable strategy as it

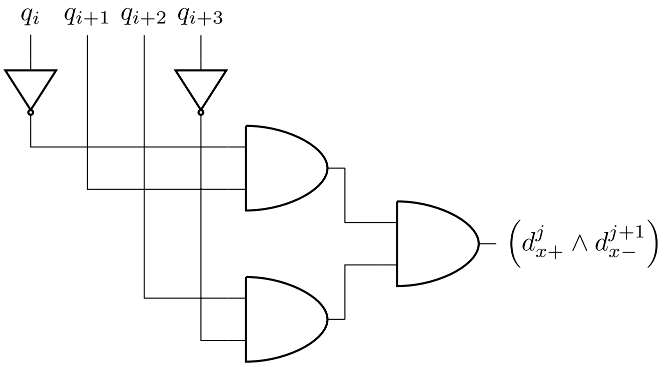


Figure 7: A logical circuit which returns TRUE if and only if $\left(d_{x+}^{j} \wedge d_{x-}^{j+1}\right)$, i.e. the turn sequence $q_i q_{i+1} q_{i+2} q_{i+3} = 0110$, meaning it went right and then left.

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,

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

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

$$E_{back}\left(q_i q_{i+1} q_{i+2} q_{i+3}\right) = \lambda_{overlap}\left(2q_i q_{i+2} - q_i - q_{i+2}\right)\left(2q_{i+1} q_{i+3} - q_{i+1} - q_{i+3}\right). \tag{12}$$

To construct the entire $E_{back}\left(\boldsymbol{q}\right)$ 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 $E_{back}(\boldsymbol{q})$ as,

$$\begin{aligned} E_{back}(\boldsymbol{q}) &= \lambda_{back}\left(q_1 q_2 + q_2 q_3 - 2q_1 q_2 q_3\right) \\ &+ \lambda_{back} \textstyle\sum_{i=2}^{2N-8}\left(2q_i q_{i+2} - q_i - q_{i+2}\right)\left(2q_{i+1} q_{i+3} - q_{i+1} - q_{i+3}\right). \end{aligned} \tag{13}$$

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 review.

It is important to point out that while the decision to use a separate $E_{back}(\boldsymbol{q})$ instead of a more general $E_{overlap}\left(\boldsymbol{q}\right)$ 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 Sec. 6.

2.2.2 Construction of $E_{overlap}\left(\boldsymbol{q}\right)$ with ancilla variables

The overlap energy function $E_{overlap}(\boldsymbol{q})$ 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 which 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,

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

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

where the position of the nth amino acid in the sequence is a function of the proceeding $n-1$ 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 which will return the square of the grid distance between any two amino acids (denoted i and j):

$$g_{ij} = \left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2. \tag{16}$$

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. **??** (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 which 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 $E_{overlap}\left(\boldsymbol{q}\right)$ we need a function which takes g_{ij} as input and returns a penalty if and only if $g_{ij} = 0$. First, we note the bounds on g_{ij},

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

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,

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

This implies that,

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

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

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

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

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

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,

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

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

$$\sum_{i=1}^{N-4} \sum_{j=i+4}^{N} \mu_{ij}. \tag{23}$$

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

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

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

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

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

$$0 \leq \alpha_{ij} \leq 2^{\mu_{ij}} - 1. \tag{26}$$

We should adjust Eq. 19 and Eq. 20 so that,

$$\forall\, g_{ij} \geq 1\, \exists\, \alpha_{ij} : 2^{\mu_{ij}} - g_{ij} - \alpha_{ij} = 0. \tag{27}$$

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

$$2^{\mu_{ij}} - g_{ij} - \alpha_{ij} \geq 1\, \forall\, \alpha_{ij}. \tag{28}$$

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 $E_{overlap}\left(\boldsymbol{q}\right) = 0$ we square the expression to obtain the following formula,

$$\gamma_{ij} = \lambda_{overlap}\left[2^{\mu_{ij}} - g_{ij} - \alpha_{ij}\right]^2. \tag{29}$$

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 $\lambda_{overlap}$ a sufficiently large penalty that all low energy solutions must have no overlaps, i.e. $g_{ij} \geq 1$ for all ij, and $\alpha_{ij} = 2^{\mu_{ij}} - g_{ij}$. Finally we can write the final expression,

$$E_{overlap}\left(\boldsymbol{q}\right) = \sum_{i=1}^{N-4} \sum_{j=i+4}^{N} \left[(1 + i - j) \bmod 2\right] \gamma_{ij}. \tag{30}$$

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 apart are prevented by $E_{back}\left(\boldsymbol{q}\right)$, 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 $E_{back}\left(\boldsymbol{q}\right)$ was so that we used fewer ancillary bits in this step.

2.2.3 Construction of $E_{pair}(\boldsymbol{q})$ with ancilla variables

Finally, we need to construct the pairwise interaction energy function. To do this we need to make an interaction matrix, J, which contains all of the pairwise interactions which lower the energy when two amino acids are adjacent on the lattice (thus all elements of 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 which 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 where as in the Miyazawa-Jernigan 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 which is only "on" without incurring an energy penalty if two amino acids are interacting (that is, if $g_{ij} = 1$). We can now write the pairwise interaction term:

$$\varphi_{ij} = \omega_{ij} J_{ij} \left(2 - g_{ij}\right) \tag{31}$$

This simple function does everything we need to write the pair function. Because $E_{overlap}\left(\boldsymbol{q}\right)$ ensures that $g_{ij} \geq 1$, we see that φ_{ij} is only positive if both J_{ij} and ω_{ij} are non-zero 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 $E_{pair}(\boldsymbol{q})$ is,

$$E_{pair}(\boldsymbol{q}) = \sum_{i=1}^{N-1} \sum_{j=i+3}^{N} \omega_{ij} J_{ij} \left(2 - g_{ij}\right). \tag{32}$$

2.3 "Turn circuit" construction of $E(\boldsymbol{q})$

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 which provides optimal efficiency at the cost of introducing high ordered many-body terms. The turn circuit construction (along with reductions explained in Sec. 6) was used to encode problems into a quantum annealing machine in (Perdomo-Ortiz et al. 2012).

