

Protein Threading by Linear Programming: Theoretical Analysis and Computational Results

Jinbo Xu¹, Ming Li¹, and Ying Xu²

¹ School of Computer Science, University of Waterloo,
Waterloo, Ontario N2L 3G1, Canada

{j3xu,mli}@uwaterloo.ca

² Department of Biochemistry, University of Georgia,
Athens, GA 30602-7229, USA

xyn@bmb.uga.edu

Abstract. In a previous paper [Xu *et al.*, 2003], we have used an integer programming approach to implement a protein threading program RAPTOR for protein 3D structure prediction, based on the threading model treating pairwise contacts rigorously and allowing variable gaps. We have solved the integer program by the canonical branch-and-bound method. In this paper we present a branch-and-cut method, a careful theoretical analysis of our formulation and why our approach is so effective. The result of cutting plane analysis is that two types of well-known cuts for this problem are already implied in the constraint set, which provides us some intuition that our formulation would be very effective. Experimental results show that for about 99 percent of real threading instances, the linear relaxations of their integer programs solve to integral optimal solutions directly. For the rest one percent of real instances, the integral solutions can be obtained with only several branch nodes. Experimental results also show that no special template or sequence features result in more possibilities of fractional solutions. This indicates that extra effort to seek for cutting planes to strengthen the existing formulation is unnecessary.

Keywords: Protein Threading, Integer Program, Branch-and-Cut, Cutting Plane, CAFASP

1 Introduction and Previous Work

A protein structure can be accurately determined by experimental techniques like *x-ray crystallography* or *nuclear magnetic resonance spectroscopy (NMR)*. However, such wet-lab approaches are costly and very time-consuming, ranging from months to years per structure. The overall strategy of the *Structural Genomics Initiatives*, launched by NIH in 1999, is to solve protein structures using experimental techniques only for a small fraction of all the proteins and to employ computational techniques to model the structures for the rest. The basic premise used here is that though there could be millions of proteins in nature, the number of unique structural folds is probably 2-3 (or even more) orders of magnitude smaller. Model-based structure prediction techniques could play a significant role in helping to achieve the goal of the Structural Genomics Initiatives. *Protein threading* is the most promising approach among such techniques with the increasing number of experimentally determined structures.

Protein threading makes a structure prediction by aligning the query sequence to each of available protein structures (templates) to see if the sequence can have a similar structure. The quality of sequence-template alignment is measured by a scoring function. The protein threading problem is NP-hard when the threading model treats pairwise contacts rigorously and allows variable gaps [Akutsu & Miyano, 1999]. Numerous prediction servers have been developed for protein structure prediction, based on the threading approach. Many [Jones, 1999, Shi *et al.*, 2001, Kelley *et al.*, 2000] of them do not take into account pairwise contacts in search for the optimal sequence-template alignment, avoiding the computational challenge. Others [Jones *et al.*, 1992, Godzik *et al.*, 1992, Bryant, 1996] use approximate algorithms to optimize their scoring functions when pairwise contacts are considered. Xu *et al.* [Xu *et al.*, 1998] proposed a divide-and-conquer method to search for the optimal solution at cost which is exponential to the topological complexity of template contact graphs. In our paper [Xu *et al.*, 2003], we have proposed an integer programming method to globally optimize the scoring function containing pairwise contact potential. We have implemented our method in the protein structure prediction server RAPTOR which was ranked first among all the programs of its category in the latest CAFASP3 (Third Critical Assessment of Fully Automated Structure Prediction) evaluation, a public and blind test of protein structure prediction servers [Fischer *et al.*, 2003].

In our paper [Xu *et al.*, 2003], we have formulated the integer program based on the following basic assumptions:

(1) Each template is parsed as a linear series of cores with the connecting loops between the adjacent cores. Each core is the most conserved segment of a α -helix or β -sheet secondary structure among the protein's homologs.

(2) When aligning a query protein sequence with a template, alignment gaps are confined to only loops. The biological justification is that cores are conserved so that the chance of insertion or deletion within them is very little.

(3) We consider only contacts between core residues. It is generally believed that interactions involving loop residues can be ignored as their contribution to fold recognition is relatively insignificant. We say that an interaction exists between two residues if the spatial distance between their C_β atoms is within 7\AA and they are at least 4 positions apart along the template sequence. *We say that an interaction exists between two cores if there exists at least one residue-residue interaction between the two cores.*

Let c_i ($i = 1, 2, \dots, M$) denote all cores of one template. The length of c_i is len_i . Let loc_i denote the sum of the length of all cores before c_i , i.e., $loc_i = \sum_{j=1}^{i-1} len_j$. For simplicity, when we say that core c_i is aligned to the sequence position j , we always mean that core c_i is aligned to a segment starting from position j to position $j + len_i - 1$.

