SAND REPORT

SAND2002-xxxx
Unlimited Release
August 2002

Discrete Optimization Models for Protein Folding

Bob Carr and Bill Hart, Sandia
Alantha Newman, MIT

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of
Energy under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online ordering:	http://www.ntis.gov/ordering.htm

SAND2002-xxxx
Unlimited Release
Printed August 2002

Distribution
Category UC-999

Discrete Optimization Models for Protein Folding

Bob Carr and Bill Hart
Organization 09215
Discrete Algorithms and Math Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-9999
rdcarr,wehart@sandia.gov

Alantha Newman
Laboratory for Computer Science
MIT
Cambridge, MA
alantha@theory.lcs.mit.edu

Abstract

Protein folding is an important problem in Computational Biology; the function of a protein depends on the three-dimensional shape to which it folds. The HP model is a widely studied model of protein folding that abstracts the dominant force in protein folding: the hydrophobic interaction. We develop discrete optimization models to help predict the best (i.e. lowest energy) folding in this model.

Contents

1 Introduction 7
2 Problem Statement 8
Notation 8
An Upper Bound 9
3 Integer and Linear Programs 9
Variables for IP and LP Formulations 10
Integer Programs 11
Linear Programs 12
4 More Integer Programming Formulations 13
Aggregate Constraints 14
Quality of the LP Solution 17
Backbone Constraints 19
Another IP and LP Formulation 21
5 Branch and Bound 27
6 Integrality Gaps 29
7 Six Index Constraints 30
8 Future Work 31
9 Implementation 32
AMPL Code for LP with 2-Index Variables and Flow Constraints (LP_3). . . . 32
Experimental Results 36
References 37

Figures

1 An example of a string folding 8
2 Showing one constraint does not imply another 14
3 Backbone constraints 19
4 An example in which LP_3 and LP_4 have different objective values. 27
5 A string which shows a gap of 2 for LP_3 and LP_4 29

Discrete Optimization Models for Protein Folding

1 Introduction

The protein folding problem is an important and widely-studied problem in Computational Biology. A protein is a sequence of 100-300 amino acid residues. Shorter amino acid chains are called peptides. There are approximately 20 different amino acids. The functions of proteins and peptides are determined by their respective three-dimensional (3D) shapes. Under certain standard conditions (e.g. extreme heat may cause a protein to unfold), proteins always fold to the same unique native 3D structure. This shape is principally determined by the one-dimensional (1D) sequence. This was shown by Christian Anfinsen, who won the 1972 Nobel Prize in Chemistry for his work on protein structure in living cells [1]. He wondered why a protein folded into a particular 3D shape and what (e.g. enzymes?) directed it to this folding. In an article in the Journal of Biological Chemistry [3], he showed that the sequence of amino acids in a protein or peptide chain determines the folding pattern. In other words, the process of protein folding can be largely explained by the physical and chemical interactions among the amino acids. This work is the basis for the belief the native structure of a protein can be predicted *computationally* using the information contained in the amino acid sequence [7].

In this report, we discuss discrete optimization approaches to the problem of protein folding in the Hydrophobic-Polar (HP) model (also known as the Hydrophobic-Hydrophilic model). The widely-studied HP model was introduced by Ken Dill [4, 5]. This model abstracts the dominant force in protein folding: the hydrophobic interaction. The hydrophobicity of an amino acid measures its affinity for water. The hydrophobic resides in a protein form a tightly clustered core. In the HP model, each amino acid residue is classified as an H (hydrophobic or non-polar) or a P (hydrophilic or polar). The model further simplifies the problem by restricting the feasible foldings to the 2D or 3D square lattice. An optimal conformation for a string of amino residues in this model is one that maximizes the number of H-H contacts, i.e. pairs of H's that are adjacent in the folding but are not neighbors on the string. Thus, the problem of protein folding in the HP model is combinatorially equivalent to folding a given string of 0's and 1's on the square lattice to form a self-avoiding walk that maximizes the number of pairs of adjacent 1's, i.e. let H=1 and P=0.

One of the most immediately obvious drawbacks of the HP model is that on the square lattice, residues in even positions in the given string can have as their neighbors on the lattice only residues from odd positions in string and vice versa. In the actual protein folding problem, there is no such restriction. The HP model has also been studied on the 2D and 3D triangular lattice [2], which does not have this parity problem. We believe that our methods can be extended to the triangular lattice. However, the square lattice seems to be the best place to start computational experiments since a string has fewer possible conformations on the square lattice than it does on the triangular lattice.

2 Problem Statement

We are given a string of 0's and 1's. Our goal is to find a valid folding of this string on the 2D square lattice that maximizes the number of pairs of adjacent 1's. In other words, we want to find a self-avoiding walk that maximizes the number of pairs of adjacent 1's when the string is superimposed on it. For example, suppose we have the string 101010101001010101. Then an optimal folding is shown in Figure 1. This folding has eight contacts.

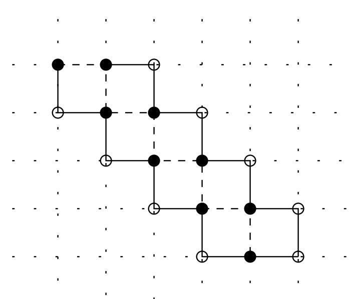

Figure 1. An optimal folding for the string 101010101001010101. 0's and 1's are denoted by unfilled and filled dots, respectively. Contacts are denoted by the dashed lines.

Notation

Let S be a string in $\{0,1\}^n$. We will refer to each 0 and each 1 in the given string as an *element*. We will refer to each 1 in an odd position on the string as an *odd-1* and each 1 in an even position on the string as an *even-1*. We will denote the number of odd-1's in the string S as $\mathcal{O}[S]$ and the number of even-1's in a string S as $\mathcal{E}[S]$.

An Upper Bound

The best-known upper bound was introduced in [7]. An even-1 or an odd-1 can have at most 2 contacts if it is not the first or last element on the string. The first and last element on the string can each have at most 3 contacts. Thus, an upper bound on the maximum number of contacts in any folding of a given string S is:

$$2 * \min\{\mathcal{O}[S], \mathcal{E}[S]\} + 2.$$

Comparing the optimal values produced by our models to this upper bound gives us some idea of how well our models are performing. These upper bounds are also used to obtain approximate solutions for this problem [7, 8].

3 Integer and Linear Programs

In this section, we present some integer programs (IPs) for the protein folding problem in the HP model as well as their respective linear programming relaxations (LPs). First, we will introduce the necessary notation.

Let I be the set of indices in S, i.e. $I = \{1, \ldots n\}$. We break down I as follows:

$\mathcal{E}$ is the set of indices of elements in even positions.

$\mathcal{O}$ is the set of indices of elements in odd positions.

We break down $\mathcal{E}$ and $\mathcal{O}$ further as follows:

$H_{\mathcal{O}}$ is the set of indices of odd-1's in S,

$H_{\mathcal{E}}$ is the set of indicies of even-1's in S,

$P_{\mathcal{O}}$ is the set of indices of odd-0's in S,

$P_{\mathcal{E}}$ is the set of indices of even-0's in S.

Thus, $H_{\mathcal{E}} \cup P_{\mathcal{E}} = \mathcal{E}$ and $H_{\mathcal{O}} \cup P_{\mathcal{O}} = \mathcal{O}$ and $H_{\mathcal{E}} \cup P_{\mathcal{E}} \cup H_{\mathcal{O}} \cup P_{\mathcal{O}} = \mathcal{E} \cup \mathcal{O} = I$.

Let V represent the set of feasible vertices in the lattice, i.e. a vertex occurs at each intersection of a horizontal and vertical line in the lattice. We will assume that one of the points (e.g. the odd point closest to the middle) on the string is assigned to a particular lattice point, which defines the feasible region of vertices in the lattice. In other words, once this middle element is fixed, there are only a finite number of lattice points to which we can assign the other elements. We classify the points in V as follows:

$V_{\mathcal{E}}$ is the set of even lattice points in V.

$V_{\mathcal{O}}$ is the set of odd lattice points in V.

