A New Algorithm for Protein Folding in the HP Model

Alantha Newman *

Abstract

We consider the problem of protein folding in the HP model on the two-dimensional square lattice. This problem is combinatorially equivalent to folding a string of 0's and 1's so that the string forms a self-avoiding walk on the lattice and the number of adjacent pairs of 1's is maximized. We present a linear-time 1/3-approximation algorithm for this problem, improving on the previous best approximation factor of 1/4. The approximation guarantee of this algorithm is based on an upper bound presented by Hart and Istrail [6] and used in all previous papers that address this problem. We show that this upper bound cannot be used to obtain an approximation factor better than 1/2.

1 Introduction

In this paper, we study the problem of folding a string of 0's and 1's on the two-dimensional (2D) square lattice. The goal is to find a folding of the string that forms a self-avoiding walk on the lattice and maximizes the number of adjacent pairs of 1's, also called *contacts*. For example, suppose we are given the string 101010101001010101. Then the folding shown in Figure 1 results in eight pairs of adjacent 1's or eight contacts (shown by the dotted lines), the most possible for this string. The 1's are denoted by black dots and the 0's are denoted by white or unfilled dots.

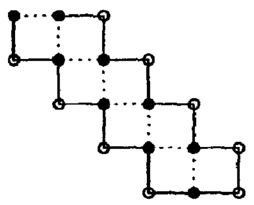

Figure 1: An optimal folding for the string 101010101001010101.

*Laboratory for Computer Science, MIT, Cambridge, MA 02139. Supported in part by NSF Grant CCR-9912342 and NSF Career Award CCR-9875024. Email: alantha@theory.lcs.mit.edu

1.1 Background

The widely-studied HP model was introduced by Dill [4, 5]. A protein is a chain of amino acid residues. In the HP model, each amino acid residue is classified as an H (hydrophobic or non-polar) or a P (hydrophilic or polar). An optimal conformation for a string of amino acids in this model is one that has the lowest energy, which is achieved when the maximum number of H-H contacts (i.e. pairs of H's that are adjacent in the folding but not in the string) are present. The *protein folding* problem in the hydrophobic-hydrophilic (HP) model on the 2D square lattice is combinatorially equivalent to the problem we just described: we are given a string of P's and H's (instead of 0's and 1's) and we wish to maximize the number of adjacent pairs of H's (instead of 1's). An informative discussion on the HP model and its applicability to protein folding is contained in the paper by Hart and Istrail [6].

The protein folding problem in the HP model on the 2D square lattice is NP-hard [3] as is the problem on the 3D square lattice [2]. On the positive side, Hart and Istrail gave a 1/4-approximation for the problem on the 2D square lattice and a 3/8-approximation for the problem on the 3D square lattice [6]. Mauri, Piccolboni, and Pavesi gave a different approximation algorithm for the problem on the 2D square lattice [7]. Their algorithm also has an approximation ratio of 1/4, but they argue that in practice the performance of their algorithm surpasses its worst-case guarantee. Improving on the factor of 1/4 has been an open problem in computational biology for several years.

In related work, Agarwala et al. gave approximations for the problem on the 2D and 3D triangular lattices [1]. They suggest that the triangular lattice is a more realistic lattice for protein folding because it does not have the "parity problem", i.e. in the square lattice, two residues must be an odd distance (of at least three) apart to be in contact with each other in any folding. They argue that this is not the case for foldings of real proteins.

1.2 Our Results

In this paper, we present a 1/3-approximation for the protein folding problem in the HP model on the 2D square lattice. We describe our linear-time algorithm in

Section 2. In Section 3, we show that using the upper bound discussed in Section 2, the best approximation factor that can be achieved is 1/2. Specifically, we describe a string for which the optimal folding achieves only half of this upper bound.

2 A $\frac{1}{3}$-Approximation Algorithm

We now present a 1/3-approximation algorithm for the problem of folding a string of 0's and 1's on the 2D square lattice so as to maximize the number of contacts. In Section 2.1, we introduce some notation. In Section 2.2, we discuss the upper bound that we use to obtain the 1/3-approximation. In Section 2.3, we state some assumptions and lemmas that are useful for our algorithm. In Section 2.4, we state the algorithm itself, and in Section 2.5 we analyze the approximation guarantee and the running time.

2.1 Some Notation

For a string $S \in \{0,1\}^n$ such that $S = s_1 \dots s_n$, we refer to a 1 in an odd position (i.e. a 1 with index i for odd i) as an *odd-1* and a 1 in an even position as an *even-1*. We call each 0 or 1 in the string an *element*. For any substring s of S, i.e. $s = s_j \dots s_k$ for some $j \leq k$, we let $\mathcal{E}[s]$ denote the number of even-1's in s and $\mathcal{O}[s]$ denote the number of odd-1's in s.