We use an undirected graph $CMG = (V(CMG), E(CMG))$ to denote the (simplified) contact map graph of a protein template structure. Here, $V(CMG) = \{c_1, c_2, \dots, c_M\}$, and $E(CMG) = \{(c_i, c_j) \mid \text{there is an interaction between } c_i \text{ and } c_j, \text{ or } |i - j| = 1\}$. Figure 2 in the appendix gives an example of a contact graph and a sequence-template alignment.

Let $D[i]$ denote the set of all the sequence positions that core c_i could be aligned to and $R[i, j, l]$ the set of all the valid alignment positions of c_j given c_i is aligned to the l^{th} sequence position. Assume $i < j$, then for any $l_2 \in D[j]$, l_2 is in $R[i, j, l]$ if and only if $(l + loc_j - loc_i - l_2) < 0$. Obviously, for all $l_2 \in R[i, i+1, l]$, we have $l + len_i \leq l_2$. Please see Figure 3 in the appendix for an example of $D[i]$ and $R[i, j, l]$.

Definition 1. *Given a template with its (simplified) contact graph CMG and a sequence $s_1s_2\dots s_l\dots s_n$, an alignment between them is defined to be a set of ordered pairs (c_i, l_i) ($i = 1, 2, \dots, M$, $l_i \in \{1, 2, \dots, n\}$) such that $\forall i, l_i \in D[i]$ and $\forall i_1 < i_2, l_{i_2} \in R[i_1, i_2, l_{i_1}]$. An optimal alignment is one minimizing the scoring function $\sum_i E_{single}(c_i, l_i) + \sum_{(c_i, c_j) \in E(CMG)} E_{pairwise}(c_i, c_j, l_i, l_j)$.*

In the above definition, $E_{single}(c_i, l_i)$ refers to the singleton score when core c_i is aligned to position l_i . $E_{pairwise}$ refers to the pairwise score between two segments (start from l_i and l_j respectively) of the sequence when c_i and c_j are aligned to position l_i and l_j respectively. The gap score between two adjacent cores can be treated as a special kind of pairwise score.

Let $x_{i,l}$ be a binary variable indicating that core c_i is aligned to the l^{th} sequence position. For any $(c_{i_1}, c_{i_2}) \in E(CMG)$, let $y_{(i_1, l_1), (i_2, l_2)}$ indicate the pairwise interactions between two variables x_{i_1, l_1} and x_{i_2, l_2} ($l_2 \in R[i_1, i_2, l_1]$). $y_{(i_1, l_1), (i_2, l_2)}$ is equal to 1 if and only if both x_{i_1, l_1} and x_{i_2, l_2} are equal to 1. We say $y_{(i_1, l_1), (i_2, l_2)}$ is generated by x_{i_1, l_1} and x_{i_2, l_2} .

As formulated in our paper [Xu *et al.*, 2003], the objective function of the threading alignment problem is the linear combination of x and y variables. Please refer to the appendix for more detailed information about the objective function.

The constraint set is as follows:

$$\sum_{l \in D[i]} x_{i,l} = 1, i = 1, 2, \dots, M; \quad (1)$$

$$\sum_{k \in R[i, j, l]} y_{(i, l)(j, k)} = x_{i, l}, (c_i, c_j) \in E(CMG); \quad (2)$$

$$\sum_{l \in R[j, i, k]} y_{(i, l)(j, k)} = x_{j, k}, (c_i, c_j) \in E(CMG); \quad (3)$$

$$x_{i, l} \geq 0, l \in D[i], i = 1, 2, \dots, M; \quad (4)$$

$$y_{(i, l)(j, k)} \geq 0, \forall l \in D[i], k \in D[j], i, j = 1, 2, \dots, M. \quad (5)$$

Constraint 1 says that one core can be aligned to a unique sequence position. Constraints 2 and 3 imply that one y variable equals to 1 if and only if its two generating x variables are 1. That is, if two cores have interactions and are aligned to two sequence positions respectively, then the interaction score between the sequence residues aligned by these two cores must be calculated. Constraints 1, 4 and 5 guarantee x and y to be a real between 0 and 1 when this formulation is relaxed to its linear version.

Our previous paper [Xu *et al.*, 2003] and this paper are totally disjoint. The former presents the integer program formulation, RAPTOR system and some experimental results, and the latter presents a branch-and-cut method, cutting plane analysis, integrality proof of a special case, and some computational results regarding to the integrality of the real threading instances.

This paper is organized as follows. Section 1 summarizes our previous work and presents some basic assumptions and the integer program formulation for the sake of theoretical and experimental analysis in the following sections. Section 2 gives a branch-and-cut method and detailed cutting planes analysis. In this section, we prove that two kinds of well-known cuts are already implied in the constraint set. We also prove that the relaxed linear program solves to integral solution directly for a class of special contact graphs. These results give

us some intuition that our formulation would be very effective. In Section 3, we experimentally show that most of time the relaxed linear programs solve to the integral optimal solution without branching, which confirms our results in section 2. Finally, Section 4 draws some conclusions and discusses possible future work.