Let $\delta(v)$ denote the set of feasible vertices adjacent to v, which consists of at most four lattice points. The set of feasible edges in the lattice is denoted by E, which is the set of (v, w) such that $v \in V_{\mathcal{O}}$ and $w \in V_{\mathcal{E}}, w \in \delta(v)$.

Variables for IP and LP Formulations

Now we will define the variables that we will use in our various integer programs. First we list the variables that we use:

1. $h_{(iv)(jw)} \quad \forall i \in H_{\mathcal{O}}, j \in H_{\mathcal{E}}, (v, w) \in E$
2. $h_{(vw)} \quad \forall (v, w) \in E,$
3. $x_{iv} \quad \forall i \in H_{\mathcal{O}}, v \in V_{\mathcal{O}},$
4. $x_{jw} \quad \forall j \in H_{\mathcal{E}}, w \in V_{\mathcal{E}}.$

Now we will explain the function/meaning of each variable. We will not use all the variables immediately–some will be used in integer programs introduced later on in the paper. Also, by convention, we will always use i and v to refer to indices for odd elements on the string and odd lattice points, respectively. Similarly, we will always use j and w to refer to indices for even elements on the string and even lattice points, respectively.

The variable $h_{(iv)(jw)}$ indicates whether or not there is a contact between elements i and j on edge (v, w). For example, if $h_{(iv)(jw)}$ is set to 1 in an integer program, then there is a contact between i and j across edge (v, w), and if $h_{(iv)(jw)}$ is set to 0, then there is no contact between i and j on edge (v, w).

The variable $h_{(v,w)}$ represents the total amount of contacts between all odd elements and all even elements on edge (v, w). In an integer solution, if there is a contact between *any* $i \in H_{\mathcal{O}}$ and *any* $j \in H_{\mathcal{E}}$ on edge (v, w), then the value of $h_{(vw)}$ would be 1. If there are no contacts on this edge, the value of $h_{(vw)}$ would be 0. Note that there is a relationship between the variables $h_{(iv)(jw)}$ and $h_{(vw)}$.

$$h_{(v,w)} = \sum_{i \in H_{\mathcal{O}}} \sum_{j \in H_{\mathcal{E}}} h_{(iv)(jw)}. \tag{1}$$

The variable x_{iv} indicates whether or not the element i is placed on vertex point v. In an integer solution, x_{iv} is set to 1 if element i is placed on lattice point v and 0 otherwise. Odd elements are placed only on odd lattice points and even elements are placed only on even lattice points. Thus, we distinguish between these two cases and create variables x_{iv} for the odd case and x_{jw} for the even case. Note that any string folding corresponds to a 0-1 assignment of the variables $\{x_{iv}, x_{jw}\}$. However, note that not every 0-1 assignment to the variables corresponds to a folding, which is why we need to impose constaints on these variables.

Integer Programs

The following integer program is one possible integer program for our problem. Every integer solution defines a valid folding and every folding corresponds to an integer solution. Thus, there is a one-to-one correspondence between foldings and integer solutions.

IP$_1$:

$$
\begin{array}{rrcll}
\max \displaystyle\sum_{(v,w)\in E} \sum_{i\in H_{\mathcal{O}}} \sum_{j\in H_{\mathcal{E}}} & h_{(iv)(jw)} & & & \\
\textit{subject to}: \quad \displaystyle\sum_{v\in V} & x_{iv} & = & 1 \quad \forall i \in I & (2)\\
\displaystyle\sum_{i\in I} & x_{iv} & \leq & 1 \quad \forall v \in V & (3)\\
\displaystyle\sum_{w\in\delta(v)} & x_{i+1w} & \geq & x_{iv} \quad \forall i \in I \setminus \{n\}, v \in V & (4)\\
\displaystyle\sum_{j\in H_{\mathcal{E}}} & h_{(iv)(jw)} & \leq & x_{iv} \quad \forall i \in H_{\mathcal{O}}, (v,w) \in E & (5)\\
\displaystyle\sum_{i\in H_{\mathcal{O}}} & h_{(iv)(jw)} & \leq & x_{jw} \quad \forall j \in H_{\mathcal{E}}, (v,w) \in E & (6)\\
 & h_{(iv)(jw)},\ x_{iv},\ x_{jw} & \in & \{0,1\}. & (7)
\end{array}
$$

Lemma 1. *There is a one-to-one correspondence between foldings and integer solutions.*

Proof: Showing that every folding corresponds to an integer solution is easy. We will show that every integer solution corresponds to a folding. In an integer solution, for each element i, there is exactly one v such that $x_{iv} = 1$ (constraint (2)). Moreover, each lattice point v contains at most one element (constraint (3)). Constraint (4) guarantees that each consecutive element on the string is placed on an adjacent lattice point to its neighbor on the string. Thus, we have a valid folding. □

Constraints (5) and (6) are used to force elements to be placed on lattice points v and w if there there is a contact between elements i and j on edge (v, w). Constraint (7) enforces the integrality of all the variables. It is possible that we only need to force the x variables to be integer and this will automatically enforce the h variables to be integer.

Linear Programs

We obtain a linear programming relaxation from IP$_1$ by relaxing constraint (7) to the following:

$$0 \ \leq \ x_{iv}, x_{jw} \ \leq \ 1. \tag{8}$$

A linear program provides an upper bound on the optimal integral solution. Also, of key importance is the fact that it can be solved much faster than an integer program. One way to measure the quality of an integer program is to determine the upper bound guaranteed by its linear relaxation. In general, the better the bound provided by the linear relaxation, the higher the quality of the integer program.

4 More Integer Programming Formulations

There are many other ways to formulate this problem as an integer program. For example, in IP_1, we could replace constraint (4) with constraint (9), which is shown below.

$$\sum_{w \in \delta(v)} x_{i-1w} \quad \geq \quad x_{iv} \quad \forall i \in I \setminus \{n\}, v \in V. \tag{9}$$

This would also result in a valid integer program. Alternatively, we can include both constraints (9) and (4). We will show that including both these constraints leads to a stronger linear program than including only one of these constraints. We will add constraint (9) to IP_1 and refer to its corresponding linear programming relaxation as LP_1.

It is not immediately clear that constraints (9) and (4) are both necessary, i.e. that constraint (9) does not imply constraint (4) or vice versa. However, we will show that constraint (9) does not imply constraint (4) or vice-versa. To do this we will give a feasible LP solution for a string of length 9 such that constraint (4) is obeyed but constraint (9) is violated.

Such a feasible solution is shown in Figure 2. The values shown in Figure 2 are the fractions of each x_i that are placed at the labeled lattice points, i.e. the x_{iv} values. Let $i = 6,\ v = q$. Note that constraint (9) is violated for x_{6q} since $x_{6q} = 2/3$ and $\sum_{w \in \delta(q)} x_{5w} = 1/3$. Note that constraint (4) is not violated for any of the x_{iv} variables. We can repeat this argument for the string labeled in the reverse order and we would obtain an example in which constraint (9) is not violated but constraint (4) is violated. Thus neither constraint is implied by the other.

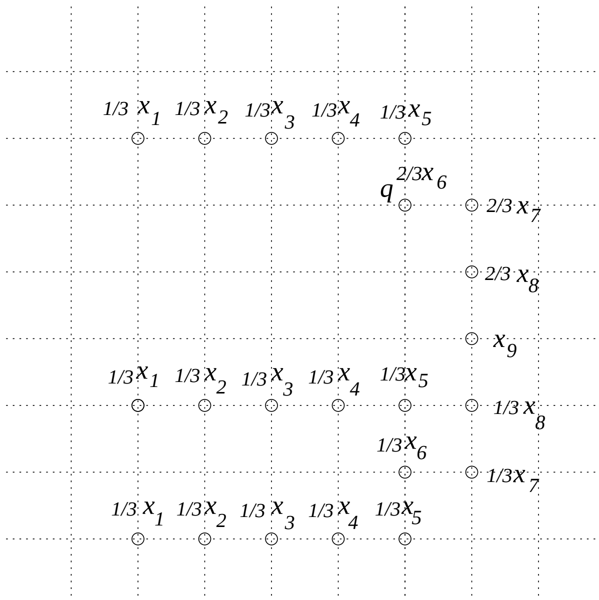

Figure 2. Constraint (9) is violated for $i = 6$, $v = q$. However, note that no other constraints (e.g. constraint (4)) are violated.