2.3.1 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, i.e. the total number of turns in that direction. This process is most straightforwardly described using a circuit model. Consider, a single Half-Adder gate (HA) consisting of an AND and an XOR gate, as shown in Fig. 8. The output of a Half-Adder can be interpreted as the two-bit sum of its two input bits. Accordingly, if we wanted to add three bits we could add two of them, and then add the resultant two-bit number to the third bit, as shown in Figure 9.

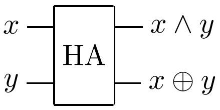


Figure 8: The Half-Adder gate sums two bits.

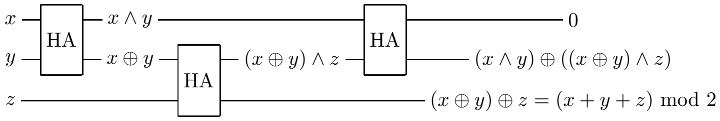

x
y
z
HA
$x \wedge y$
$x \oplus y$
HA
$(x \oplus y) \wedge z$
HA
0
$(x \wedge y) \oplus ((x \oplus y) \wedge z)$
$(x \oplus y) \oplus z = (x + y + z) \bmod 2$

Figure 9: A circuit to sum three bits.

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 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.

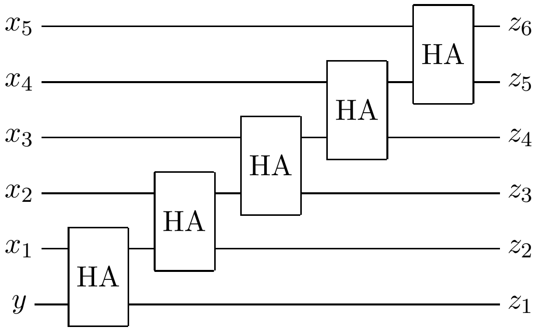


Figure 10: Circuit for the addition of a single bit y to the 5-bit $x = x_5x_4x_3x_2x_1$ to form the 6-bit sum $x + y = z_6z_5z_4z_3z_2z_1$.

Thus, given an arbitrary number of bits we can find their sum, in binary, by successively combining the strategies shown in Fig. 11, i.e. 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 three bit number which 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.

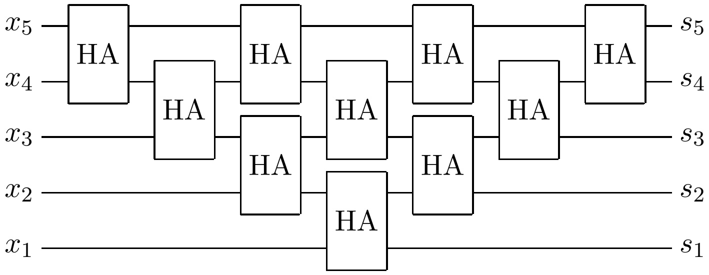


Figure 11: The circuit for the sum, $s_1s_2s_3s_4s_5$, of 5 bits $x_1 + x_2 + x_3 + x_4 + x_5$.

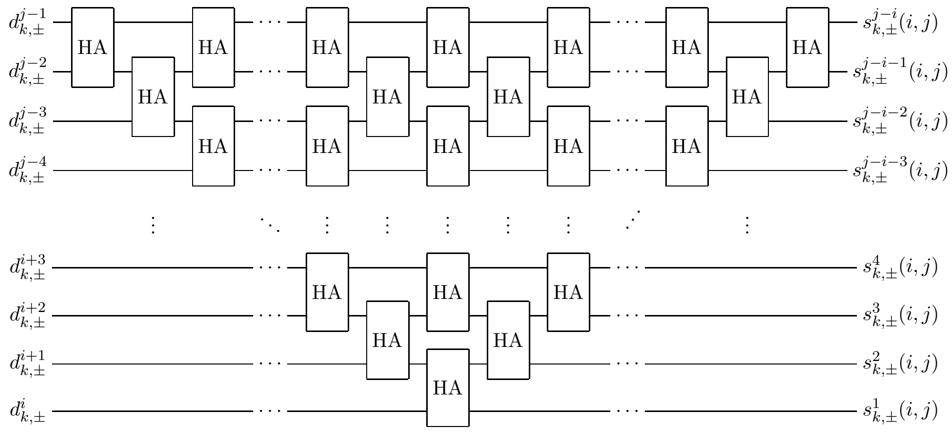


Figure 12: The circuit for the number $s_{k\pm}(i,j)$ of turns between amino acids i and j in the $\pm k$ direction.

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

$$s^r_{k\pm}(i,j) = r^{\text{th}} \text{ digit of } \sum_{p=i}^{j-1} d^p_{k\pm}. \tag{33}$$

This sum tells us how many turns our protein has taken in the $k\pm$ direction between any two amino acids. For instance, $s^1_{x-}(3,9)$ would tell us the value of the 1st binary digit of an integer representing the number of times that the protein turned in the negative x direction (aka 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 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.

2.3.2 Construction of $E_{overlap}(\boldsymbol{q})$ 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, $s^{j-i}_{k+} \cdots s^1_{k+}$ and $s^{j-i}_{k-} \cdots s^1_{k-}$, are the same. As discussed above, since only the first $\lceil \log_2(j-i) \rceil$ digits of $s_{k\pm}$ are non-zero, the overlap penalty function for amino acids i,j is

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

where

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

is the exclusive NOR function which evaluates to TRUE if and only if the two 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

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

2.3.3 Construction of $E_{pair}(\boldsymbol{q})$ 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 one. The construction of the equality condition is the same as in as for the overlap function; to construct the latter condition, consider the set of 4 bit numbers, as shown in Figure 13.

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.

Note that when the first of two sequential binary is even, the Hamming distance between those bit-strings are the same except for the least significant bit, e.g. 0000 and 0001, 1000 and 1001, 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, i.e.

$$\begin{array}{rr} & 00000000000\cdots01 \\ + & **\cdots**011\cdots11 \\ \hline & **\cdots**100\cdots00 \end{array}, \tag{37}$$

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

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 which 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, $a_k(i,j)$:

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

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

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

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

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

3 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 which is natively 2-local (without any reductions) and which has a very sparse QUBO (quadratic unconstrained binary optimization) matrix. Despite the fact that the diamond encoding requires no ancillary bits whatsoever, the encoding is still less bit-wise 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.