2 Theoretical Analysis

An integer program can be solved by either a branch-and-bound or branch-and-cut method. The branch-and-cut method solve an integer program by adding some cutting planes to strengthen the original formulation during the solution process. When we use the branch-and-cut method to solve an integer program, we need to recognize some effective cutting planes (also called valid inequalities or cuts) which could help to strengthen the integer program. The threading alignment problem can be treated as a special form of Max Independent Set (MIS) problem (see Section 2.2). For MIS problems, there are a priori two kinds of well-known valid inequalities that might be used to add the strength to the formulation. However, we find out that they are already implied by the constraints of our formulation due to the special structure of our problem, which gives us some intuition that our formulation is very strong and effective. We will also prove that for a special type of contact graphs, the optimal linear solutions of the relaxed linear programs are integral.

2.1 Branch-and-Cut Method

Instead of the branch-and-bound method used in our previous paper [Xu *et al.* , 2003], we resort to the branch-and-cut method to solve the integer program mentioned in Section 1. The major difference between the branch-and-bound and branch-and-cut method is that the branch-and-cut method recognizes and adds some cuts (valid inequalities) to strengthen the integer program during the branch-and-bound solution process. First of all, we relax the integer program by allowing all x and y to be a real number between 0 and 1 and solve the resulting linear program. If the solution (x^*, y^*) of the linear program is integral, then we get the optimal solution. Otherwise, we search for a valid inequality $ax + by \leq c$ that can separate (x^*, y^*) from the feasible integral solutions. That is, (x^*, y^*) violates $ax + by \leq c$ while the feasible integral solutions obey it. Any valid inequality found is added to the linear program as an extra constraint and the linear program is solved again. The generation of the valid inequalities is repeated until an integral solution is found or we cannot find any more valid inequalities. In the latter case, we choose one non-integral variable according to

a criterion, and generate two subproblems by setting it to 0 or 1 respectively. These two subproblems are solved recursively. An introduction to the branch-and-bound and branch-and-cut method for solving an integer program can be found in Wolsey's book [Wolsey, 1998].

2.2 Seeking for Cutting Planes

In this subsection, we try to seek for some effective cuts to strengthen our original formulation. However, we found out that one kind of well-known valid inequalities for this problem was already implied in the constraint set of our formulation and the other type of valid inequalities did not exist at all. Before presenting the detailed justification of our results, we first prepare some definitions and some simple lemmas. For the proof of these lemmas, please refer to our previous paper [Xu *et al.*, 2003].

Definition 2. For any two different x variables x_{i_1, j_1} and x_{i_2, j_2} , if $(loc_{i_1} - loc_{i_2}) \times (j_1 + loc_{i_2} - loc_{i_1} - j_2) \leq 0$, then we say that they are in conflict. Obviously, $j_2 \notin R[i_1, i_2, j_1]$ and $j_1 \notin R[i_2, i_1, j_2]$.

Lemma 1. For any two variables x_{i, l_0} and x_{j, k_0} ($i \neq j$), if x_{i, l_0} and x_{j, k_0} conflict, then for all $l \geq l_0, k \leq k_0$, $x_{i, l}$ and $x_{j, k}$ conflict.

Lemma 2. For any three different x variables x_{i_r, j_r} , $r = 1, 2, 3$ and $i_1 < i_2 < i_3$, if x_{i_1, j_1} conflicts with x_{i_2, j_2} and x_{i_2, j_2} conflicts with x_{i_3, j_3} , then x_{i_1, j_1} conflicts with x_{i_3, j_3} .

Lemma 3. For any three different x variables x_{i_r, j_r} , $r = 1, 2, 3$ and $i_1 < i_2 < i_3$, if x_{i_1, j_1} does not conflict with x_{i_2, j_2} and x_{i_2, j_2} does not conflict with x_{i_3, j_3} , then x_{i_1, j_1} does not conflict with x_{i_3, j_3} .

Lemma 4. For any three different x variables x_{i_r, j_r} , $r = 1, 2, 3$, $i_1 < i_2 = i_3$ and $j_2 < j_3$, if x_{i_1, j_1} does not conflict with x_{i_2, j_2} , then x_{i_1, j_1} does not conflict with x_{i_3, j_3} .

Lemma 5. For any three different edges x_{i_r, j_r} , $r = 1, 2, 3$ and $i_1 < i_2 < i_3$, if x_{i_1, j_1} conflicts with x_{i_3, j_3} , then x_{i_2, j_2} conflicts with either x_{i_1, j_1} or x_{i_3, j_3} .

Lemma 6. For any three different edges x_{i_r, j_r} , $r = 1, 2, 3$, $j_2 < j_3$ and $i_1 < i_2 = i_3$, if x_{i_1, j_1} conflicts with x_{i_3, j_3} , then x_{i_2, j_2} conflicts with x_{i_1, j_1} .

We have defined the ‘‘conflict’’ relationship between two x variables. Now we proceed to define the ‘‘conflict’’ relationship between x, y variables and between two y variables.