Aggregate Constraints

We can obtain another integer program and its corresponding linear programming relaxation by replacing constraints (5) and (6) in IP_1 and LP_1 with the aggregate constraints (10) and (11). We refer to the resulting integer and linear programs as IP_2 and LP_2, respectively.

$$\sum_{i \in H_{\mathcal{O}}} \sum_{j \in H_{\mathcal{E}}} h_{(iv)(jw)} \leq \sum_{i \in H_{\mathcal{O}}} x_{iv} \quad \forall (v, w) \in E \tag{10}$$

$$\sum_{j \in H_{\mathcal{E}}} \sum_{i \in H_{\mathcal{O}}} h_{(iv)(jw)} \leq \sum_{j \in H_{\mathcal{E}}} x_{jw} \quad \forall (v, w) \in E \tag{11}$$

We can use the variables $h_{(vw)}$ to simplify IP_2. Thus, IP_2 is a formulation which has fewer h variables than IP_1. Recall the definition of $h_{(vw)}$ from (1).

$$h_{(v,w)} = \sum_{i \in H_{\mathcal{O}}} \sum_{j \in H_{\mathcal{E}}} h_{(iv)(jw)}$$

We restate IP_2 here for clarity and convenience:

IP$_2$:

$$\begin{aligned}
\max \sum_{v \in V_{\mathcal{O}}} \sum_{w \in \delta(v)} & h_{(v,w)} \\
\textit{subject to}: \quad \sum_{v \in V} x_{iv} &= 1 \quad \forall i \in I \\
\sum_{i \in I} x_{iv} &\leq 1 \quad \forall v \in V \\
\sum_{w \in \delta(v)} x_{i-1w} &\geq x_{iv} \quad \forall i \in I \setminus \{n\}, v \in V \\
\sum_{w \in \delta(v)} x_{i+1w} &\geq x_{iv} \quad \forall i \in I \setminus \{n\}, v \in V \\
h_{(v,w)} &\leq \sum_{i \in H_{\mathcal{O}}} x_{iv} \quad \forall (v,w) \in E \\
h_{(v,w)} &\leq \sum_{j \in H_{\mathcal{E}}} x_{jw} \quad \forall (v,w) \in E \\
x_{iv},\ x_{jw} &\in \{0,1\}.
\end{aligned}$$

It is clear that the optimal objective value for LP_2 is at least as large as the optimal objective value for LP_1. This is because a solution for LP_1 does not violate any constraints in LP_2. Additionally, we can also show that the optimal objective value for LP_1 is at least as large as the optimal objective value for LP_2 when the objective function is of the form $\{c_{vw}\}$, i.e. there is a cost function that associates a cost with every edge (v,w).

Lemma 2. *The optimal values of LP_1 and LP_2 are equal, i.e. $|OPT(LP_1)| = |OPT(LP_2)|$.*

Proof: We will show that if we use any objective function of the form $\{c_{vw}\}$, then the objective values of LP_1 and LP_2 will be the same. First, we will show that given a set of $\{h_{(iv)(jw)}\}$, we can find a set of $\{h_{(vw)}\}$ such that $\{h_{(vw)}\}$ satisfy all the constraints in LP_2. We define $h_{(vw)}$ as follows:

$$\sum_{i \in H_{\mathcal{O}}} \sum_{j \in H_{\mathcal{E}}} h_{(iv)(jw)} = h_{(vw)}. \tag{12}$$

Constraints (5) and (6) imply (10) and (11). Thus, using (1), we see that from any feasible solution for LP_1, we can obtain a feasible solution for LP_2 with the same objective value.

Now, we want to show that given a solution for LP_2, i.e. given a set of $\{h_{(vw)}\}$, we can find a solution set $\{h_{(iv)(jw)}\}$ that obeys all the constraints in LP_1. Without loss of generality, assume that for some v, w:

$$\sum_{i \in H_{\mathcal{O}}} x_{iv} \leq \sum_{j \in H_{\mathcal{E}}} x_{jw}.$$

Consider the following table for v, w. Assume there are k i's in $H_{\mathcal{O}}$ labeled $i_1 \ldots i_k$ and assume there are m j's in $H_{\mathcal{E}}$ labeled $j_2 \ldots j_m$.

	i:	1	3	5	$\ldots$	k		
j :								
2		$h_{(1v)(2w)}$	$h_{(3v)(2w)}$	$h_{(5v)(2w)}$	$\cdots$	$h_{(kv)(2w)}$	$\leq$	x_{2w}
4		$h_{(1v)(4w)}$	$h_{(3v)(4w)}$	$h_{(5v)(4w)}$	$\cdots$	$h_{(kv)(4w)}$	$\leq$	x_{4w}
6		$h_{(1v)(6w)}$	$h_{(3v)(6w)}$	$h_{(5v)(6w)}$	$\cdots$	$h_{(kv)(6w)}$	$\leq$	x_{6w}
.		.	.	.		.		.
.		.	.	.		.		.
.		.	.	.		.		.
m		$h_{(1v)(mw)}$	$h_{(3v)(mw)}$	$h_{(5v)(mw)}$	$\cdots$	$h_{(kv)(mw)}$	$\leq$	x_{mw}
		x_{1v}	x_{3v}	x_{5v}	$\cdots$	x_{kv}		

We are trying to assign a value to each $h_{(iv)(jw)}$ so that constraints (5) and (6) are not violated and equality (12) is met.

We will assign values to the $h_{(iv)(jw)}$ variables in the first column so that the sum of the variables in the first column is equal to x_{1v}. We can do this by setting $h_{(1v)(2w)}$ to be as large as possible such that it is at most x_{2w} and at most x_{1v}. Then we set $h_{(1v)(4w)}$ to be as large as possible so that the sum of the two variables is no more than

x_{1v} and $h_{(1v)(4w)}$ is no greater than $x_{(4w)}$. We repeat this for $h_{(1v)(jw)}$, where $j > 4$ and $j \in H_{\mathcal{E}}$. When we are done, we will have the following:

$$\sum_{j \in H_{\mathcal{E}}} h_{(1v)(jw)} = x_{1v}.$$

Then we repeat for x_{3v}, etc. Recall that the sum of the x_{iv}'s is no more than the sum of the x_{jw}'s. Thus, we can always find an assignment for the $h_{(iv)(jw)}$'s such that none of the constraints are violated. If for some x_{iv}, we could not find a set of $h_{(iv)(jw)}$ variables to assign the value (because doing so would violate constraint (6)) then we would have a contradiction, since this would mean that the sum of the x_{jw}'s is less than the sum of the x_{iv}'s. $\square$

Lemma 3. *For a given string S, the values of the x_{iv} variables in optimal LP_1 and LP_2 solutions are the same. In other words, the projections of the LP_1 and LP_2 solutions onto the x variables are the same.*

Proof: Note that in the proof of Lemma 2, as we go from the $\{h_{vw}\}$ variables to the $h_{(iv)(jw)}$ variables and vice-versa, we use the same set of x variables. $\square$

Another way to deal with LP$_2$ is to not have variables for $h_{(iv)(jw)}$ when i and j are consecutive, i.e. $j = i+1$ or $j = i-1$. In this case, the proof of Lemma 2 does not go through. However, note that it still goes through if there are no consecutive 1's in the input string S. If there are consecutive 1's in the input string S, then the bound provided by LP$_2$ with this alternation could be better than the bound provided by LP$_1$. However, we will show in the next section that the quality of the LP solutions are roughly the same regardless of whether or not we allow $h_{(iv)(jw)}$ variables for consecutive i and j.

Quality of the LP Solution

Unfortunately, the relaxations discussed so far may not provide fractional solutions that are very close to integral solutions. As noted in Section 2, the upper bound on the number of contacts in a string S is $2 * \min\{\mathcal{O}[S], \mathcal{E}[S]\} + 2$. These relaxations can yield a fractional answer that is twice as large as this upper bound.

Lemma 4. *The objective values of LP_1 and LP_2 are each at least* $4 * \min(\mathcal{O}[S], \mathcal{E}[S])$.