2.2 An Upper Bound

On the square lattice, an even-1 can only be adjacent to odd-1's and vice versa, i.e. two even-1's or two odd-1's cannot be adjacent to each other. Each element in the string except for those at the two endpoints of the string, can have at most two *topological neighbors*, i.e. elements that are adjacent in the folding but are not adjacent in the string. Thus, an upper bound M on the total number of pairs of adjacent 1's or contacts is:

$$M = 2 * \min(\mathcal{E}[S], \mathcal{O}[S])$$

This upper bound was used in [6] and [7]. In those papers, it was shown that for every binary string, there exists a folding in which either half of the odd-1's or half of the even-1's have at least one contact on average, resulting in a 1/4-approximation. The approximation guarantee of 1/3 for our algorithm is obtained by showing that at least 2/3 of the odd-1's or 2/3 of the even-1's average at least one contact each. In Section 3, we describe a string for which the optimal folding achieves only one contact on average for the minimum of the odd-1's and even-1's, which is 1/2 of this upper bound.

2.3 Preliminaries

Without loss of generality, we make the following assumptions about any binary string S that we wish to fold.

(i) The length of S is even (we can pad S with an extra 0 if this is not already the case).

(ii) The number of odd-1's is equal to the number of even-1's. If one appears more than the other, say $\mathcal{O}[S] > \mathcal{E}[S]$, we can turn an arbitrarily chosen subset of $\mathcal{O}[S] - \mathcal{E}[S]$ odd-1's into 0's.

Note that neither of these assumptions changes the value of the upper bound, since $(\min(\mathcal{O}[S], \mathcal{E}[S])$ remains the same. For the sake of convenience, we consider folding a *loop* rather than a string. That is, given a string (which has even length by the assumption above), we connect the first and last elements so that s_1 and s_n are adjacent in the loop. Note that the upper bound stated in Section 2.2 is also a valid upper bound for the number of contacts that can be obtained by folding a loop. Since the loop is closed, we need to demarcate which 1's are odd-1's and which are even-1's. It suffices to choose any 1, call it an odd-1, call every 1 an even distance away from this 1 an odd-1, and call the rest of the 1's even-1's.

LEMMA 2.1. *Let S be a string such that if we join the endpoints, we obtain the loop L. Then a folding of L resulting in αM contacts also yields at least αM contacts for S.*

Proof. Consider any folding of L with k contacts. Any string that is obtained by disconnecting two adjacent elements of L can assume the same configuration as this folding. So this configuration also yields at least k contacts for such a string. □

The following combinatorial observation plays a key role in the algorithm.

LEMMA 2.2. *In any loop L that contains an equal number of odd-1's and even-1's, there is an element $p = s_i$ such that if we go around L in one direction (i.e. clockwise or counter-clockwise) starting at s_i to any point s_j, then $\mathcal{O}[s_i s_{i+1} \dots s_j] \geq \mathcal{E}[s_i s_{i+1} \dots s_j]$, and if we go around L in the other direction from s_{i-1} to any point s_k, then $\mathcal{E}[s_{i-1} s_{i-2} \dots s_k] \geq \mathcal{O}[s_{i-1} s_{i-2} \dots s_k]$.*

Proof. Let $S = s_1 \dots s_n$ be some even length string such that joining the endpoints results in the loop L. Let $f(j) = \mathcal{O}[s_1 \dots s_j] - \mathcal{E}[s_1 \dots s_j]$. In other words, $f(j)$ is the number of odd-1's minus even-1's in the string $s_1 \dots s_j$. Then let j^* be a point such that $f(j^*)$ is minimum. For example, in Figure 2, the function $f(j)$ is shown for the string 110101101011 and $j^* = 6$ for

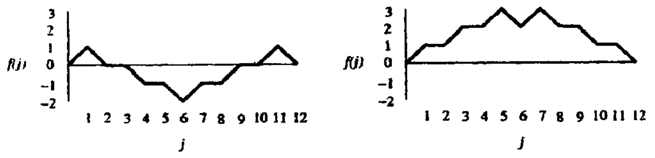

Figure 2: The graph of the function $f(j)$ for the string $s = 110101101011$ and the string $s' = 101011110101$.

this string. Note that we can always assume j^* is even since there is always an even j^* where $f(j)$ is minimum. To see this, assume j^* is odd. Then if $s_{j^*+1} = 1$, it must be that $f(j^*+1) < f(j^*)$ and if $s_{j^*+1} = 0$, then $f(j^*+1) = f(j^*)$. So we can assume j^* is even.

Now consider the string $s'_1 \dots s'_n$ such that $s'_1 = s_{j^*+1}$ and $s'_2 = s_{j^*+2}$, etc. Note that since j^* is even, j^*+1 is odd. Thus, s'_1 is an odd-1 in both the string S and the new string $s'_1 \dots s'_n$. For the string in Figure 2, we have $s'_1 \dots s'_n = 101011110101$. The function $f(j)$ for this new string s' is always positive. Thus, $\mathcal{O}[s'_1 s'_2 \dots s'_j] \geq \mathcal{E}[s'_1 s'_2 \dots s'_j]$ for any s'_j in this new string. If we consider the reverse string $s'_n \dots s'_1$, then it is always the case that $\mathcal{E}[s'_n s'_{n-1} \cdots s'_j] \geq \mathcal{O}[s'_n s'_{n-1} \cdots s'_j]$ for any point s'_j. Thus, if we start at point s'_1 and go through s'_2 and s'_3, etc., to any point s'_j, it is always the case that $\mathcal{O}[s'_1 s'_2 \dots s'_j] \geq \mathcal{E}[s'_1 s'_2 \dots s'_j]$. And if we start at point s'_n and go through s'_{n-1} and s'_{n-2}, etc., to any point s'_j, it is always that case that $\mathcal{E}[s'_n s'_{n-1} \dots s'_j] \geq \mathcal{O}[s'_n s_{n-1} \dots s'_j]$. Thus, the theorem is true when $p = s'_1$. □