3.1 Embedding physical structure

The diamond encoding can be thought of as a more sophisticated and subtle version of the “spatial” encoding used in (Perdomo et al. 2008). 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 which 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.

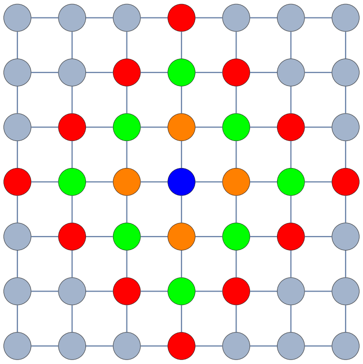


Figure 14: A “map” of the diamond encoding in 2D. If the first amino acid is fixed to the blue lattice point in the center then the second amino acid must be on an orange lattice point, the third must be on a green lattice point and the fourth must be on either an orange or red lattice point.

Fig. 14 illustrates what the “diamond” of valid lattice sites looks like for the first 4 amino acids in a SAW. In the diamond encoding each bit refers to a specific lattice site which 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, the third may occupy 8 and the fourth may occupy 18. Accordingly, we need this many bits for each amino acid.

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

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 which 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 (Baker 2000, Oakley, Wales & Johnston 2011, Shakhnovich 1996). 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 (Dill 1993, Shea, Onuchic & Brooks 2000, Camacho 1995, Nymeyer, García & Onuchic 1998). Therefore, for any interesting problem its 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 to us 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 cubicly 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.

3.2 Natively 2-local $E(\boldsymbol{q})$

The major advantages of the diamond encoding become evident as soon as one starts to construct $E(\boldsymbol{q})$. 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, $E_{one}\left(\boldsymbol{q}\right)$ 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 hard-code a primary structure constraint so we will need a term, $E_{connect}\left(\boldsymbol{q}\right)$ to guarantee that each sequential amino acid is adjacent. Like the turn encoding the diamond encoding will also require $E_{overlap}\left(\boldsymbol{q}\right)$ and $E_{pair}\left(\boldsymbol{q}\right)$ terms. The overall energy function looks like,

$$E\left(\boldsymbol{q}\right) = E_{one}\left(\boldsymbol{q}\right) + E_{connect}\left(\boldsymbol{q}\right) + E_{overlap}\left(\boldsymbol{q}\right) + E_{pair}\left(\boldsymbol{q}\right). \tag{42}$$

3.2.1 Construction of $E_{one}\left(\boldsymbol{q}\right)$

Each amino acid is encoded by flipping a bit in the part of the total bit-string sequence which 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 the case for low energy solutions 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 $\boldsymbol{q}^k$ is the binary vector which represents the kth amino acid and n_k represents the length of this vector then we can write,

$$E_{one}\left(\boldsymbol{q}\right) = \lambda_{one} \sum_{k=2}^{N} \sum_{i=1}^{n_k - 1} \sum_{j>i}^{n_k} q_i^k q_j^k. \tag{43}$$

λ_{one} in Eq. 43 yields terms which 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 $E_{connect}\left(\boldsymbol{q}\right)$ 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.

3.2.2 Construction of $E_{connect}\left(\boldsymbol{q}\right)$

To form $E_{connect}\left(\boldsymbol{q}\right)$ we take a very similar approach to how we formed $E_{one}\left(\boldsymbol{q}\right)$. To guarantee that the low energy solution space contains only amino acids chains which 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,

$$E_{connect}\left(\boldsymbol{q}\right) = N - 2 - \lambda_{connect} \sum_{k=2}^{N} \sum_{i=1}^{n_k - 1} \sum_{j=1}^{n_{k-1}} q_i^k q_j^{k-1}. \tag{44}$$

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 $\lambda_{connect} << \lambda_{one}$ so that the system cannot overcome the λ_{one} penalty by having multiple $\lambda_{connect}$ couplings. Finally we put the constant factor of $N-2$ into the equation to adjust the energy back to zero overall for valid solutions which contain $N-2$ connections.

3.2.3 Construction of $E_{overlap}(\boldsymbol{q})$

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 which have an even number of bonds between them to flip bits corresponding to the same lattice location. For instance, in Fig. 14 its 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,

$$E_{overlap}(\boldsymbol{q}) = \lambda_{overlap} \sum_{k=2}^{N-1} \sum_{h>k}^{N} \sum_{i=1}^{n_k} \left[(1+k-h) \bmod 2\right] q_i^k q_i^h. \tag{45}$$

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.

3.2.4 Construction of $E_{pair}(\boldsymbol{q})$

To form the pairwise interaction term we simply couple each bit to the possible adjacent lattice locations which 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:

$$E_{pair}(\boldsymbol{q}) = \sum_{k=2}^{N-1} \sum_{h=k+2}^{N} \sum_{<ij>} J_{hk} \left[(k-h) \bmod 2\right] q_i^k q_j^h \tag{46}$$

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

4 Pseudo-boolean Function to W-SAT

In order to take advantage of state-of-the-art satisfiability (SAT) solvers to optimize our pseudo-boolean function, it is necessary to map the problem to Weighted Maximum Satisfiability (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 which 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 (Cook 1971).

4.1 MAX-SAT and W-SAT

Maximum Satisfiability (MAX-SAT) is a more difficult version of the canonical SAT problem which 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 which satisfies all clauses (such a solution string may not even exist); rather, the goal is to find the solution string which satisfies the maximum number of clauses.

An extension of MAX-SAT known as Weighted Maximum Satisfiability (aka W-SAT) is what will be most relevant to us. In W-SAT each clause is given a positive integer valued “weight” which 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 (Xing & Zhang 2005, Boros & Hammer 2002). We can more succinctly state the problem as follows: given m number of clauses (y) each with a weight of w, minimize

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

The same approximation schemes and exact solver algorithms which work for MAX-SAT also work for W-SAT (Boughaci & Drias 2004, Pankratov & Borodin 2010). 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. Fig. 15 shows this clause as a logic circuit.