Proof: To show this, we will give a solution for LP_1 that is valid for any string S and that has an objective value of $4 * \min\{\mathcal{O}[S], \mathcal{E}[S]\}$. We will let k represent the number of elements in S, i.e. the length of S. Without loss of generality, assume $\mathcal{O}[S] \leq \mathcal{E}[S]$ and let n be the number of lattice points, i.e. $|V_{\mathcal{O}}| = |V_{\mathcal{E}}| = \frac{n}{2}$. We also assume $k \leq n$, i.e. the string can actually be folded onto the lattice. We let $x_{iv} = \frac{2}{n}$ for all $i \in H_{\mathcal{O}}, v \in V_{\mathcal{O}}$ and $x_{jw} = \frac{2}{n}$ for all $j \in H_{\mathcal{E}}, w \in V_{\mathcal{E}}$. Then we let $h_{(iv)(jw)} = \frac{2}{(\mathcal{E}[S])n}$ for all $i \in H_{\mathcal{O}}, j \in H_{\mathcal{E}}, v \in V_{\mathcal{O}}, w \in V_{\mathcal{E}}$.

Note that constraint (2) is satisfied since for each i, there are $\frac{n}{2}$ possible $v \in V$ with the same parity. Constraint (3) will be satisfied because we have:

$$\sum_{i \in I} x_{iv} = \sum_{i \in H_{\mathcal{O}}} x_{iv} \leq \frac{k}{2} * \frac{2}{n} \leq 1.$$

Constraints (9) and (4) will be satisfied as long as each lattice point v has at least one neighbor. Constraint (5) is satisfied since for $i \in H_{\mathcal{O}}$, we have $\frac{2}{(\mathcal{E}[S])n} * \mathcal{O}[S] \leq \frac{2}{n}$ and for even i, we have $\frac{2}{(\mathcal{E}[S])n} * \mathcal{E}[S] = \frac{2}{n}$. The number of $h_{(iv)(jw)}$ variables is $\mathcal{E}[S] * \mathcal{O}[S] * 4(\frac{n}{2})$. This is because there are $\mathcal{E}[S] * \mathcal{O}[S]$ pairs of 1's such that odd-1's are paired with even-1's. And there are $\frac{n}{2}$ odd lattice points each with 4 neighbors, i.e. each odd lattice point serves as an endpoint for 4 edges so we have a total of $4(\frac{n}{2})$ edges. Thus, the objective value will be:

$$\max \sum_{i \in H_{\mathcal{O}}} \sum_{j \in H_{\mathcal{E}}} \sum_{v \in V_{\mathcal{O}}} \sum_{w \in \delta(v)} h_{(iv)(jw)} \;=\; \mathcal{E}[S] * \mathcal{O}[S] * \frac{n}{2} * 4 * \frac{2}{\mathcal{E}[S]n} \;=\; 4\mathcal{O}[S]. \quad (13)$$

So the value of the objective function is at least $4 * \min\{\mathcal{O}[S], \mathcal{E}[S]\}$. Note that this is the right value asymptotically. Since we can choose the n lattice points so that they form a convex region, about $4\sqrt{n}$ of the lattice points have less than 4 neighboring lattice points. $\square$

If we use the 4 index formulation but do not allow $h_{(iv)(jw)}$ variables for consecutive i and j, then we can still use the same values for the x variables. However, asymptotically, this does not change the value of the LP solution given in Equation 13. Specifically, for every $j \in H_{\mathcal{E}}$ and edge $(v, w) \in E$, there are only $\mathcal{O}[S] - 2$ $h_{(iv)(jw)}$ variables. So when we remove the $h_{(iv)(jw)}$ variables for consecutive i and j, the value of the optimal LP_2 solution is at least:

$$\max \sum_{i\in H_{\mathcal{O}}} \sum_{j\in H_{\mathcal{E}}} \sum_{v\in V_{\mathcal{O}}} \sum_{w\in\delta(v)} h_{(iv)(jw)} \;=\; \mathcal{E}[S] * (\mathcal{O}[S]-2) * \frac{n}{2} * 4 * \frac{2}{\mathcal{E}[S]n} \;=\; 4\mathcal{O}[S]-8.$$

The integrality gap for both formulations is 4 since there are strings for which the optimal folding achieves only $o(1)+\min\{\mathcal{O}[S],\mathcal{E}[S]\}$ contacts [8].

Backbone Constraints

We can add more constraints to strengthen our LP. Figure 3 gives an example where adding new constraints may help. Figure 3 depicts a situation in which $x_{iv} = x_{j+1,v} = \frac{1}{2}$ and $x_{i+1,w} = x_{jw} = \frac{1}{2}$. If $i, j+1 \in H_{\mathcal{O}}$ and $j, i+1 \in H_{\mathcal{E}}$, then $h_{(iv)(jw)}$ and $h_{(j+1,v)(i+1,w)}$ can each be assigned a value as high as $\frac{1}{2}$.

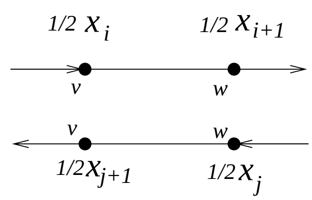

Figure 3. An example in which *backbone* constraints can be added to the LP formulation to give a better bound on the optimal folding.

Even in a fractional solution, this situation should not occur because the *backbone* or actual string is occupying the edge so the edge cannot be used for a contact. For example, in an integral solution, if element i were placed on lattice point v and element $i+1$ were placed on lattice point w, then the edge (v, w) would not be used for any contacts since it is occupied by the actual string.

In order to make the optimal LP value closer to the optimal integer value of a folding, we will add constraints that we refer to as *backbone* constraints. We will use the following variables: The variable $E_{(iv)(i+1,w)}$ means that element i is on lattice position v and element $i+1$ is on lattice element w. Since these variables are only for *consecutive* elements on the string, we can abbreviate them as follows:

$$E^+_{ivw} = E_{(iv)(i+1,w)}, \qquad\qquad E^-_{ivw} = E_{(iv)(i-1,w)}.$$

Then we can add the following valid inequalities to strengthen our LP formulation. We will add these constraints to LP_1 and refer to the resulting LP as LP_3.

$$\begin{aligned}
\sum_{w\in\delta(v)} E^-_{ivw} &= x_{iv} && (14)\\
\sum_{w\in\delta(v)} E^+_{ivw} &= x_{iv} &&\\
\sum_{v\in\delta(w)} E^-_{j+1,vw} &= x_{jw} &&\\
\sum_{v\in\delta(w)} E^+_{j-1,vw} &= x_{jw} &&\\
\sum_{i\in H_{\mathcal{O}}} E^-_{ivw} + \sum_{i\in H_{\mathcal{O}}} E^+_{ivw} + h_{(v,w)} &\leq \sum_{i\in H_{\mathcal{O}}} x_{iv} && (15)\\
\sum_{j\in H_{\mathcal{E}}} E^-_{j+1,vw} + \sum_{j\in H_{\mathcal{E}}} E^+_{j-1,vw} + h_{(v,w)} &\leq \sum_{j\in H_{\mathcal{E}}} x_{jw}. &&
\end{aligned}$$

Note that if $i \in H_{\mathcal{O}}$ and $i+1 \in H_{\mathcal{E}}$, then E^+_{ivw} has the same function as $h_{(iv)(i+1,w)}$. Similarly for E^-_{ivw} and $h_{(iv)(i-1,w)}$. Also, note that if $i \in H_{\mathcal{O}}$ and $i-1, i+1 \in H_{\mathcal{E}}$, then constraint (15) (written below) is the same as constraint (10).

$$\sum_{i\in H_{\mathcal{O}}} E^-_{ivw} + \sum_{i\in H_{\mathcal{O}}} E^+_{ivw} + \sum_{i\in H_{\mathcal{O}}} \sum_{j\in H_{\mathcal{E}}, j\neq i-1,i+1} h_{(iv)(jw)} \leq \sum_{i\in H_{\mathcal{O}}} x_{iv}.$$

In LP_1, the four-index LP, constraint (15) would be replaced with:

$$E^-_{ivw} + E^+_{ivw} + \sum_{j\in H_{\mathcal{E}}, j\neq i+1,i-1} h_{(iv)(jw)} \leq x_{iv}.$$