Using Lemma 2.2, without loss of generality, we assume that for any loop L, if we start at point p and move in the clockwise direction, we will see at least as many odd-1's as even-1's, and vice versa for the counter-clockwise direction. Let the i^{th} odd-1 be the i^{th} odd-1 found if we start at point $p+1$ and go along L in the clockwise direction. Let the i^{th} even-1 be the i^{th} even-1 found if we start at point $p-2$ and move along L in the counter-clockwise direction. Define $B_{\mathcal{O}}(i)$ to be the substring from the element directly following the $i-1^{th}$ odd-1 up to and including the i^{th} odd-1. And define $B_{\mathcal{E}}(i)$ to be the substring from the element directly following the $i-1^{th}$ even-1 up to and including the i^{th} even-1. Let the length of $B_{\mathcal{O}}(i)$ be $\ell_{\mathcal{O}}(i)+1$ and the length of $B_{\mathcal{E}}(i)$ be $\ell_{\mathcal{E}}(i)+1$. Note that $\ell_{\mathcal{E}}(i)$ and $\ell_{\mathcal{O}}(i)$ are always odd integers. For example, given the loop corresponding to the string $S = 11010110100011$, where $p = s_7$, we have that $B_{\mathcal{O}}(1) = 01$, $B_{\mathcal{O}}(2) = 0001$, $B_{\mathcal{O}}(3) = 11$. We also have that $B_{\mathcal{E}}(1) = 01$, $B_{\mathcal{E}}(2) = 01$, and $B_{\mathcal{E}}(3) = 11$. See Figure 3 for an illustration.

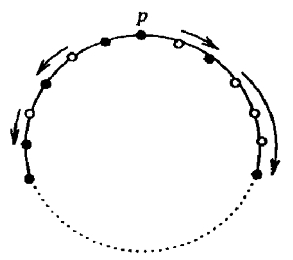

Figure 3: Moving clockwise from p, we have the substrings $B_{\mathcal{O}}(1)$, $B_{\mathcal{O}}(2)$, etc. Moving counter-clockwise from p, we have the substrings $B_{\mathcal{E}}(1)$, $B_{\mathcal{E}}(2)$, etc.

2.4 The Algorithm

We now describe our algorithm. Our goal is to find a folding of a given string so as to maximize the number of pairs of adjacent 1's. Using Lemma 2.1, we assume that we are given a loop to fold. Using Lemma 2.2, we find a point p, such that if we go around the loop L in the clockwise direction from p, we always see at least as many odd-1's as even-1's and if go around L in the counter-clockwise direction, we always see at least as many even-1's as odd-1's.

1. Start: Lay p and $p+1$ and their adjacent elements as shown in Figure 4. Let $i = j = 1$.

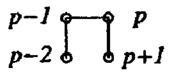

Figure 4: Placement of elements $p-2, \dots p+1$.

2. Iteration: Consider $B_{\mathcal{E}}(i)$ and $B_{\mathcal{O}}(j)$. There are four cases.

(a) $\ell_{\mathcal{E}}(i) = 1$ and $\ell_{\mathcal{O}}(j) = 1$: Fold $B_{\mathcal{E}}(i), B_{\mathcal{E}}(i+1), B_{\mathcal{O}}(j)$, and $B_{\mathcal{O}}(j+1)$ as in Figures 5(a) and 6(a). Set $i = i+2$ and $j = j+2$. The idea is to make sure there are 3 contacts: one between the i^{th} even-1 and j^{th} odd-1, one between the $i+1^{th}$ even-1 and j^{th} odd-1, and one between the $i+1^{th}$ even-1 and $j+1^{th}$ odd-1.

(b) $\ell_{\mathcal{E}}(i) \geq 3$ and $\ell_{\mathcal{O}}(j) \geq 3$: Fold $B_{\mathcal{E}}(i), B_{\mathcal{E}}(i+1), B_{\mathcal{O}}(j)$, and $B_{\mathcal{O}}(j+1)$ as in Figures 5(b) and 6(b). Set $i = i+2$ and $j = j+2$. The idea is that same as in case (a), except we must move the segments $B_{\mathcal{E}}(i)$ and $B_{\mathcal{O}}(j)$ out of the way if either $\ell_{\mathcal{E}}(i) \geq 3$ or $\ell_{\mathcal{O}}(j) \geq 3$.

(c) $\ell_{\mathcal{E}}(i) = 1$ and $\ell_{\mathcal{O}}(j) \geq 3$: Fold $B_{\mathcal{E}}(i), B_{\mathcal{O}}(j)$, and $B_{\mathcal{O}}(j+1)$ as in Figures 5(c) and 6(c). Set $i = i+1$

and $j = j + 2$. The idea is to make sure there are 2 contacts: one between the i^{th} even-1 and the j^{th} odd-1 and one between the i^{th} even-1 and the $j+1^{th}$ odd-1.