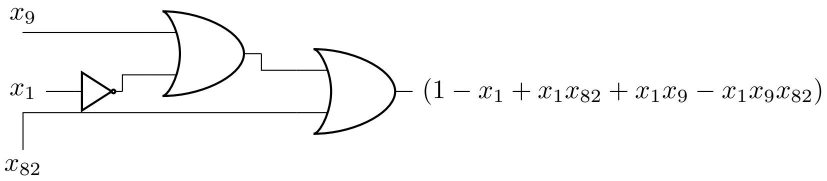


Figure 15: A logical circuit representation of the CNF clause: "9 −1 82"

4.2 Constructing WCNF clauses

To prepare the WCNF input file from a pseudo-boolean function one will need to write a short script which transforms each term in the pseudo-boolean function into a WCNF clause. There is more than one way to accomplish this transformation and we will only discuss one method here. For a more complete review of this topic, see (Eén & Sörensson 2006).

It will be very useful to think of CNF clauses as logical circuits which 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 which takes all variables as input). This circuit is illustrated in Fig. 16 for the case of a 5 variables clause.

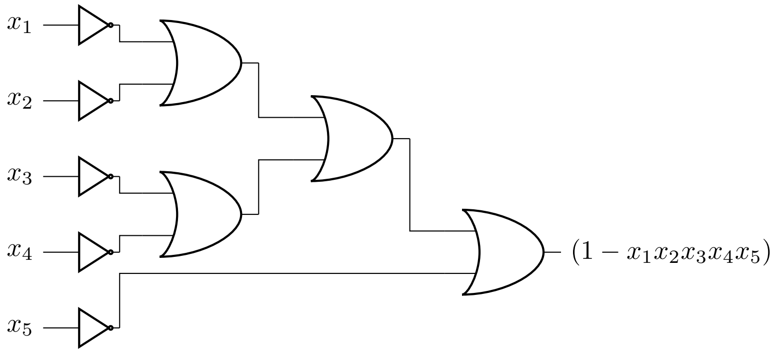


Figure 16: A logical circuit which shows that any pseudo-boolean term with positive weight is equivalent (up to a constant) to a CNF clause with each variable negated. The term produced here is negative because the weight is only added when the clause evaluates to FALSE.

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 get us anywhere - we turned a term into CNF only to introduce a new term into the pseudo-boolean which 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.

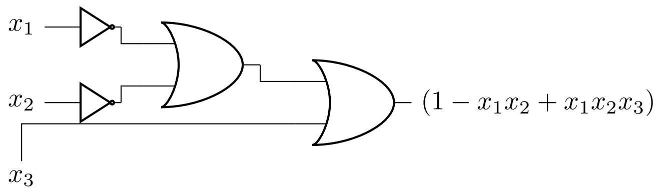


Figure 17: A logical circuit on three variables which gives a positive valued 3-local CNF term.

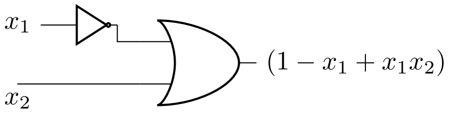


Figure 18: A logical circuit on three variables which gives a positive valued 2-local CNF term.

4.3 Solving SAT problems

While MAX-SAT is known to be NP-hard, there exist heuristic algorithms which 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 $\frac{k}{k+1}$, Tabu search achieves a ratio of at least $\frac{k+1}{k+2}$ and non-oblivious local search achieves an approximation ratio of $\frac{2^k-1}{2^k}$ 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 $\frac{21}{22} + \epsilon \approx 0.955 + \epsilon$ [9] in polynomial time (Pankratov & Borodin 2010, Choi, Standley & Darwiche 2009). Additionally, there are a great deal of exact MAX-SAT solvers which run in super-polynomial 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 (Marques-Silva & Sakallah 2007, Larrosa, Heras & De Givry 2006).

5 W-SAT to Integer-Linear Programming

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 (particular for logistics scheduling) has produced many extremely good exponential-time exact solvers and polynomial-time heuristic solvers (Xing & Zhang 2005). Pseudo-boolean optimization is an even more specific case of ILP sometimes known as 0-1 ILP where the integer variables are boolean (Boros & Hammer 2002). The mapping between W-SAT and ILP is very straightforward.

5.1 Mapping to ILP

In ILP, the goal is to minimize an objective function of integer-valued variables subject to a list of inequality constraints which must be satisfied. The inequality constraints come directly from the clauses in W-SAT. As described in Sec. 4.1, the logical clause from the WCNF clause "4000 9 -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 which 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 (Xing & Zhang 2005).

5.2 Solving ILP problems

Commercial logistic scheduling software such as IBM ILOG CPLEX Optimization Studio (aka CPLEX) is designed to solve in integer programming, linear programming, and mixed integer-linear programming problems on a very large scale (IBM 2009). Constraint satisfaction problems which 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 over-constrained (Aloul, Ramani, Markov & Sakallah 2002). On the other hand, for problems which are under-constrained 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 (Sakallah 2006, Sheini & Sakallah 2005, Manolios & Papavasileiou 2011).

6 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 (Pankratov & Borodin 2010, Kahl & Strandmark 2011, Xing & Zhang 2005). In fact, the degree of constraints determines the complexity class of certain constraint satisfaction problems; e.g. 2-SAT is proven to be in P whereas 3-SAT is in NP-Complete (Cook 1971). Clearly for instances such as this there can be no efficient method which 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 (Perdomo-Ortiz et al. 2012) 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 which 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 $E_{\wedge}\left(q_i, q_j, \tilde{q}_n; \delta_n\right)$ in Eq. 48 which 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 (Perdomo et al. 2008).