Another IP and LP Formulation

IP$_3$:

$$
\begin{array}{rcll}
\max \displaystyle\sum_{(v,w)\in E} h_{(vw)} & & & \\
subject\ to: \quad \displaystyle\sum_{v\in V_{\mathcal{O}}} x_{iv} & = & 1 & \forall i \in H_{\mathcal{O}} \\
\displaystyle\sum_{v\in V_{\mathcal{E}}} x_{jw} & = & 1 & \forall j \in H_{\mathcal{E}} \\
\displaystyle\sum_{i\in H_{\mathcal{O}}} x_{iv} & \leq & 1 & \forall v \in V_{\mathcal{O}} \\
\displaystyle\sum_{j\in H_{\mathcal{E}}} x_{jw} & \leq & 1 & \forall w \in V_{\mathcal{E}} \\
\displaystyle\sum_{w\in \delta(v)} E^{-}_{ivw} & = & x_{iv} & \forall i \in H_{\mathcal{O}}, v \in V_{\mathcal{O}} \\
\displaystyle\sum_{w\in \delta(v)} E^{+}_{ivw} & = & x_{iv} & \forall i \in H_{\mathcal{O}}, v \in V_{\mathcal{O}} \\
\displaystyle\sum_{v\in \delta(w)} E^{-}_{j+1,vw} & = & x_{jw} & \forall j \in H_{\mathcal{E}}, w \in V_{\mathcal{E}} \\
\displaystyle\sum_{v\in \delta(w)} E^{+}_{j-1,vw} & = & x_{jw} & \forall j \in H_{\mathcal{E}}, w \in V_{\mathcal{E}} \\
\displaystyle\sum_{i\in H_{\mathcal{O}}} E^{-}_{ivw} + \sum_{i\in H_{\mathcal{O}}} E^{+}_{ivw} + h_{(v,w)} & \leq & \displaystyle\sum_{i\in H_{\mathcal{O}}} x_{iv} & \forall v \in V_{\mathcal{O}} \\
\displaystyle\sum_{j\in H_{\mathcal{E}}} E^{-}_{j+1,vw} + \sum_{j\in H_{\mathcal{E}}} E^{+}_{j-1,vw} + h_{(v,w)} & \leq & \displaystyle\sum_{j\in H_{\mathcal{E}}} x_{jw} & \forall v \in V_{\mathcal{E}} \\
 & & E_{ivw}, x_{iv}, x_{jw}, h_{(vw)} \ \in \ \{0,1\}. &
\end{array}
$$

Lemma 5. *Backbone constraints imply the connectivity constraints, i.e. constraints* (14) *imply constraints* (9) *and* (4).

Proof: From the backbone constraints, we have:

$$x_{iv} = \sum_{w\in\delta(v)} E^{-}_{ivw}.$$

For each variable $x_{i-1,w}$, we also have:

$$x_{i-1,w} = \sum_{u \in \delta(w)} E^{+}_{i-1,wu}.$$

This last constraint implies that $x_{i-1,w} \geq E^{+}_{i-1,wv}$, since $v \in \delta(w)$. Note that $E^{+}_{i-1,wv} = E^{-}_{ivw}$. For each of terms in the first constraint in this proof, we can obtain the inequality $x_{i-1,w} \geq E^{-}_{ivw}$. Thus, we have the desired inequality:

$$x_{iv} \leq \sum_{w \in \delta(v)} x_{i-1,w}.$$

We can repeat this argument to derive constraint (4). $\square$

Lemma 6. *The optimal solution for LP_3 is at most* $2 * \min\{\mathcal{O}[S], \mathcal{E}[S]\} + 2$.

Proof: The optimal solution for the linear program is $\sum_{(v,w) \in E} h_{(vw)}$. Without loss of generality, we assume $\mathcal{O}[S] \leq \mathcal{E}[S]$. Recall that constraint (15) is in the linear program. We rewrite this constraint as follows:

$$h_{(vw)} \leq \sum_{i \in H_{\mathcal{O}}} x_{iv} - \sum_{i \in H_{\mathcal{O}}} E^{-}_{ivw} - \sum_{i \in H_{\mathcal{O}}} E^{+}_{ivw}.$$

Summing over all the edges, we have:

$$\sum_{(v,w) \in E} h_{(vw)} \leq \sum_{(v,w) \in E} \sum_{i \in H_{\mathcal{O}}} x_{iv} - \sum_{(v,w) \in E} \sum_{i \in H_{\mathcal{O}}} E^{-}_{ivw} - \sum_{(v,w) \in E} \sum_{i \in H_{\mathcal{O}}} E^{+}_{ivw}.$$

The first sum is upper bounded by $4\mathcal{O}[S]$. To show this, first we note that:

$$\sum_{v \in V_{\mathcal{O}}} x_{iv} = 1.$$

If we sum over all edges, as opposed to all odd vertices, note that each odd vertex $v \in V_{\mathcal{O}}$ is an endpoint in at most 4 edges. Thus, we have:

$$\sum_{(v,w)\in E} x_{iv} = \sum_{v\in V_{\mathcal{O}}} \sum_{w\in\delta(v)} x_{iv} = \sum_{w\in\delta(v)} \sum_{v\in V_{\mathcal{O}}} x_{iv} = \sum_{w\in\delta(v)} 1 \leq 4,$$

$$\sum_{(v,w)\in E} \sum_{i\in H_{\mathcal{O}}} x_{iv} = \sum_{i\in H_{\mathcal{O}}} \sum_{(v,w)\in E} x_{iv} \leq \sum_{i\in H_{\mathcal{O}}} 4 = 4\mathcal{O}[S].$$

Now we will analyze the following sum:

$$\sum_{(v,w)\in E} \sum_{i\in H_{\mathcal{O}}, i\neq 1} E^{-}_{ivw} = \sum_{i\in H_{\mathcal{O}}, i\neq 1} \sum_{(v,w)\in E} E^{-}_{ivw}.$$

Each variable E^{-}_{ivw} is associated with a unique odd vertex, i.e. the odd vertex v. We have the following constraints for each odd vertex:

$$\sum_{w\in\delta(v)} E^{-}_{ivw} = x_{iv} \quad \forall i \in H_{\mathcal{O}}, v \in V_{\mathcal{O}}.$$

Thus, we can rewrite the sum as follows:

$$\sum_{i\in H_{\mathcal{O}}, i\neq 1} \sum_{(v,w)\in E} E^{-}_{ivw} = \sum_{i\in H_{\mathcal{O}}, i\neq 1} \sum_{v\in V_{\mathcal{O}}} \sum_{w\in\delta(v)} E^{-}_{ivw} = \sum_{i\in H_{\mathcal{O}}, i\neq 1} \sum_{v\in V_{\mathcal{O}}} x_{iv} = \sum_{i\in H_{\mathcal{O}}, i\neq 1} 1 = \mathcal{O}[S]-1.$$

Note that:

$$\sum_{(v,w)\in E} E^{-}_{ivw} = \sum_{(v,w)\in E} E^{+}_{ivw}.$$

Thus,

$$\sum_{i\in H_{\mathcal{O}}, i\neq 1} \sum_{(v,w)\in E} E^{-}_{ivw} = \sum_{i\in H_{\mathcal{O}}, i\neq n} \sum_{(v,w)\in E} E^{+}_{ivw} = \mathcal{O}[S] - 1.$$

Therefore, we have:

$$\sum_{(v,w)\in E} h_{(vw)} \leq 4\mathcal{O}[S] - (\mathcal{O}[S]-1) - (\mathcal{O}[S]-1) \leq 2\mathcal{O}[S] + 2.$$

So the maximum value of the objective function is $2 * \min\{\mathcal{O}[S], \mathcal{E}[S]\} + 2$. $\square$

Note that this LP will *not* always give a solution whose objective value is at least $2 * \min\{\mathcal{O}[S], \mathcal{E}[S]\}$. It may give a solution whose objective value is strictly better. For example, if we consider the string of 20 consecutive 1's, the objective value is 14.5 according to our AMPL implementation. (See Section 9 for the AMPL Code.)