(d) $\ell_{\mathcal{E}}(i) \geq 3$ and $\ell_{\mathcal{O}}(j) = 1$: Fold $B_{\mathcal{E}}(i), B_{\mathcal{E}}(i+1)$, and $B_{\mathcal{O}}(j)$ as in Figure 5(d) and in the mirror image of Figure 6(c). Set $i = i+2$ and $j = j+1$. The idea here is the same as in case (c) except here there are 2 contacts for the j^{th} odd-1 and one contact for the i^{th} and $i+1^{th}$ even-1.

3. Repeat Step 2 while $B_{\mathcal{E}}(i)$ and $B_{\mathcal{O}}(j)$ do not overlap.

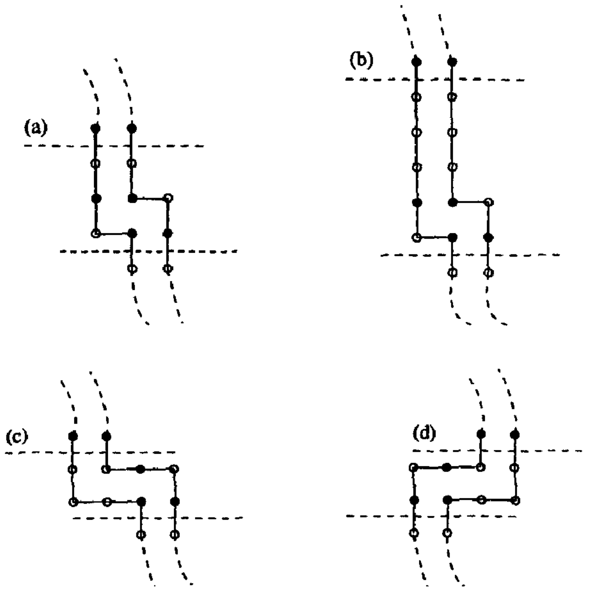

Figure 5: Case (a), (b), (c), and (d) folds.

In Figure 5, we show how to fold the string when $\ell_{\mathcal{E}}(i+1) = 1$ and $\ell_{\mathcal{O}}(j+1) = 1$. For larger values of $\ell_{\mathcal{E}}(i+1)$ and $\ell_{\mathcal{O}}(j+1)$, i.e. when $\ell_{\mathcal{E}}(i+1) \geq 3$ or $\ell_{\mathcal{O}}(j+1) \geq 3$, we fold the string as shown in Figure 6. For a case (b) fold, Figure 5 shows how to fold the string when $\ell_{\mathcal{E}}(i) = 3$ and $\ell_{\mathcal{O}}(j) = 3$ and Figure 6 shows how to fold the string when $\ell_{\mathcal{E}}(i) \geq 5$ or $\ell_{\mathcal{O}}(j) \geq 5$.

2.5 Analysis

THEOREM 2.1. *The algorithm finds a folding with at least M/3 contacts, i.e. a 1/3-approximation.*

Proof. Without loss of generality, assume there are k more case (c) folds than case (d) folds, where $k \geq 0$. We will count how many contacts the odd-1's are involved in. (If there are more case (d) folds than case (c)

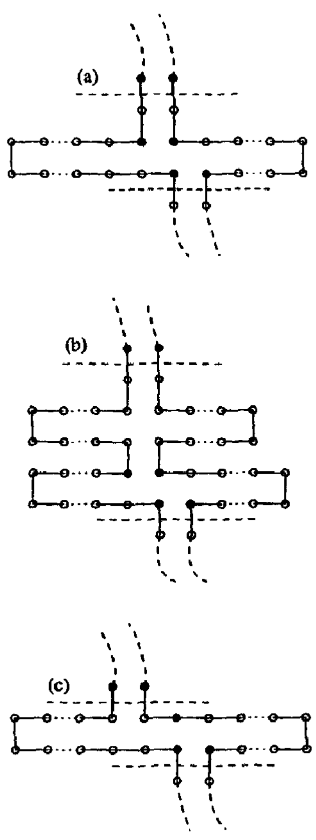

Figure 6: Foldings for higher values of $\ell_{\mathcal{E}}(i)$ and $\ell_{\mathcal{O}}(j)$.

folds, we would count how many contacts the even-1's are involved in.) Consider the folding of a string S found by the algorithm. Let i^* and j^* be the value of i and j during the last iteration of the algorithm. Then $\mathcal{O}[p+1, p+2, \ldots j^*]$ denotes the number of odd-1's that are guaranteed to be used in some contact(s). How many odd-1's are not necessarily in any contacts? The odd-1's in the string $p-2, p-3, \ldots i^*$ are not necessarily used in any contacts. By Lemma 2.2, we have:

(2.1) $$\mathcal{O}[p-2, p-3, \ldots i^*] \leq \mathcal{E}[p-2, p-3, \ldots i^*]$$