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

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

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

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 which we could use to reduce each 3-local term in Eq. 50.

$$\begin{pmatrix}
q_1 & q_2 & q_3 \\
q_1 & q_2 & q_4 \\
q_1 & q_3 & q_4 \\
q_2 & q_3 & q_4 \\
q_1 & q_2 & q_3 \\
q_1 & q_2 & q_5 \\
q_1 & q_3 & q_5 \\
q_2 & q_3 & q_5 \\
q_1 & q_4 & q_5 \\
q_2 & q_4 & q_5 \\
q_1 & q_2 & q_4 \\
q_3 & q_4 & q_5 \\
q_2 & q_3 & q_4 \\
q_1 & q_3 & q_6 \\
q_1 & q_3 & q_7 \\
q_2 & q_4 & q_7 \\
q_1 & q_5 & q_7 \\
q_3 & q_5 & q_7 \\
q_1 & q_3 & q_8 \\
q_2 & q_4 & q_8 \\
q_1 & q_5 & q_8 \\
q_3 & q_5 & q_8 \\
q_1 & q_3 & q_9 \\
q_2 & q_4 & q_9 \\
q_1 & q_5 & q_9 \\
q_3 & q_5 & q_9 \\
q_1 & q_3 & q_{10} \\
q_2 & q_4 & q_{10} \\
q_1 & q_5 & q_{10} \\
q_3 & q_5 & q_{10}
\end{pmatrix}
\iff
\begin{pmatrix}
q_1q_2 & q_1q_3 & q_2q_3 \\
q_1q_2 & q_1q_4 & q_2q_4 \\
q_1q_3 & q_1q_4 & q_3q_4 \\
q_2q_3 & q_2q_4 & q_3q_4 \\
q_1q_2 & q_1q_3 & q_2q_3 \\
q_1q_2 & q_1q_5 & q_2q_5 \\
q_1q_3 & q_1q_5 & q_3q_5 \\
q_2q_3 & q_2q_5 & q_3q_5 \\
q_1q_4 & q_1q_5 & q_4q_5 \\
q_2q_4 & q_2q_5 & q_4q_5 \\
q_1q_2 & q_1q_4 & q_2q_4 \\
q_3q_4 & q_3q_5 & q_4q_5 \\
q_2q_3 & q_2q_4 & q_3q_4 \\
q_1q_3 & q_1q_6 & q_3q_6 \\
q_1q_3 & q_1q_7 & q_3q_7 \\
q_2q_4 & q_2q_7 & q_4q_7 \\
q_1q_5 & q_1q_7 & q_5q_7 \\
q_3q_5 & q_3q_7 & q_5q_7 \\
q_1q_3 & q_1q_8 & q_3q_8 \\
q_2q_4 & q_2q_8 & q_4q_8 \\
q_1q_5 & q_1q_8 & q_5q_8 \\
q_3q_5 & q_3q_8 & q_5q_8 \\
q_1q_3 & q_1q_9 & q_3q_9 \\
q_2q_4 & q_2q_9 & q_4q_9 \\
q_1q_5 & q_1q_9 & q_5q_9 \\
q_3q_5 & q_3q_9 & q_5q_9 \\
q_1q_3 & q_1q_{10} & q_3q_{10} \\
q_2q_4 & q_2q_{10} & q_4q_{10} \\
q_1q_5 & q_1q_{10} & q_5q_{10} \\
q_3q_5 & q_3q_{10} & q_5q_{10}
\end{pmatrix} \tag{50}$$

Eq. 50 shows that there are 30, 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 $S_1S_2...S_{30}$. We can then make the following bipartite graph which connects the 3-local terms to the 2-local terms which collapse them.

Fig. 21 shows that we can now restate the problem in the following way: "choose the fewest number of 2-local terms (on the left) which covers all 3-local terms (on the right) with at least one edge." In general, this problem is isomorphic to the canonical "hitting set" problem which is equivalent to set cover, one

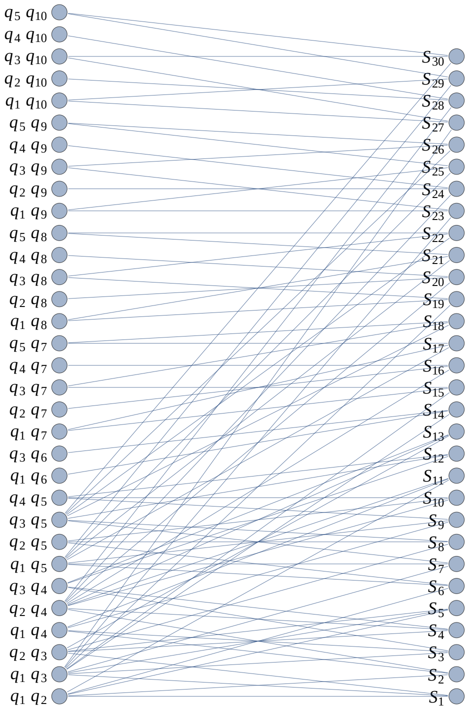


Figure 19: A bipartite graph connecting the 3-local terms (S_n) in Eq. 49 to the 2-local terms ($q_i q_j$) which collapse them.

of Karp's 21 NP-Complete Problems (Chandrasekaran, Karp, Moreno-Centeno & Vempala 2011, Chvatal 1979, Laue 2008). 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 1 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 (Shi & Cai 2010) for a very efficient algorithm which can exactly solve hitting cover in $O\left(1.23801^n\right)$.

7 Quantum Realization

A primary goal of this review 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 review 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, i.e., amino acid chains in "free space" [1]. As a final step we will reduce these Hamiltonians to a 2-local form specifically design for the Dwave One used in (Perdomo-Ortiz et al. 2012, Neven, Rose & Macready 2008, Denchev, Ding, Vishwanathan & Neven 2012, Johnson, Amin, Gildert, Lanting, Hamze, Dickson, Harris, Berkley, Johansson, Bunyk, Chapple, Enderud, Hilton, Karimi, Ladizinsky, Ladizinsky, Oh, Perminov, Rich, Thom, Tolkacheva, Truncik, Uchaikin, Wang, Wilson & Rose 2011). The final Hamiltonian we present is more efficient than that used in (Perdomo-Ortiz et al. 2012) as we have since realized several tricks to make the energy function more compact.

7.1 Previous experimental implementation

Throughout this review we have referred to an experimental implementation of quantum annealing to solve lattice heteropolymer problems in (Perdomo-Ortiz et al. 2012). The quantum hardware employed consists of 16 units of a recently characterized eight qubit unit cell (Johnson et al. 2011, Harris, Johnson, Lanting, Berkley, Johansson, Bunyk, Tolkacheva, Ladizinsky, Ladizinsky, Oh, Cioata, Perminov, Spear, Enderud, Rich, Uchaikin, Thom, Chapple, Wang, Wilson, Amin, Dickson, Karimi, Macready, Truncik & Rose 2010). 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 (J_{ij}), find the assignment $\mathbf{s} = \mathbf{s_1 s_2 s_3 ... s_N}$, that minimizes the objective function $E(\mathbf{s})$, where,

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

[1] External interactions could also be included as presented and verified experimentally in (Perdomo-Ortiz et al. 2012).