An alternate formulation for the linear program above would entail using the four index variables $h_{(iv)(jw)}$ instead of the two index variables $h_{(vw)}$.

$$\begin{aligned} E^-_{ivw} + E^+_{ivw} + \sum_{j\in H_\mathcal{E}} h_{(iv)(jw)} &\leq x_{iv} \quad \forall i \in H_\mathcal{O}, (v,w) \in E, \\ E^-_{j+1,vw} + E^+_{j-1,vw} + \sum_{i\in H_\mathcal{O}} h_{(iv)(jw)} &\leq x_{jw} \quad \forall j \in H_\mathcal{E}, (v,w) \in E. \end{aligned} \tag{16}$$

Suppose we substitute constraints (16) for constraints (15). We will refer to the resulting integer and linear program as IP_4 and LP_4, respectively.

IP$_4$:

$$
\begin{aligned}
\max \sum_{(v,w)\in E} \sum_{i\in H_{\mathcal{O}}} \sum_{j\in H_{\mathcal{E}}} h_{(iv)(jw)} & \\
\textit{subject to}: \quad \sum_{v\in V_{\mathcal{O}}} x_{iv} &= 1 \quad \forall i \in H_{\mathcal{O}} \\
\sum_{v\in V_{\mathcal{E}}} x_{jw} &= 1 \quad \forall j \in H_{\mathcal{E}} \\
\sum_{i\in H_{\mathcal{O}}} x_{iv} &\leq 1 \quad \forall v \in V_{\mathcal{O}} \\
\sum_{j\in H_{\mathcal{E}}} x_{jw} &\leq 1 \quad \forall w \in V_{\mathcal{E}} \\
\sum_{w\in\delta(v)} E^{-}_{ivw} &= x_{iv} \quad \forall i \in H_{\mathcal{O}}, v \in V_{\mathcal{O}} \\
\sum_{w\in\delta(v)} E^{+}_{ivw} &= x_{iv} \quad \forall i \in H_{\mathcal{O}}, v \in V_{\mathcal{O}} \\
\sum_{v\in\delta(w)} E^{-}_{j+1,vw} &= x_{jw} \quad \forall j \in H_{\mathcal{E}}, w \in V_{\mathcal{E}} \\
\sum_{v\in\delta(w)} E^{+}_{j-1,vw} &= x_{jw} \quad \forall j \in H_{\mathcal{E}}, w \in V_{\mathcal{E}} \\
E^{-}_{ivw} + E^{+}_{ivw} + \sum_{j\in H_{\mathcal{E}}} h_{(iv)(jw)} &\leq x_{iv} \quad \forall i \in H_{\mathcal{O}}, (v,w) \in E \\
E^{-}_{j+1,vw} + E^{+}_{j-1,vw} + \sum_{i\in H_{\mathcal{O}}} h_{(iv)(jw)} &\leq x_{jw} \quad \forall j \in H_{\mathcal{E}}, (v,w) \in E \\
E_{ivw}, x_{iv}, x_{jw}, h_{(vw)} &\in \{0,1\}
\end{aligned}
$$

Lemma 7. *Suppose S contains no consecutive 1's. Then LP_4 is no stronger than LP_3, i.e. substituting constraints* (16) *for constraints* (15) *does not lead to a stronger relaxation.*

Proof: We can apply the following modification of the proof of Lemma 2. Consider the following table for an arbitrary edge $(v,w) \in E$. Assume there are k i's in $H_{\mathcal{O}}$ labeled $i_1 \ldots i_k$ and assume there are m j's in $H_{\mathcal{E}}$ labeled $j_1 \ldots j_m$. Instead of using x_{iv} and x_{jw} in the bottom row and right column, as we did in the proof of Lemma 2, we use f_{iv} and f_{jw}, which we define below:

$$
\begin{aligned}
f_{iv} &= x_{iv} - E^-_{ivw} - E^+_{ivw}, \\
f_{jw} &= x_{jw} - E^-_{j+1,vw} - E^+_{j-1,vw}.
\end{aligned}
$$

$$
\begin{array}{lllllllll}
 & i\text{:} & 1 & 3 & 5 & \dots & k & & \\
j: & & & & & & & & \\
2 & & h_{(i_1v)(j_1w)} & h_{(i_2v)(j_1w)} & h_{(i_3v)(j_1w)} & \dots & h_{(i_kv)(j_1w)} & \leq & f_{j_1w} \\
4 & & h_{(i_1v)(j_2w)} & h_{(i_2v)(j_2w)} & h_{(i_3v)(j_2w)} & \dots & h_{(i_kv)(j_2w)} & \leq & f_{j_2w} \\
6 & & h_{(i_1v)(j_3w)} & h_{(i_2v)(j_3w)} & h_{(i_3v)(j_3w)} & \dots & h_{(i_kv)(j_3w)} & \leq & f_{j_3w} \\
. & & . & . & . & & . & & . \\
. & & . & . & . & & . & & . \\
. & & . & . & . & & . & & . \\
m & & h_{(i_1v)(j_mw)} & h_{(i_2v)(j_mw)} & h_{(i_3v)(j_mw)} & \dots & h_{(i_kv)(j_mw)} & \leq & f_{j_mw} \\
 & & & & & & & & \\
 & & f_{i_1v} & f_{i_2v} & f_{i_3v} & \dots & f_{i_kv} & &
\end{array}
$$

Note that if there are no consecutive 1's in S, then there will be no $h_{(iv)(jw)}$ variables in the above table in which $j = i+1$ or $j = i-1$. If there were such variables, then they would have to be assigned 0 and we would not be able to apply the proof of Lemma 2. But since all the $h_{(iv)(jw)}$ variables in the table can be non-zero, we can use the same technique as in the proof of Lemma 2.

Note that:

$$
\begin{aligned}
\sum_{i \in H_{\mathcal{O}}} f_{iv} &= \sum_{i \in H_{\mathcal{O}}} (x_{iv} - E^-_{ivw} - E^+_{ivw}), \\
\sum_{j \in H_{\mathcal{E}}} f_{jw} &= \sum_{j \in H_{\mathcal{E}}} (x_{jw} - E^-_{j+1,vw} - E^+_{j-1,vw}).
\end{aligned}
$$

Without loss of generality, assume $\sum_{i \in H_{\mathcal{O}}} f_{iv} \leq \sum_{j \in H_{\mathcal{E}}} f_{jw}$. We want to distribute the value $h_{(v,w)}$ among the $h_{(iv)(jw)}$ variables. We can set the variable $h_{(i_1v)(j_1w)}$ to $\min\{f_{i_1v}, f_{j_1w}\}$. Then we can set the variable $h_{(i_1v)(j_2w)}$ to be as large as possible so that $h_{(i_1v)(j_1w)} + h_{(i_1v)(j_2w)} \leq f_{i_1v}$, etc. We can set all the $h_{(iv)(jw)}$ variables so that their sum equals the sum of the f_{iv} variables. □

Note that if the string S contains consecutive 1's, then the proof of Lemma 7 does not go through. Furthermore, we can construct an example in which LP_3 and LP_4

have different objective values. Figure 4 gives an example in which LP_3 has a higher objective function than that of LP_4.

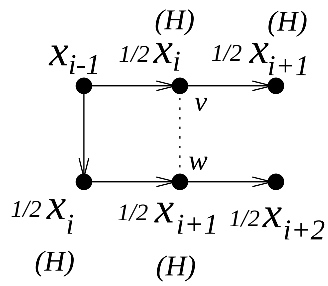

Figure 4. The variable $h_{(v,w)}$ can have value at least $\frac{1}{2}$ in LP_3. In LP_4, the contribution of edge (v,w) would be 0 since the variable $h_{(iv)(i+1,w)}$ is not defined, i.e. it is implicitly 0.

The only difference between LP_3 and LP_4 is that and LP_4 does not allow "contacts" between adjacent elements on the string. Let $f(S)$ represent the number of pairs of consecutive 1's in S. Then the values of LP_3 and LP_4 for a string S are related as follows: $LP_3 \geq LP_4 \geq LP_3$ - $f(S)$. There is no other other benefit to using the 4-index variables rather than the 2-index variables with the current set of constraints.