(2.2) $$\mathcal{O}[S] = \mathcal{O}[p+1, p+2, \ldots j^*] + \mathcal{O}[p-2, p-3, \ldots i^*]$$

Combining equations (2.1) and (2.2), we have:

(2.3) $$\mathcal{O}[S] \leq \mathcal{O}[p+1, p+2, \ldots j^*] + \mathcal{E}[p-2, p-3, \ldots i^*]$$

We assumed that there are k more case (c) folds than case (d) folds. Let's pair up each case (d) fold with a case (c) fold and call each of these pairs a (c-d)-fold. Thus, the number of odd-1's used in case (a), case (b), or (c-d) folds is $\mathcal{O}[p+1, p+2, \ldots j^*] - 2k$ and $2k$ odd-1's are used in unpaired case (c) folds, since each case (c) fold uses 2 odd-1's. The number of even-1's used in case (a), case (b), or case (c-d) folds is also $\mathcal{O}[p+1, p+2, \ldots j^*] - 2k$, since in these folds the number of even-1's used is the same as the number of odd-1's. Then there are k even-1's used in the extra case (c) folds. Thus,

$$\text{(2.4)} \qquad \mathcal{E}[p-2, p-3, \ldots i^*] = \mathcal{O}[p+1, p+2, \ldots j^*] - k$$

Combining (2.3) and (2.4), we have:

$$\text{(2.5)} \qquad \mathcal{O}[S] \le (\mathcal{O}[p+1, p+2, \ldots j^*]) + (\mathcal{O}[p+1, p+2, \ldots j^*] - k)$$

Equation (2.5) can be rewritten as:

$$\text{(2.6)} \qquad \mathcal{O}[p+1, p+2, \ldots j^*] \ge \frac{\mathcal{O}[S]}{2} + \frac{k}{2}$$

If we consider the subset of the odd-1's in the string $p+1, p+2, \ldots j^*$ involved in case (a), case (b), or (c-d) folds, we note that there are at least 4 contacts for every 3 odd-1's. (i.e. In case (a) and case (b) folds, we have 3 contacts for every 2 odd-1's, and in case (c-d) folds, we have 4 contacts for every 3 odd-1's.) In the unpaired case (c) folds, we have at least one contact for every odd-1. Thus, the number of contacts we have is at least:

$$\text{(2.7)} \qquad \frac{4}{3}(\mathcal{O}[p+1, p+2, \ldots j^*] - 2k) + 2k$$

Using equation (2.6), we have that the quantity in equation (2.7) is at least:

$$\text{(2.8)} \qquad \left(\frac{\mathcal{O}[S]}{2} - \frac{3k}{2}\right)\frac{4}{3} + 2k = \frac{2\mathcal{O}[S]}{3}$$

Recall that $\mathcal{O}[S] = \mathcal{E}[S]$ by assumption, which implies that $M = 2\mathcal{O}[S]$. Therefore, the number of contacts that the algorithm achieves is at least $M/3$. □

The algorithm runs in $O(n)$ time where n is the length of L. We can find point p in $O(n)$ time. Finding $B_{\mathcal{E}}(i)$ and $B_{\mathcal{O}}(i)$ and folding these blocks takes time proportional to the size of the blocks, but since each element is included in only one of the blocks, the total time it takes to find all the blocks and fold them is $O(n)$.

3 How Good is the Upper Bound?

In this section, we examine the upper bound presented in Section 2.2. Recall that the upper bound is $M = 2 * \min(\mathcal{O}[S], \mathcal{E}[S])$. How good is this bound? In the previous section, we saw that $OPT/M \ge 1/3$ for any string. In this section, we describe a string for which $OPT/M = 1/2 + o(1)$. Thus, this upper bound cannot be used to obtain an approximation factor better than 1/2.

Let $\hat{S} = \{0\}^{4k^2}\{01\}^k\{0\}^{8k^2}\{1000\}^k\{0\}^{4k^2}$ for an integer $k > 0$. We will show that no folding of $\hat{S}$ has more than $(1 + o(1))M/2$ contacts.

Theorem 3.1. *No folding of $\hat{S}$ results in more than $(1 + o(1))M/2$ contacts.*

Note that for the string $\hat{S}$, there are k even-1's and k odd-1's. Thus, $k = M/2$ for the string $\hat{S}$. So we need to show that no folding contains more than $(1+o(1))k$ contacts. To prove Theorem 3.1, we consider two strings S_1 and S_2 such that $\hat{S} = S_1S_2$. Let $S_1 = \{0\}^{4k^2}\{01\}^k\{0\}^{4k^2}$ and let $S_2 = \{0\}^{4k^2}\{1000\}^k\{0\}^{4k^2}$. All the 1's in S_1 are even-1's and all the 1's in S_2 are odd-1's. Note that since all the 1's in S_1 are even-1's, no folding of $\hat{S}$ contains a contact between a pair of 1's from S_1. Similarly, no folding of $\hat{S}$ contains a contact between a pair of 1's from S_2, since all the 1's in S_2 are odd-1's. Thus, we can assume that all contacts are comprised of an even-1 from S_1 and an odd-1 from S_2. Therefore, it suffices to show that no folding of $\hat{S}$ results in more than $(1+o(1))k$ contacts *between* the two strings S_1 and S_2.