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,

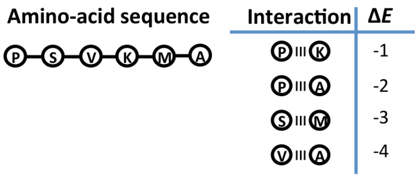
$$\mathcal{H}_p = \sum_{1 \leq i \leq N} h_i \sigma_i^z + \sum_{1 \leq i \leq j \leq N} J_{ij} \sigma_i^z \sigma_j^z. \tag{52}$$

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 (Hydrophobic-Polar model) and 5, 27, 28, and 81 qubits experiments for the six-amino-acid sequence under the Miyazawa-Jernigan model for general pairwise interactions.

7.2 Six unit Miyazawa-Jernigan protein

The example we will present here is a different encoding of the largest problem performed in (Perdomo-Ortiz et al. 2012): the Miyazawa-Jernigan (MJ) protein, Proline-Serine-Valine-Lysine-Methionine-Alanine (PSVKMA) on a $2D$ lattice. We will use the pair-wise nearest-neighbor MJ interaction energies presented in Table 3 of (Miyazawa & Jernigan 1996) and shown in Fig. 20.


Amino-acid sequence
P S V K M A

Interaction	ΔE
P⦀K	-1
P⦀A	-2
S⦀M	-3
V⦀A	-4

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

We will use the turn ancilla construction for our energy function and constrain the first three virtual bits to 010, as before. Recall that the turn ancilla construction requires $2N - 5$ physical information bits; thus, our 6-unit MJ protein will be encoded into 7 bits.

7.2.1 $E_{back}(\boldsymbol{q})$ for 6-unit SAW on $2D$ lattice

Using Eq. 13, we find that our 6-unit protein has the backwards energy function,

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

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

7.2.2 $E_{overlap}(\boldsymbol{q})$ for 6-unit SAW on $2D$ lattice

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

$$\begin{aligned} &E_{overlap}(\boldsymbol{q}) = \lambda_{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 - 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). \end{aligned} \tag{54}$$

We notice that as discussed in Sec. 6, all the 3-local terms here contain at least two physical information qubits (i.e. q_1 through q_7).

7.2.3 $E_{pair}(\boldsymbol{q})$ for MJ-model PSVKMA

Using the J matrix as defined in Eq. 32 we calculate the pair-wise energy function as,

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

7.2.4 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 which still impose the correct constraints. In our problem we choose the value of $\lambda_{overlap}$ by asking ourselves: 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 $\lambda_{overlap} = +10$.

7.2.5 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,

$$\begin{aligned}
q_2q_4 &\rightarrow q_{20} \\
q_1q_3 &\rightarrow q_{21} \\
q_3q_5 &\rightarrow q_{22} \\
q_1q_5 &\rightarrow q_{23} \\
q_2q_6 &\rightarrow q_{24} \qquad (56) \\
q_4q_6 &\rightarrow q_{25} \\
q_3q_7 &\rightarrow q_{26} \\
q_5q_7 &\rightarrow q_{27} \\
q_1q_7 &\rightarrow q_{28} \quad .
\end{aligned}$$

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.

Table 1: 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

To understand how Eq. 48 accomplishes this see Table 7.2.5. In order for Eq. 48 to work we must choose δ_n which 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 which 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 $E(\boldsymbol{q})$ is as difficult as minimizing $E(\boldsymbol{q})$ (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 all negative psuedo-boolean coefficients or the sum of all positive psuedo-boolean coefficients corresponding to the variables being collapsed in $E(\boldsymbol{q})$ (whichever sum is larger).

7.2.6 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.

$$\begin{pmatrix}
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 & -250 & -85924 & -85924 & -85924 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -160 & 320 & 160 & 80 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 320 & 320 & 320 & 320 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -160 & 80 & 40 & 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 & 0 & -100 & 20 & 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 & 0 & -55 & 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 & 320 & 320 & 160 & 80 & 0 & 0 & 0 & 0 & 320 & 320 & 320 & 320 & 320 & 320 & 320 & 320 & 320 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 80 & 80 & 40 & 0 & 0 & 0 & 0 & 160 & 160 & 160 & 160 & 160 & 160 & 160 & 160 & 160 \\
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 & 80 \\
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 & 40 \\
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 & 0 & 30 & 0 & 0 & 8 & 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 & 28 & 16 & 0 & 16 & 0 & 16 & 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 & 160 \\
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 & 128846
\end{pmatrix} \tag{57}$$

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

$$E(\boldsymbol{q}) = \boldsymbol{q}Q\boldsymbol{q}. \tag{58}$$

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,

$$0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0. \tag{59}$$

The energy given by Eq. 58 is -186. In the original expression this corresponds to an energy of $C - 186 = 180 - 186 = -6$. Let's 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:

$$\boldsymbol{q} = \underbrace{01}_{\text{right}}\underbrace{00}_{\text{down}}\underbrace{00}_{\text{down}}\underbrace{10}_{\text{left}}\underbrace{11}_{\text{up}} \tag{60}$$

which corresponds to the fold,

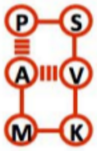


Figure 21: The solution to our example problem for MJ protein PSVKMA.

8 Conclusion

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 review, 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 backwards 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 which 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 under-constrained 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 (Perdomo-Ortiz et al. 2012). Accordingly, as quantum information science continues to develop, we hope that the methods discussed in this review will be useful to scientists wishing to leverage similar technology for the solution of discrete optimization problems.

References

Aloul, F. A., Ramani, A., Markov, I. L. & Sakallah, K. A. (2002). Generic ILP versus specialized 0-1 ILP: an update.

Apolloni, A., Cesa-Bianchi, N. & De Falco, D. (1988). A numerical implementation of quantum annealing, *Stochastic Processes, Physics and Geometry, Proceedings of the Ascona-Locarno Conference* pp. 97–111.

Apolloni, B., Carvalho, C. & De Falco, D. (1989). Quantum stochastic optimization, *Stoc. Proc. Appl.* **33**: 233–244.