Definition 3. For any $x_{i,l}$ and $y_{(i_1,l_1)(i_2,l_2)}$, $(i,l) \neq (i_1,l_1)$, $(i,l) \neq (i_2,l_2)$, we say $x_{i,l}$ conflicts with $y_{(i_1,l_1)(i_2,l_2)}$ if $x_{i,l}$ conflicts with either x_{i_1,l_1} or x_{i_2,l_2} .

Definition 4. For any two different $y_{(i_1,l_1)(i_2,l_2)}$ and $y_{(i_3,l_3)(i_4,l_4)}$, we say they conflict if either x_{i_1,l_1} or x_{i_2,l_2} conflicts with $y_{(i_3,l_3)(i_4,l_4)}$.

Let X denote the set of all x variables and Y denote the set of all y variables. Let $X(c_i)$ denote the set of x variables derived from core c_i .

Definition 5. A x -variable graph $G_x = (V(G_x), E(G_x))$ is generated from X , where $V(G_x) = X$ and $E(G_x) = \{(x^1, x^2) | x^1, x^2 \in X \text{ and } x^1 \text{ conflicts with } x^2\}$.

Definition 6. A y -variable graph $G_y = (V(G_y), E(G_y))$ is generated from Y , where $V(G_y) = Y$ and $E(G_y) = \{(y^1, y^2) | y^1, y^2 \in Y \text{ and } y^1 \text{ conflicts with } y^2\}$.

Definition 7. A x clique inequality refers to that the sum of all x variables in one clique of graph G_x is no more than 1.

Definition 8. A y clique inequality refers to that the sum of all y variables in one clique of graph G_y is no more than 1.

For any feasible integral solution of our integer program, the set of x variables with value 1 form a maximal independent set of G_x . Similarly, the set of y variables with value 1 form a maximal independent set of G_y . So our threading alignment problem is transformed into the problem of finding a maximal independent set which minimizes the scoring function. For the maximal independent set problem, we have two kinds of well-known valid inequalities to strengthen the integer program [Carr *et al.*, 2000]. One is the clique inequality, that is, at most one vertex of a clique can be in the maximal independent set. The other is the odd hole (A hole is a chordless cycle of length at least four) inequality, that is, at most $\lfloor \frac{|H|}{2} \rfloor$ vertices of an odd cycle H ($|H| \geq 5$) can be in the maximal independent set. Generally, the number of these two kinds of inequalities is exponential so they cannot be incorporated into the integer program formulation explicitly. The branch-and-cut method makes use of them by dynamically incorporating some cuts that are violated by the current optimal linear solution if these violated inequalities can be recognized within a reasonable time. In the following paragraphs, we will prove that all the x -clique inequalities are already implied by the constraints of the integer program (Theorem 1). Also, there is no odd hole in G_x (Theorem 2). Several types of simple y -clique inequalities are also implied. Therefore, it is unnecessary for us to add the x -clique inequalities during the solution process of the branch-and-cut method. Before proving Theorem 1 and Theorem 2, we need to prove some lemmas.

Lemma 7. *Given three cores c_{i_1} , c_{i_2} , and c_{i_3} with $i_1 < i_2 < i_3$ and two variable sets $X^1 = \{x_{i_1,l}, l \geq l_0\}$ and $X^3 = \{x_{i_3,k}, k \leq k_0\}$, suppose that any two variables $x_{i_1,l} \in X^1$ and $x_{i_3,k} \in X^3$ conflict. Then each variable in $X(c_{i_2})$ conflicts with either all variables in X^1 or all in X^3 .*

Proof. For each $x_{i_2,r}$ in $X(c_{i_2})$, because x_{i_1,l_0} conflicts with x_{i_3,k_0} , based on Lemma 5, we see that $x_{i_2,r}$ conflicts with x_{i_1,l_0} or x_{i_3,k_0} . Without loss of generality, assume $x_{i_2,r}$ conflicts with x_{i_1,l_0} . Then $(r - l_0 + loc_{i_1} - loc_{i_2}) \leq 0$ (because $loc_{i_2} > loc_{i_1}$). For each $x_{i_1,l}$, $l \geq l_0$, we have $(r - l + loc_{i_1} - loc_{i_2}) \leq (r - l_0 + loc_{i_1} - loc_{i_2}) \leq 0$. Therefore, $x_{i_2,r}$ conflicts with $x_{i_1,l}$ for all $l \geq l_0$.

Lemma 7 states that for any two nonadjacent cores c_i , c_j and two different x variable sets $X^1 \subseteq X(c_i)$, $X^2 \subseteq X(c_j)$, if any two variables in $X^1 \cap X^2$ conflict with each other, then any x variable from any core in segment c_{i+1}, \dots, c_{j-1} conflicts with either all variables in X^1 or all variables in X^2 . Lemma 7 can be regarded as a generalization of Lemma 5.