Since we are only concerned with contacts between the strings S_1 and S_2, we focus on foldings of these two strings rather than on foldings of $\hat{S}$. Note that for any folding of $\hat{S}$, there is a folding of S_1 and S_2 that has at least as many contacts as the folding of $\hat{S}$. This is because S_1 and S_2 are substrings of $\hat{S}$. Thus, proving that no folding of the two strings S_1 and S_2 results in more than $(1 + o(1))k$ contacts would prove Theorem 3.1.

Suppose that for each of the strings S_1 and S_2, we color one side red and the other side blue. Alternatively, we can imagine that the strings S_1 and S_2 are each enclosed within two adjacent strings, one red and one blue. A contact is a red-red contact if the red sides (or red strings) face each other in the contact, or a red-blue contact if one red side (or red string) faces a blue side (or blue string) in the contact. Some examples of red-red contacts are illustrated in Figure 7. There are four types of contacts if we always consider the color of the S_1 string first: red-red, red-blue, blue-red, and blue-blue. We now show that it is only possible to have one type

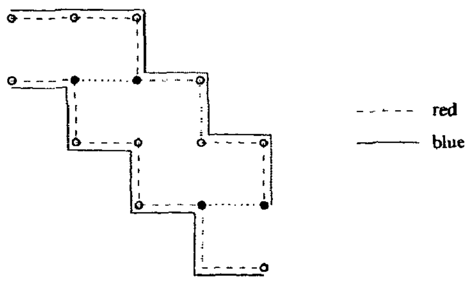

Figure 7: Some red-red contacts.

of contact between S_1 and S_2 in any folding. In other words, if some contact is a red-red contact, then all the contacts must be red-red contacts. Thus, we only have to consider foldings in which all contacts are of one type. If an odd-1 is involved in 2 contacts, both must be with even-1's on the same side of the odd-1. For example, we can ignore contacts such as those shown in Figure 8.

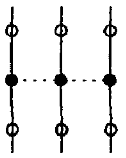

Figure 8: These two contacts are each of a different type.

Lemma 3.1. *In any folding of $\hat{S}$, all contacts between S_1 and S_2 are of the same type.*

Proof. Assume for the sake of contradiction that there is some folding of S_1 and S_2 with at least two different types of contacts (of the four possible types). Let c_1 be a red-red contact and c_2 be a blue-blue contact, as shown in Figure 9. Suppose c_1 is a contact between x_1 and y_1 where x_1 is an even-1 in S_1 and y_1 is an odd-1 in S_2. Similarly, c_2 is a contact between x_2 and y_2, where x_2 is an even-1 in S_1 and y_2 is an odd-1 in S_2.

Then there is a closed path from y_1 to y_2 along S_2, from y_2 to x_2, from x_2 to x_1 along S_1 and from x_1 back to y_1. Note that the farthest distance between any two 1's is $2k-1$ in S_1 and $4k-1$ in S_2. Thus, the total length of this closed path is no more than $6k$. However, as shown in Figure 9, at least one of the substrings of 0's at the end of S_1 or S_2 is enclosed by this path. The number of 0's in this substring is $4k^2$. But this is a contradiction, because the maximum number of

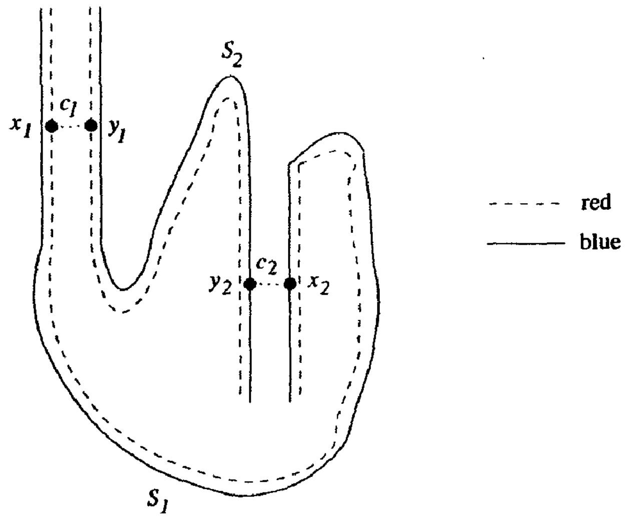

Figure 9: One way to connect a red-red and a blue-blue contact.

lattice points that can fit an enclosed area of perimeter $6k$ is $9k^2/4$. We obtain the same contradiction for the other possible arrangement of a red-red and a blue-blue contact as shown in Figure 10. Furthermore, we obtain the same contradiction for any arrangement of a red-red contact and a red-blue contact or a red-red contact and a blue-red contact. This can be verified by inspecting the two cases (which are similar to the two possible arrangements of a red-red and a blue-blue contact) corresponding to each of these pairs of different types of contacts. □

Now we consider the case in which all contacts are of the same type. Without loss of generality, we assume they are all red-red.