Backofen, R. (1998). Using Constraint Programming for lattice Protein Folding, *Energy* **3**: 389–400.

Backofen, R. & Will, S. (2006). A Constraint-Based Approach to Fast and Exact Structure Prediction in Three-Dimensional Protein Models, *Constraints* **11**(1): 5–30.

Baker, D. (2000). A surprising simplicity to protein folding., *Nature* **405**(6782): 39–42.

Berger, B. & Leighton, T. (1998). Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete., *Journal of computational biology a journal of computational molecular cell biology* **5**(1): 27–40.

Boros, E. & Hammer, P. L. (2002). Pseudo-boolean optimization, *Discrete Applied Mathematics* **123**(1-3): 155–225.

Boughaci, D. & Drias, H. (2004). Solving weighted Max-Sat optimization problems using a Taboo Scatter Search metaheuristic, *Proceedings of the 2004 ACM symposium on Applied computing SAC 04* p. 35.

Camacho, C. J. (1995). Entropic Barriers, Frustration and Order: Basic Ingredients in Protein Folding, *Physical Review Letters* **77**(11): 4.

Chandrasekaran, K., Karp, R., Moreno-Centeno, E. & Vempala, S. (2011). Algorithms for Implicit Hitting Set Problems, *SODA* pp. 614–629.

Choi, A., Standley, T. & Darwiche, A. (2009). Approximating Weighted Max-SAT Problems by Compensating for Relaxations, *CP* pp. 211–225.

Chvatal, V. (1979). A Greedy Heuristic for the Set-Covering Problem, *Mathematics of Operations Research* **4**(3): 233–235.

Cook, S. A. (1971). The complexity of theorem-proving procedures, *Proceedings of the third annual ACM symposium on Theory of computing STOC 71* **50**(1): 151–158.

Crescenzi, P., Goldman, D., Papadimitriou, C., Piccolboni, A. & Yannakakis, M. (1998). On the complexity of protein folding., *Journal of computational biology a journal of computational molecular cell biology* **5**(3): 423–465.

Dal Palù, A., Dovier, A. & Fogolari, F. (2004). Constraint Logic Programming approach to protein structure prediction, *BMC Bioinformatics* **5**(1): 186.

Das, A. & Chakrabarti, B. K. (2008). Quantum Annealing and Analog Quantum Computation, *Reviews of Modern Physics* **80**(3): 22.

Denchev, V. S., Ding, N., Vishwanathan, S. V. N. & Neven, H. (2012). Robust Classification with Adiabatic Quantum Optimization, *Proc Int Conf on Machine Learning* p. 1205.1148.

Dill, K. A. (1993). Folding Proteins - Finding A Needle in A Haystack, *Current Opinion in Structural Biology* **3**(1): 99–103.

Dill, K. A. (1995). Simple lattice models of protein folding, *Polymer Preprints American Chemical Society Division of Polymer Chemistry* **36**(1): 635.

Dill, K. A., Ozkan, S. B., Shell, M. S. & Weikl, T. R. (2008). The protein folding problem., *Annual review of biophysics* **37**(1): 289–316.

Eén, N. & Sörensson, N. (2006). Translating Pseudo-Boolean Constraints into SAT, *Journal on Satisfiability Boolean Modeling and Computation* **2**(3-4): 1–26.

Even, S., Itai, A. & Shamir, A. (1976). On the Complexity of Timetable and Multicommodity Flow Problems, *SIAM Journal on Computing* **5**(4): 691–703.

Farhi, E., Goldstone, J., Gutmann, S., Lapan, J., Lundgren, A. & Preda, D. (2001). A Quantum Adiabatic Evolution Algorithm Applied to Random Instances of an NP-Complete Problem, *Science* **292**(5516): 472–475.

Farhi, E., Goldstone, J., Gutmann, S. & Sipser, M. (2000). Quantum Computation by Adiabatic Evolution, *Science* **quant-ph**(162): 24.

Finnila, A. B., Gomez, M. A., Sebenik, C., Stenson, C. & Doll, J. D. (1994). Quantum Annealing: A New Method for Minimizing Multidimensional Functions, *Chemical Physics Letters* **2614**(March): 343–348.

Gruebele, M. & Wolynes, P. G. (1998). Satisfying turns in folding transitions.

Hansen, P. & Jaumard, B. (1990). Approximation Algorithms for the Maximum Satisfiability Problem, *Computing* **7**(656-666): 279–303.

Hansen, P., Jaumard, B. & De Aragao, M. P. (1998). Mixed-integer column generation algorithms and the probabilistic maximum satisfiability problem, *Eur J Operational Research* **108**(3): 671–683.

Harris, R., Johnson, M. W., Lanting, T., Berkley, A. J., Johansson, J., Bunyk, P., Tolkacheva, E., Ladizinsky, E., Ladizinsky, N., Oh, T., Cioata, F., Perminov, I., Spear, P., Enderud, C., Rich, C., Uchaikin, S., Thom, M. C., Chapple, E. M., Wang, J., Wilson, B., Amin, M. H. S., Dickson, N., Karimi, K., Macready, B., Truncik, C. J. S. & Rose, G. (2010). Experimental Investigation of an Eight Qubit Unit Cell in a Superconducting Optimization Processor, *Physical Review B* **82**(2): 16.

Hart, W. E. & Istrail, S. (1997). Robust proofs of NP-hardness for protein folding: general lattices and energy potentials., *Journal of computational biology a journal of computational molecular cell biology* **4**(1): 1–22.

Hemmecke, R., Köppe, M., Lee, J. & Weismantel, R. (2009). Nonlinear Integer Programming, *50 Years of Integer Programming 19582008* p. 57.

IBM (2009). IBM ILOG CPLEX V12.1: User's Manual for CPLEX.

Johnson, M. W., Amin, M. H. S., Gildert, S., Lanting, T., Hamze, F., Dickson, N., Harris, R., Berkley, A. J., Johansson, J., Bunyk, P., Chapple, E. M., Enderud, C., Hilton, J. P., Karimi, K., Ladizinsky, E., Ladizinsky, N., Oh, T., Perminov, I., Rich, C., Thom, M. C., Tolkacheva, E., Truncik, C. J. S., Uchaikin, S., Wang, J., Wilson, B. & Rose, G. (2011). Quantum annealing with manufactured spins, *Nature* **473**(7346): 194–198.