5 Branch and Bound

Using branch and bound, we would like to branch only on x variables in odd positions or only on x variables in even positions. This would allow us to cut down the number of variables to branch on by a factor of 2. This would be a good approach if the following conjecture holds.

Conjecture 1. *Suppose we have an optimal solution $\{x_{iv}, h_{(v,w)}\}$ for LP_2 such that x_{iv} is integral if i, v are odd. We will call this an* odd integral *solution. Then we can use this solution (e.g. round this solution) to obtain a fully integral solution with the same objective value.*

Given an odd integral solution, we want to show that we can construct a solution with the same objective value in which all the x_{jw} are also integral for even j, w. We have not been able to prove this conjecture. If we consider the path formed by consecutive x_{iv} variables for odd i, we can easily see that it forms a self-avoiding walk on the subset of odd lattice points and for every even j, at most two x_{jw} variables can be non-zero.

Lemma 8. *In an odd-integral solution, at most two x_{jw} can be non-zero when j is even.*

Proof: For all odd i, we have that x_{iv} are integral. Consider x_{ip} and $x_{(i+2)q}$ for some odd i and some p, q such that $x_{ip} = 1$ and $x_{(i+2)q} = 1$. By constraint (4), we have:

$$\sum_{v \in \delta(p)} x_{(i+1)v} \geq x_{ip}.$$

Thus, the total value of x_{i+1} distributed on the four neighbors of p is 1. Similarly, by constraint (9), we have:

$$\sum_{v \in \delta(q)} x_{(i+1)v} \geq x_{(i+2)q}.$$

So p and q must share neighbors and the most neighbors any two points have in common is 2. $\square$

Empirically, we've observed that in odd integral solutions, the value of the x_{jw} variables for even j is usually 0 or $\frac{1}{2}$.

6 Integrality Gaps

We can show that the integrality gap for LP_3 and LP_4 is $2-\epsilon$ for any $\epsilon > 0$. We will use the string $S = \{0\}^q\{01\}^k\{0\}^{2q}\{1000\}^k\{0\}q$. We let k denote a positive integer and $q = 4k^2$. In [8], it is shown that no folding of S has more than $(1+o(1))\mathcal{O}[S]$ contacts. However, we can easily construct a fractional solution for LP_3 for which the objective function is $2\mathcal{O}[S]$.

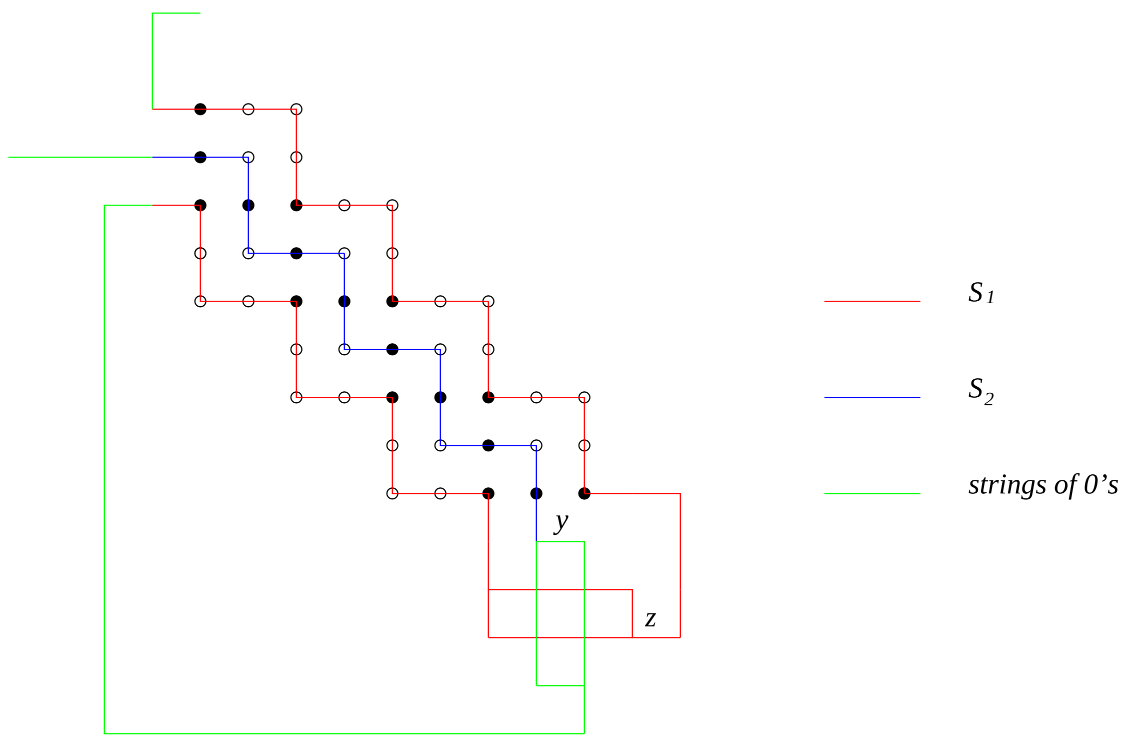

Figure 5. Let $S_1 = \{01\}^k$ and let $S_2 = \{0001\}^k$. The string splits in half at points y and z, which allows the string to cross itself, something not allowed in an integral solution.

7 Six Index Constraints

Another idea for strengthening the linear program is to add *6-index* constraints. One reason to use such constraints is that they would invalidate the solution given in Figure 5.

Suppose we let the variable $h_{(iv)(jw)(ku)}$ be a 1 if there is a contact between i and j on edge (v, w) and between j and k on edge (w, u). Then we can have the following constraint for collinear v, w, u. Recall that n denotes the length of the input string.

$$h_{(iv)(jw)(ku)} = 0 \quad i < k < j : |i - k| < 2 * (n - j).$$

The idea behind this constraint is as follows: Suppose i and k are distance d apart on the string and both form a contact with j. Suppose i, j, k are placed on lattice points $v, w.u$, respectively, where v, w, u are collinear. Then since the string cannot cross itself, the distance from j to the last point on the string n (or the nearest endpoint) must be less than distance $d/2$. If this is not the case, then at some point the substrings $j \ldots n$ and $i \ldots k$ will have to cross each other.

We cannot simply add this constraint to LP_3 or LP_4 because we are not optimizing over *double constraints.* However, this constraint might still be used to obtain information about the optimal folding of a string, because if a string has more than $\mathcal{O}[S]$ contacts, then it must have double contacts. We define a *double contact* as two contacts that are adjacent to each other, i.e. contacts formed on edges (v, w) and (w, u) where v, w, u are either collinear or the two edges form a right angle. In other words, if a folding has more than $\mathcal{O}[S]$ contacts, then some 1's must have more than 1 contact. Thus, we could add the following constraints to LP_3 for all adjacent v, w, u:

$$h_{(iv)(jw)(ku)} \leq h_{(iv)(jw)},$$
$$h_{(iv)(jw)(ku)} \leq h_{(jw)(ku)}.$$

And we could replace the objective function with the following:

$$\max \sum_{i,k \in H_{\mathcal{O}}} \sum_{j \in H_E} \sum_{adjacent\ \ v,w,u} h_{(iv)(jw)(ku)}.$$

Or, alternatively, with:

$$\sum_{i,k \in H_{\mathcal{E}}} \sum_{j \in H_O} \sum_{adjacent\ v,w,u} h_{(iv)(jw)(ku)}.$$

If the LP solution for both of these objective functions were 0, then we would know that an optimal folding contains only $\max\{\mathcal{O}[S], \mathcal{E}[S]\}$ contacts.

8 Future Work

Two of the known approximation algorithms for the folding problem on the 2D lattice [7, 8] have the following property in common. Both algorithms result in a folding in which the original string is divided into two strings and there are only contacts between elements on different strings. In other words, the folding results in two strings S_1 and S_2 such that a contact only occurs between i in S_1 and j in S_2 but never i, j in S_1 or i, j in S_2.

Rounding the LP, e.g. LP_3 or LP_4, seems difficult. An easier approach may be to divide the string S into 2 strings S_1 and S_2 (there are n^2 possibilities) and solve LP_4 only allowing the variables $h_{(iv)(jw)}$ to be non-zero when i is from S_1 and j is from S_2 or vice-versa.