Lemma 3.2. *There are at most $(1+o(1))k$ red-red contacts between S_1 and S_2.*

Proof. We will show that the average number of contacts per even-1 cannot exceed $(1+o(1))$. We note that if an even-1 has two contacts, then its two contacts much be perpendicular to each other since both are red-red contacts.

We will use the following notions in our proof. First, we assume that the contacts are ordered consecutively in a folding of S_1 and S_2. We begin folding S_1 and S_2 by considering a specific endpoint for each of these strings. Then the even-1 in S_1 and the odd-1 in S_2 that are closest to these endpoints and that make a contact are in the first contact. The next closest even-1 in S_1

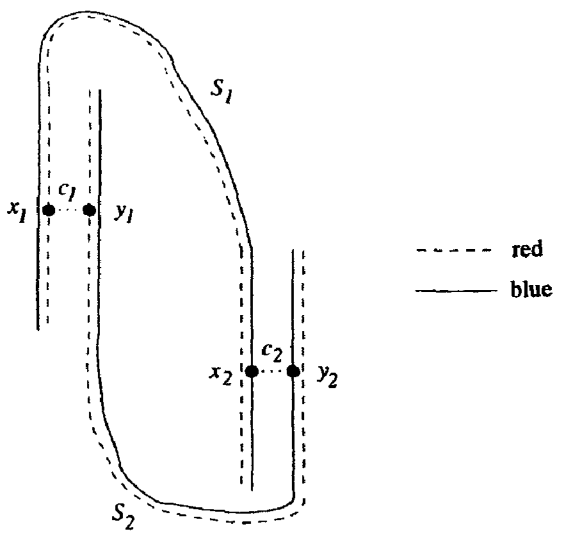

Figure 10: Another way to connect a red-red and a blue-blue contact.

and the next closest odd-1 in S_2 that make a contact are in the second contact, etc. Note that consecutive contacts, e.g. the two contacts involving x in Figure 11, may involve the same odd-1 or same even-1.

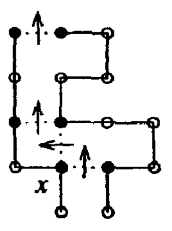

Figure 11: The arrows indicate the orientation of each contact.

Next, we associate an orientation with each contact. A contact can have orientation up, down, right, or left. For example, in Figure 11, say that the horizontal contact involving the even-1 labeled x is the first contact in the folding, and the vertical contact involving x is the second, etc. Then the first contact has orientation up and the second contact has orientation left. A horizontal contact has orientation up if the next contact is above it and down if the next contact is below it. A vertical contact has orientation left if the next contact is to its left and right if the next contact is to its right. Note that the orientation of a contact is only well-defined if the next even-1 or the next odd-1 in the string are

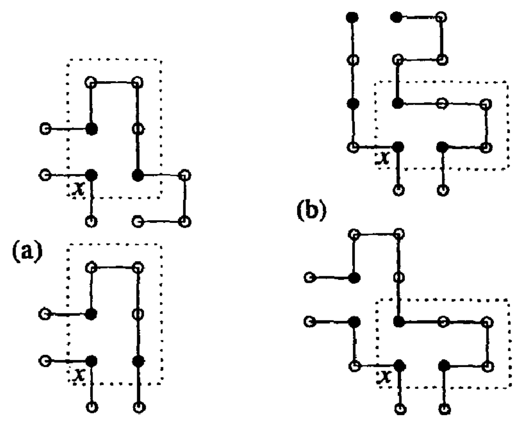

Figure 12: x is the first even-1 in the folding with two contacts.

involved in a contact.

Let x be the first even-1 to have two contacts. Without loss of generality, assume that the first of these contacts is oriented up. The two possibilities for this situation are shown in Figure 12. First, we consider case (a) in Figure 12. Say that x has an up and a left contact as in case (a). If the next even-1 also has two contacts, then its second contact will have a down orientation as shown in Figure 13(a). If the next even-1 has only one contact, but the next *next* even-1 has two contacts, then *its* second contact will have a down orientation, as shown in Figure 13(b). In other words, consider the next even-1 (call it y) after x that has contacts with two odd-1's. If all the even-1's between x and y have at least one contact, then the orientation of y's contacts makes a counter-clockwise turn. If some even-1 between x and y does not have any contacts, then the second contact of y may have a left orientation. So in this case, we are not in a downward orientation (i.e. we have not made a counter-clockwise turn), but we do not have more than one contact per even-1 on average for the set of even-1's between x and y.

If the next even-1 after x has only one contact, it can have a left or a down orientation, but it cannot have an up orientation. In order for a contact to have an up orientation, we need to make a clockwise turn. However, for every clockwise turn, there will be two even-1's with no contacts. To see this, consider Figure 14. Now suppose r and s make a contact as shown in Figure 14. Note that r can be in the same situation as x is in in Figure 12(a) or (b). If r is in case (a) and we make another clockwise turn and then go back to case (a), etc., then we will average less than 1 contact for each even-1. If r is in the same position as x in Figure 12(b),

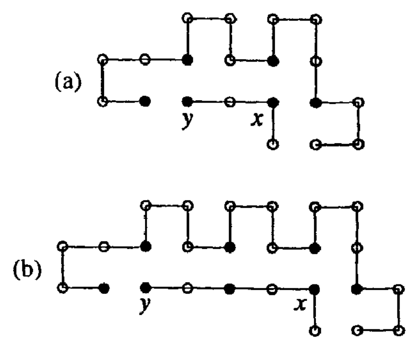

Figure 13: y is the next even-1 after x to have two contacts.

then we can make a counter-clockwise turn so that the next two even-1's will have two contacts each. But in this case, we will average only one contact per even-1 over the course of a counter-clockwise and clockwise turn.