Kadowaki, T. & Nishimori, H. (1998). Quantum Annealing in the Transverse Ising Model, *Physical Review E* **58**(5): 15.

Kahl, F. & Strandmark, P. (2011). Generalized roof duality for pseudo-boolean optimization.

Krippahl, L. & Barahona, P. (1999). Applying Constraint Programming to Protein Structure Determination, *Principles and Practice of Constraint Programming CP99* **1713**: 289–302.

Larrosa, J., Heras, F. & De Givry, S. (2006). A Logical Approach to Efficient Max-SAT solving, *Artificial Intelligence* **172**(2-3): 204–233.

Lau, K. F. & Dill, K. A. (1989). A lattice statistical mechanics model of the conformational and sequence spaces of proteins, *Macromolecules* **22**(10): 3986–3997.

Laue, S. (2008). Geometric Set Cover and Hitting Sets for Polytopes in Rˆ3, *Science* **2008**: 479–490.

Manolios, P. & Papavasileiou, V. (2011). Pseudo-Boolean Solving by incremental translation to SAT.

Marques-Silva, J. a. & Sakallah, K. A. (2007). Theory and Applications of Satisfiability Testing - SAT 2007, 10th International Conference, Lisbon, Portugal, May 28-31, 2007, Proceedings, *in* J. a. Marques-Silva & K. A. Sakallah (eds), *SAT*, Vol. 4501 of *Lecture Notes in Computer Science*, Springer, p. 384.

Mirny, L. & Shakhnovich, E. (2001). Protein folding theory: from lattice to all-atom models., *Annual Review of Biophysics and Biomolecular Structure* **30**(NIL): 361–96.

Miyazawa, S. & Jernigan, R. L. (1996). Residue-residue potentials with a favorable contact pair term and an unfavorable high packing density term, for simulation and threading., *Journal of Molecular Biology* **256**(3): 623–644.

Neven, H., Rose, G. & Macready, W. G. (2008). Image recognition with an adiabatic quantum computer I. Mapping to quadratic unconstrained binary optimization, *Computer* **3**: 7.

Nymeyer, H., García, A. E. & Onuchic, J. N. (1998). Folding funnels and frustration in off-lattice minimalist protein landscapes., *Proceedings of the National Academy of Sciences of the United States of America* **95**(11): 5921–5928.

Oakley, M. T., Wales, D. J. & Johnston, R. L. (2011). Energy Landscape and Global Optimization for a Frustrated Model Protein, *The Journal of Physical Chemistry B* **115**(August): 11525–9.

Pande, V., Grosberg, A. & Tanaka, T. (2000). Heteropolymer freezing and design: Towards physical models of protein folding, *Reviews of Modern Physics* **72**(1): 259–314.

Pande, V. S. (2010). Simple theory of protein folding kinetics., *Physical Review Letters* **105**(19): 198101.

Pankratov, D. & Borodin, A. (2010). On the Relative Merits of Simple Local Search Methods for the MAX-SAT Problem, *in* O. Strichman & S. Szeider (eds), *Theory and Applications of Satisfiability Testing SAT 2010*, Vol. 6175 of *Lecture Notes in Computer Science*, Springer, pp. 223–236.

Perdomo, A., Truncik, C., Tubert-Brohman, I., Rose, G. & Aspuru-Guzik, A. (2008). On the construction of model Hamiltonians for adiabatic quantum computation and its application to finding low energy conformations of lattice protein models, *Physical Review A* **78**(1): 35.

Perdomo-Ortiz, A., Dickson, N., Drew-Brook, M., Rose, G. & Aspuru-Guzik, A. (2012). Finding low-energy conformations of lattice protein models by quantum annealing, *Nature Scientific Reports* **2**.

Sakallah, K. A. (2006). Pueblo : A Hybrid Pseudo-Boolean SAT Solver, *Electrical Engineering* **2**: 155–179.

Sali, A., Shakhnovich, E. & Karplus, M. (1994). How does a protein fold?, *Nature* **369**(6477): 248–251.

Santoro, G. E. & Tosatti, E. (2006). Optimization using quantum mechanics: quantum annealing through adiabatic evolution, *Journal of Physics A: Mathematical and General* **39**(36): R393–R431.

Schram, R. D., Barkema, G. T. & Bisseling, R. H. (2011). Exact enumeration of self-avoiding walks, *Journal of Statistical Mechanics: Theory and Experiment* **2011**(06): 6.

Shakhnovich, E. I. (1994). Proteins with selected sequences fold into unique native conformation, *Physical Review Letters* **72**(24): 3907–3910.

Shakhnovich, E. I. (1996). Modeling protein folding: the beauty and power of simplicity., *Folding design* **1**(3): R50–R54.

Shea, J.-E., Onuchic, J. N. & Brooks, C. L. (2000). Energetic frustration and the nature of the transition state in protein folding, *Journal of Chemical Physics* **113**(17): 7663–7671.

Sheini, H. M. & Sakallah, K. A. (2005). Pueblo: a modern pseudo-Boolean SAT solver.

Shi, L. S. L. & Cai, X. C. X. (2010). An Exact Fast Algorithm for Minimum Hitting Set.

Smelyanskiy, V. N., Rieffel, E. G., Knysh, S. I., Williams, C. P., Johnson, M. W., Thom, M. C., Macready, W. G. & Pudenz, K. L. (2012). A Near-Term Quantum Computing Approach for Hard Computational Problems in Space Exploration, *Electrical Engineering* p. 68.

Soos, M., Nohl, K. & Castelluccia, C. (2009). Extending SAT Solvers to Cryptographic Problems, *SAT* **5584**: 244–257.

Ullah, A. D. & Steinhöfel, K. (2010). A hybrid approach to protein folding problem integrating constraint programming with local search., *BMC Bioinformatics* **11 Suppl 1**: S39.

Xing, Z. & Zhang, W. (2005). MaxSolver: An efficient exact algorithm for (weighted) maximum satisfiability, *Artificial Intelligence* **164**(1-2): 47–80.

Yue, K. & Dill, K. A. (1995). Forces of tertiary structural organization in globular proteins., *Proceedings of the National Academy of Sciences of the United States of America* **92**(1): 146–150.