In order to prove Theorem 1, that is, any x clique inequality is already implied by the constraint set of our formulation, we first prove a simple case: any x clique inequality involving with x variables of only two cores is implied by the constraint set. We then generalize the simple case. The simple case is shown in Lemma 8.

Lemma 8. *Each x -clique inequality involved with only two cores is implied by constraints 1–5.*

Proof. Suppose that the maximal clique of G_x consists of all x -variables from two sets $X(c_i)$ and $X(c_j)$ ($i < j$). We use induction on the value of $j - i$. Let $X_i = \{x_{i,l}, l \geq l_0\} \subseteq X(c_i)$ and $X_j = \{x_{j,k}, k \leq k_0\} \subseteq X(c_j)$ be the variables contained in the clique. We need to show $\sum_{x \in X_i} x + \sum_{x \in X_j} x \leq 1$. Obviously we have k_0 is less than the minimum element in $R[i, j, l_0]$.

First we prove the case $j - i = 1$. From Constraints 2 and 3, we have

$$\begin{aligned}
\sum_{l \geq l_0} x_{i,l} + \sum_{k \leq k_0} x_{i+1,k} &= \sum_{l \geq l_0} \sum_{k_1 \in R[i, i+1, l]} y_{(i,l)(i+1,k_1)} + \sum_{k \leq k_0} x_{i+1,k} \\
&\leq \sum_{k_1 > k_0} \sum_{l \geq l_0, l \in R[i+1, i, k_1]} y_{(i,l)(i+1,k_1)} + \sum_{k \leq k_0} x_{i+1,k} \\
&\leq \sum_{k_1 > k_0} \sum_{l \in R[i+1, i, k_1]} y_{(i,l)(i+1,k_1)} + \sum_{k \leq k_0} x_{i+1,k} \\
&= \sum_{k_1 > k_0} x_{i+1,k_1} + \sum_{k \leq k_0} x_{i+1,k}
\end{aligned}$$

$$\begin{aligned}
 &= \sum_{k \in D[i+1]} x_{i+1,k} \\
 &= 1
 \end{aligned}$$

Now assume that when $j - i \leq m$, the proposition holds. We will prove it for the case $j - i = m + 1$. For any r between i and j , obviously, $r - i \leq m$ and $j - r \leq m$. Based on Lemma 7, for all $x_{r,l}$, $x_{r,l}$ must conflict with either all $x_{i,l}$, ($l \geq l_0$) or all $x_{j,k}$, ($k \leq k_0$). Let $X_r^1 = \{x_{r,l_r} | x_{r,l_r} \text{ conflicts with all edges in } X_i\}$ and $X_r^2 = \{x_{r,l_r} | x_{r,l_r} \text{ conflicts with all edges in } X_j\}$. Then $X(c_r) \subseteq (X_r^1 \cup X_r^2)$ and $\sum_{x \in X_r^1} x + \sum_{x \in X_r^2} x \geq \sum_{x \in X(c_r)} x \geq 1$. X_i, X_r^1 form a clique and X_j, X_r^2 form a clique. This yields,

$$\begin{aligned}
 \sum_{l \geq l_0} x_{i,l} + \sum_{k \leq k_0} x_{j,k} &= \left(\sum_{l \geq l_0} x_{i,l} + \sum_{x \in X_r^1} x \right) + \left(\sum_{k \leq k_0} x_{j,k} + \sum_{x \in X_r^2} x \right) - \left(\sum_{x \in X_r^1} x + \sum_{x \in X_r^2} x \right) \\
 &\leq 1 + 1 - 1 \\
 &= 1
 \end{aligned}$$

This completes the proof of the lemma by induction.

Theorem 1. *Each x -clique inequality is implied by constraints 1–5.*

Proof. For a given maximal clique of G_x , we use induction on the number of cores.

In the base case, the maximal clique consists of variables from just two cores. Then the proposition holds according to Lemma 8.

Now assume the proposition holds if the maximal clique is formed by x variables from no more than i cores. Consider a maximal clique formed by the x variables of core $c_{q_1}, c_{q_2}, \dots, c_{q_i}, c_{q_{i+1}}$. Let $X_{clique}(c_r)$ denote the x variables of cores c_r for all r in $\{q_1, q_2, \dots, q_{i+1}\}$ contained in this maximal clique. Based on Lemma 7 and Lemma 2, for any $x \in X(c_{q_i})$, at least one of the following two statements holds:

1. x conflicts with all variables in sets $X_{clique}(c_r)$ for all r in $\{q_1, q_2, \dots, q_{i-1}\}$. Let $X^1(c_{q_i})$ denote the set of this kind of x , then $X_{clique}(c_{q_1}), X_{clique}(c_{q_2}), \dots, X_{clique}(c_{q_{i-1}})$ and $X^1(c_{q_i})$ form a clique;
2. x conflicts with all variables in set $X_{clique}(c_{q_{i+1}})$. Let $X^2(c_{q_i})$ denote the set of this kind of x , then $X^2(c_{q_i})$ and $X_{clique}(c_{q_{i+1}})$ form a clique.