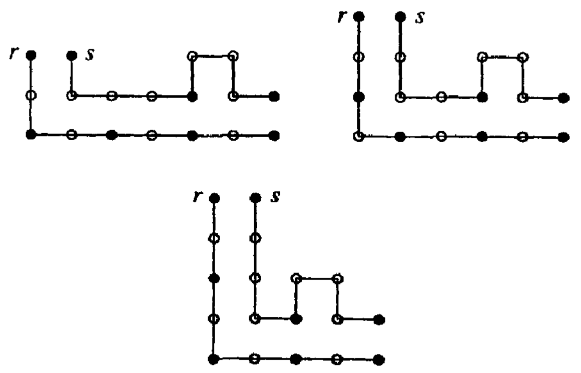

Figure 14: If the orientation of the contacts makes a clockwise turn, then two even-1's have no contacts.

Next, we consider case (b) in Figure 12(b). If x is in case (b), then the even-1 that follows x will have one contact as shown in the first figure in Figure 12(b) or it will have two contacts and be in the same position as x is in in case (a). Thus, if we start from case (b), we can get only one more contact than if we were to start in case (a).

Therefore, the only way to fold the string so that a constant fraction of the even-1's are contained in more than one contact is to have more counter-clockwise turns than clockwise turns. In this case, the string forms a "spiral", as shown in Figure 15. Every time we make a counter-clockwise turn in this configuration, we can

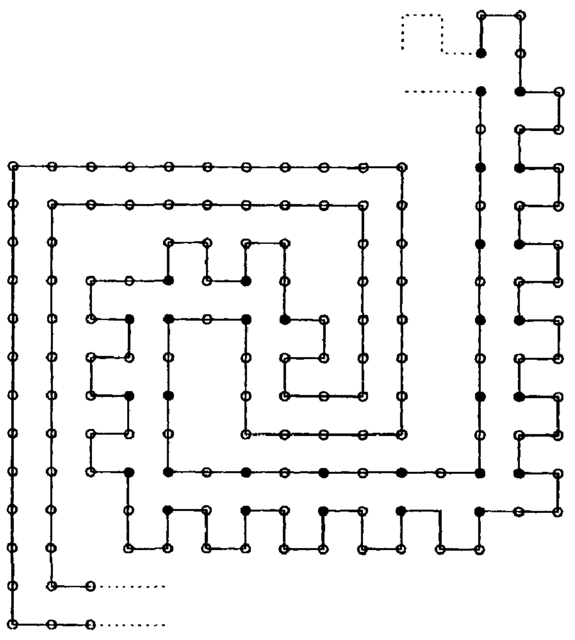

Figure 15: A "spiral" configuration of $\hat{S}$.

have an even-1 with two contacts. How many counter-clockwise turns can we make? After completing the first four counter-clockwise turns in the spiral, we have four even-1's with two contacts each. Then, one out of the next five even-1's has two contacts, then one out of next six, one out of the next seven, etc. Thus, the total number of even-1's with two contacts each is $\sqrt{2k}$. The total number of contacts is $k+\Theta(1)+\sqrt{2k} = (1+o(1))k$. $\square$

Theorem 3.1 follows from Lemmas 3.1 and 3.2.

Acknowledgments

I would like to thank Santosh Vempala for giving me this problem to work on, for suggesting that I work on the two aspects of the problem addressed in this paper, for many helpful discussions, for reading and commenting profusely on many drafts, and for being my advisor. I would like to thank Anna Lysyanskaya for her many helpful comments on the presentation of this paper. I would also like to acknowledge the support of an Edgerton Fellowship from the MIT EECS Department for the Spring 2001 semester.

References

[1] 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*, Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms (SODA '97) and Journal of Computational Biology, 4(2):275-296, 1997.

[2] Bonnie Berger and Tom Leighton, *Protein Folding in the Hydrophobic-Hydrophilic (HP) Model is NP-Complete*, Proceedings of the 2nd Conference on Computational Molecular Biology (RECOMB '98).

[3] P. Crescenzi, D. Goldman, C. Papadimitiou, A. Piccolboni, and M. Yannakakis, *On the Complexity of Protein Folding*, Proceedings of the 2nd Conference on Computational Molecular Biology (RECOMB '98).

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

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

[6] William E. Hart and Sorin Istrail, *Fast Protein Folding in the Hydrophobic-Hydrophilic Model Within Three-eighths of Optimal*, Proceedings of the 27th ACM Symposium on the Theory of Computing (STOC '95).

[7] Giancarlo Mauri, Antonio Piccolboni, and Giulio Pavesi, *Approximation Algorithms for Protein Folding Prediction*, Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA '99).