9 Implementation

In this section, we present the ampl code and experimental results for LP_3.

AMPL Code for LP with 2-Index Variables and Flow Constraints (LP_3).

```
param length; #length is the length of the input string S


param feasibleDistance := length/2+2;
param firstAcid := 1;
param lastAcid := firstAcid+length-1;
param middleOddAcid := (floor((lastAcid-firstAcid)/2)-1+firstAcid +
      (floor((lastAcid-firstAcid)/2)-1+firstAcid+1)mod 2);

set Acids := firstAcid .. lastAcid;

param H := 1;
param P := 0;

set SequenceValues := {H,P};

param sequence Acids within SequenceValues;

set O := {i in Acids: (i-firstAcid) mod 2 = 0};
set E := {i in Acids: (i-firstAcid) mod 2 = 1};

set H_O := {i in Acids: (i-firstAcid) mod 2 = 0 and sequence[i]=H};
set H_E := {i in Acids: (i-firstAcid) mod 2 = 1 and sequence[i]=H};

set P_O := {i in Acids: (i-firstAcid) mod 2 = 0 and sequence[i]=P};
set P_E := {i in Acids: (i-firstAcid) mod 2 = 1 and sequence[i]=P};

################lattice#################

param firstX := 1;
param firstY := 1;
```

```
param numX := length+1;
param numY := length+1;

param lastX := firstX+numX-1;
param lastY := firstY+numY-1;

set Xcoord := firstX ..  lastX;
set Ycoord := firstY ..  lastY;

param firstVertex := 1;
param numVertex := numX*numY;
param lastVertex := firstVertex+numVertex-1;

set Vertices := firstVertex ..  lastVertex;

param extractX v in Vertices within Xcoord :=
      ((v-firstVertex) mod numX) + firstX;

param extractY v in Vertices within Ycoord :=
      floor((v-firstVertex)/numX) + firstY;

param extractVertex x in Xcoord, y in Ycoord within Vertices :=
      firstVertex+ (y-firstY)*numX + (x-firstX);

param xdiff v in Vertices, w in Vertices :=
      extractX[v]-extractX[w];

param ydiff v in Vertices, w in Vertices :=
      extractY[v]-extractY[w];

param middleOddX :=
      (floor(numX/2)-1+firstX + (floor(numX/2)+firstX)mod 2);

param middleOddY :=
      (floor(numY/2)-1+firstY + (floor(numY/2)+firstY)mod 2);

param middleOddVertex := extractVertex[middleOddX,middleOddY];

set FeasibleVertices := v in Vertices:
```

```
      abs(xdiff[v,middleOddVertex]) + abs(ydiff[v,middleOddVertex])
      <= feasibleDistance ;

set V_O := v in FeasibleVertices:
      (extractX[v]-firstX+extractY[v]-firstY) mod 2 = 0;

set V_E := v in FeasibleVertices:
      (extractX[v]-firstX+extractY[v]-firstY) mod 2 = 1;

set Neighbors v in FeasibleVertices := w in FeasibleVertices:
      (abs(xdiff[v,w]) + abs(ydiff[v,w])) = 1;

set Edges := (v,w) in V_O cross V_E : w in Neighbors[v];

set OddTriangle :=
      v in V_O: (extractX[v] <= extractX[middleOddVertex]) and
         ((extractX[middleOddVertex]-extractX[v]) <=
         (extractY[middleOddVertex]-extractY[v]));

######################variables######################


var h Edges >=0, <=1;

var e_minus (Acids cross Edges) >= 0, <=1;

var e_plus (Acids cross Edges) >= 0, <=1;

var x_O (O cross V_O) >= 0, <=1;

var x_E (E cross V_E) >= 0, <=1;

######################constraints######################


subject to placeOddElements i in O:
   sum v in V_O x_O[i,v] = 1;

subject to placeEvenElements j in E:
   sum w in V_E x_E[j,w] = 1;
```

```
subject to limitOddVertexLoad v in V_O:
   sum i in O x_O[i,v] <= 1;

subject to limitEvenVertexLoad w in V_E:
   sum j in E x_E[j,w] <= 1;

subject to oddMinusFlow (i,v) in O cross V_O : i != firstAcid:
   sum w in Neighbors[v] e_minus[i,v,w] = x_O[i,v];

subject to oddPlusFlow (i,v) in O cross V_O : i != lastAcid:
   sum w in Neighbors[v] e_plus[i,v,w] = x_O[i,v];

subject to evenMinusFlow (j,w) in E cross V_E : j != lastAcid:
   sum v in Neighbors[w] e_minus[j+1,v,w] = x_E[j,w];

subject to evenPlusFlow (j,w) in E cross V_E : j != firstAcid:
   sum v in Neighbors[w] e_plus[j-1,v,w] = x_E[j,w];

subject to hOddSideWithFlow (v,w) in Edges:
   sum i in H_O : i !=firstAcid e_minus[i,v,w]
   + sum i in H_O : i != lastAcid e_plus[i,v,w]
   + h[v,w] <= sum i in H_O x_O[i,v];

subject to hEvenSideWithFlow (v,w) in Edges:
   sum j in H_E : j != lastAcid e_minus[j+1,v,w]
   + sum j in H_E : j != firstAcid e_plus[j-1,v,w]
   + h[v,w] <= sum j in H_E x_E[j,w];

subject to fixMiddleOddVertex:
   x_O[middleOddAcid,middleOddVertex] = 1;

subject to fixFirstVertex:
   sum v in OddTriangle x_O[firstAcid,v] = 1;

#####################objective function #####################

minimize contacts:  - sum (v,w) in Edges h[v,w];
```

Experimental Results

We ran LP_3 on some of the benchmarks for the problem in the 2D HP model. These were taken from: www.cs.sandia.gov/tech_reports/compbio/tortilla-hp-benchmarks.html.

We ran the LP's on the following strings:

1. hphpphhphpphphhpphph
2. hhpphpphpphpphpphpphpphh
3. pphpphhpppphhpppphhpppphh
4. ppphhpphhppppphhhhhhhpphhpppphhpphpp
5. pphpphhpphhppppphhhhhhhhhhpppppphhpphhpphpphhhhh
6. hhhpphphphpphphphpph

String	length	upper bound	LP_3	Opt
1	20	11	10.67529996	9
2	24	11	11	9
3	25	8	8	8
4	36	16	14.89908257	14
5	48	25	24.88770748	22
6	20	11	10.76264643	10

References

[1] "Protein Foldings and the Thermodynamic Hypothesis, 1950-1962", *http://profiles.nlm.nih.gov/KK/Views/Exhibit.*

[2] R. Agarwala, S. Batzoglou, V. Dancik, S. Decatur, M. Farach, S. Hannenhalli, S. Muthukrishnan and S. Skiena, "Local Rules for Protein Folding on a Triangular Lattice and Generalized Hydrophobicity in the HP Model", *Journal of Computational Biology* (1997) Vol. 4(2):275-296.

[3] Christian Antifsen, Robert R. Redfield, Warren I. Choate, Juanita Page and William R. Carroll, "Studies on the Gross Structure, Cross-Linkages, and terminal Sequences in Ribonuclease", *Journal of Biological Chemistry* Vol. 207, No. 1 (March 1954):201-210.

[4] K. A. Dill, "Theory for the Folding and Stability of Globular Proteins", *Biochemistry* (1985) Vol. 24:1501.

[5] K. A. Dill, "Dominant Forces in Protein Folding, *Biochemistry* (1990) Vol. 29:7133-7155.

[6] H. J. Greenberg, W. E. Hart, and G. Lancia, "Opportunities for Combinatorial Optimization in Computational Biology", *INFORMS Journal of Computing*, *To appear.*

[7] William Hart and Sorin Istrail, "Fast Protein Folding in the Hydrophobic-Hydrophilic Model within Three-Eighths of Optimal", *Journal of Computational Biology* Vol. 3, No. 1, 1996:53-96.

[8] Alantha Newman, "A New Algorithm for Protein Folding in the HP Model", *Proceedings of SODA, 2002*, 876-884.