Obviously $X^1(c_{q_i}) \cup X^2(c_{q_i}) = X(c_{q_i})$ and $X^1(c_{q_i}) \cap X^2(c_{q_i}) = X_{clique}(c_{q_i})$ (otherwise, this clique will not be maximal). Let $S(Z)$ denote the sum of x variables in the set Z . We have

$$\sum_{r=1}^{i+1} S(X_{clique}(c_{q_r})) = \sum_{r=1}^{i-1} S(X_{clique}(c_{q_r})) + S(X_{clique}(c_{q_i})) + S(X_{clique}(c_{q_{i+1}}))$$

$$\begin{aligned}
&= \sum_{r=1}^{i-1} S(X_{\text{clique}}(c_{q_r})) + S(X^1(c_{q_i})) + S(X^2(c_{q_i})) \\
&\quad + S(X_{\text{clique}}(c_{q_{i+1}})) - S(X(c_{q_i})) \\
&\leq 1 + 1 - 1 \\
&= 1
\end{aligned}$$

This concludes the proof by induction of the lemma.

So far, we have proved that any x clique inequality is implied by the constraint set of our formulation. The following theorem shows that there is no odd hole in the G_x graph.

Theorem 2. *There is no odd hole in graph G_x .*

Proof. Assume there is a minimal odd hole H with $|H| \geq 5$. Let $H = x^1, x^2, \dots, x^{|H|}, x^1$. Let c_{r_j} (for j in $\{1, 2, \dots, |H|\}$) be the cores of the x^j and k_j be the sequence positions of x^j . Obviously, there are no three common cores among the c_{r_j} ; otherwise, H would have at least one chord. Because $|H|$ is odd, there must be one j such that $r_j \leq r_{j+1} \leq r_{j+2}$. In this proof, if $j + d$ is greater than $|H|$, then it should be interpreted as $j + d - |H|$ ($d = 1, 2, \dots$); if $l - d$ is less than 0, then it should be interpreted as $l - d + |H|$ ($d = 1, 2, \dots$). We have the following two cases:

1. r_j, r_{j+1} , and r_{j+2} are all different. Based on Lemma 2, x^j conflicts with x^{j+2} . Thus (x^j, x^{j+2}) is an edge in G_x that forms a chord in H . So H is not a hole, which is a contradiction.
2. Two of r_j, r_{j+1} , and r_{j+2} are equal. Without loss of generality, assume r_j is less than r_{j+1} which is equal to r_{j+2} . Then $k_{j+1} < k_{j+2}$, otherwise, x^j would conflict with x^{j+2} based on Lemma 6. We now prove that for all l not equal to $j + 1, j + 2$, we have that r_l is less than r_{j+1} . For all l such that l and $l - 1$ are not equal to $j + 1$ or $j + 2$, either $l - 1 \not\equiv j + 3 \pmod{|H|}$ or $l \neq j$ holds; otherwise, $|H| \leq 4$. We have the following two cases:
 - If $l - 1 \not\equiv j + 3 \pmod{|H|}$, then (x^l, x^{j+2}) and (x^{l-1}, x^{j+2}) cannot be the edges of G_x ; otherwise H has chords. Since (x^l, x^{l-1}) is an edge of H and G_x (i.e., x^l conflicts with x^{l-1}), based on Lemma 5, we have that either both r_l and r_{l-1} are greater than r_{j+2} or both r_l and r_{l-1} are less than r_{j+2} .
 - If $l \neq j$, then (x^l, x^{j+1}) and (x^{l-1}, x^{j+1}) cannot be the edges of G_x because H has no chord. Similarly, we have that either both r_l and r_{l-1} are greater than r_{j+1} or both r_l and r_{l-1} are less than r_{j+1} .

Thus, for all l in $\{j, j-1, \dots, j+4\}$, we have either both r_l and r_{l-1} are less than r_{j+1} and r_{j+2} or both r_l and r_{l-1} are greater than r_{j+1} and r_{j+2} . Because we have already assumed r_j less than r_{j+1} , for all l in $\{j, j-1, \dots, j+4\}$, r_l is less than r_{j+1} and r_{j+2} . Since (x^{j+3}, x^{j+1}) is not an edge of G_x (otherwise H has chords), $k_{j+1} < k_{j+2}$ and $r_{j+3} < r_{j+1} = r_{j+2}$, based on Lemma 4, we have (x^{j+3}, x^{j+2}) is not an edge of G_x or H , which is a contradiction!

This completes the proof of this theorem.

Based on Lemma 8, we have the following lemma about a class of very simple y cliques.

Lemma 9. *Any y -clique inequality consisting of only two y variables is implied by constraints 1–5.*

Proof. Consider their corresponding x variables. There are at least two x variables from these two y variables respectively being in conflict by Definition 3 and 4. The sum of these two y variables are no more than that of these two x variables according to Constraint 2 and 3. Based on Lemma 8, the proposition holds.

Based on the above results (Theorem 1, Theorem 2 and Lemma 9), we can see that our formulation captures the special structure of this problem so well that we do not have to explicitly add any x -clique inequality during the solving of our integer program. Experimental results show that most of time, the optimal solution of our integer program's linear relaxation is integral. We do not need to consider more complex y -clique inequalities nor any further cuts.

2.3 Integrality of Linear Solutions of A Special Case

If the template contact graph does not contain the interaction preference edges, that is, it only contains the variable gap edges between two adjacent cores, then a simple dynamic programming algorithm can solve the threading alignment problem within time bounded by a low degree polynomial $O(Mn^2)$ where M is the number of template cores and n is the sequence length. In this subsection, we will show that in this case, the linear solution of the relaxed linear program is also integral.

Theorem 3. *If the interaction preferences are not considered, that is, the contact graph CMG only contains edges corresponding to the variable gaps, then the optimal solution of the relaxed linear program is integral.*

Proof. Let $S^* = (x^*, y^*)$ denote the optimal linear solution. We will show that $S^* = (x^*, y^*)$ is a convex combination of feasible integral solutions if it is not integral. That is, there exists a set of K integral solutions $S_j = (x^j, y^j)$ and positive fractions α_j for j from 1 to K with $\sum_{j=1}^K \alpha_j = 1$, such that

$$S^* = \sum_{j=1}^K \alpha_j S^j \quad (6)$$

Let $(x^0, y^0) = (x^*, y^*)$. We have the following properties based on Constraint 2 and 3.

$$x_{i,l_i}^0 = \sum_{l \in R[i,i+1,l_i]} y_{(i,l_i)(i+1,l)}^0, \text{ for all } l_i \in D[i] \quad (7)$$

$$\sum_{l \in R[i+1,i,l_{i+1}]} y_{(i,l)(i+1,l_{i+1})}^0 = x_{i+1,l_{i+1}}^0, \text{ for all } l_{i+1} \in D[i+1] \quad (8)$$

$$\sum_{l \in D[i]} x_{i,l}^0 = \sum_{k \in D[j]} x_{j,k}^0, \text{ for all } i, j \quad (9)$$

We decompose (x^*, y^*) by repeating the following two steps until $(x^0, y^0) = 0$.

Step 1: Starting from core c_1 , alternatively choose positive elements x_{1,l_1}^0 , $y_{(1,l_1)(2,l_2)}^0$, x_{2,l_2}^0 , $y_{(2,l_2)(3,l_3)}^0$, ..., $y_{(M-1,l_{M-1})(M,l_M)}^0$, x_{M,l_M}^0 from vector (x^0, y^0) such that no conflict occurs among them. We can always do so. Given $x_{i,l_i}^0 > 0$, $\sum_{l \in R[i,i+1,l_i]} y_{(i,l_i)(i+1,l)}^0 = x_{i,l_i}^0 > 0$, so we can choose one $y_{(i,l_i)(i+1,l_{i+1})}^0 > 0$ with $l_{i+1} \in R[i, i+1, l_i]$. Given one $y_{(i,l_i)(i+1,l_{i+1})}^0 > 0$, we can choose $x_{i+1,l_{i+1}}^0$ because $x_{i+1,l_{i+1}}^0 \geq y_{(i,l_i)(i+1,l_{i+1})}^0 > 0$. Obviously, we can always choose $x_{1,l_1}^0 > 0$ because $\sum_{l \in D[1]} x_{1,l}^0 > 0$. Let α_j denote the smallest element in the sequence x_{1,l_1}^0 , $y_{(1,l_1)(2,l_2)}^0$, x_{2,l_2}^0 , ..., x_{M,l_M}^0 .

Step 2: Let (x^j, y^j) denote a vector with $x_{1,l_1}^j = y_{(1,l_1)(2,l_2)}^j = x_{2,l_2}^j = \dots = x_{M,l_M}^j = 1$ and other elements being 0. Based on the above choice, (x^j, y^j) is a feasible integral solution to the linear program. Let $(x^0, y^0) = (x^0, y^0) - \alpha_j(x^j, y^j)$. Obviously, $(x^0, y^0) \geq 0$. It is easy to verify that Equation 7, 8 and 9 still hold.

Assume that after K iterations, (x^0, y^0) becomes 0. Obviously, $\sum_{j=1}^K \alpha_j = \sum_{l \in D[1]} x_{1,l}^* = 1$ because at each iteration j , for any core c_i , $\sum_{l \in D[i]} x_{i,l}^0$ is decreased by α_j . Therefore, Equation 6 holds. However, for any linear program, an optimal solution cannot be a convex combination of other feasible solutions [Chvátal, 1983, Schrijver, 1986]. Therefore, the optimal solution (x^*, y^*) is integral.

The results of Theorem 1, Theorem 2 and Theorem 3 reveal that our formulation is very effective. In the next section, we will describe the behavior of our formulation on the actual protein data.

3 Practical Computational Efficiency

In this section, we will investigate some experimental results on the integrality of the linear solutions of our integer program and the CPU time with respect to the sequence size.

3.1 Integrality of Linear Solutions

Table 1. Statistic of integrality of the linear solutions.

Test Set	Integrality Ratio	Maximum Branch Number	Sequence Fraction Average (Deviation)	Sequence Fraction Maximum	Template Fraction Average (Deviation)	Template Fraction Maximum
Fischer	98.7	7	3.03 (3.04)	18	0.79 (1.04)	5
Holm	99.1	7	2.02 (1.94)	7	1.67 (1.91)	10
Lindhal	99.0	9	6.31 (5.72)	31	1.70 (1.88)	10

(1) $SFracNum(S)$: the number of threading pairs with the query sequence being S whose linear programs solve to fractional optimal solutions; (2) $TFracNum(T)$: the number of threading pairs with the template being T whose linear programs solve to fractional optimal solutions; (3) Integrality Ratio: percent of the linear solutions being integral; (4) Max Branch Number: the maximum number of branch nodes among all threading pairs; (5) Sequence Fraction Average: average of $SFracNum(S)$ among all sequences in the test set; (6) Sequence Fraction Deviation: standard deviation of $SFracNum(S)$ among all sequences in the test set; (7) Sequence Fraction Maximum: the maximum of $SFracNum(S)$ among all sequences; (8) Template Fraction Average: average of $TFracNum(T)$ among all templates in the test set; (9) Template Fraction Deviation: standard deviation of $TFracNum(T)$ among all templates in the test set; (10) Template Fraction Maximum: the maximum of $TFracNum(T)$ among all templates.

To evaluate the behaviors of the linear solutions of the relaxed linear programs, the sequences and templates from Fischer *et al.*'s test set [Fischer *et al.* , 1996], Lindahl and Elofsson's test set [Lindahl & Elofsson, 2000] and Holm and Sander's test set [Holm & Sander, 1997] are used. For Fischer *et al.*'s test set, all sequences are threaded against all templates in the test set. For Lindahl and Elofsson's and Holm and Sander's test sets, some sequences are randomly selected and threaded against all the templates in the corresponding test set. Those templates with no more than 4 cores or without interactions between

cores are excluded from this experiment because their corresponding linear programs always produce integral solutions. In our experiment, there are 68×262 threading pairs from Fischer *et al.*'s test set, 192×712 pairs from Lindahl and Elofsson's test set, and 197×238 pairs from Holm and Sander's test set. For each threading pair, we record whether the optimal linear solution of its linear program is integral and the number of branch nodes if the solution is fractional.

Let $SFracNum(S)$ denote the number of threading pairs with the query sequence being S whose linear programs solve to the fractional optimal solutions, and $TFracNum(T)$ denote the number of threading pairs with the template being T whose linear programs solve to the fractional optimal solutions. We calculate some indices as shown in Table 1. Approximately 99% of linear programs solve to integral optimal solutions directly without branching at all. This result, together with the theoretical results proved in Section 2, demonstrates that our integer program formulation is well defined. Since linear programs can be solved within polynomial time, in a practical sense, threading alignment problems can be solved within polynomial time with our scoring function and alignment model. Based on the mean and standard deviation statistics in Table 1, it seems unlikely that there are some specific sequences or templates which can lead to a much greater chance of fractional linear solutions.

Table 2. Correlation coefficient between the number of fractional linear solutions and sequence length, and four template structure features: topological complexity, #cores (number of cores), template size and #edges (number of edges in the structure contact graph).

	topological complexity	#cores	template size	#edges	sequence length
Holm	0.0389	0.0879	-0.0742	0.1155	0.1467
Lindhal	0.0774	0.0587	-0.1510	0.0816	-0.0506
Fischer	-0.1107	0.2207	0.1230	0.1867	0.1991

Table 2 shows the correlation coefficients between $TFracNum(T)$ and the structure features of the templates, and the correlation coefficients between $SFracNum(S)$ and the sequence size. Please refer to Xu *et al.*'s paper [Xu *et al.* , 1998] for the detailed description of topological complexity. As shown in these two tables, there is a very weak relationship between the non-integrality of the linear solutions and the topological features of the templates and sequences. This means that for any linear program derived from any threading pair with any topology, there appears to be a good chance of solving to an integral optimal solution directly. Our integer programming approach is effective and efficient for most of the threading pairs.

3.2 CPU time

Besides the previous features of our algorithm, another advantage is that the computing time increases roughly with the sequence size. Figure 1 reflects the CPU time of aligning a template 119l with length 162 to 170 sequences (chosen randomly from Lindahl and Elofsson's benchmark) with sizes ranging from 28 to 527. The figure shows that the computing time of our algorithm increases very slowly (almost linearly rather than exponentially) with respect to the sequence size.

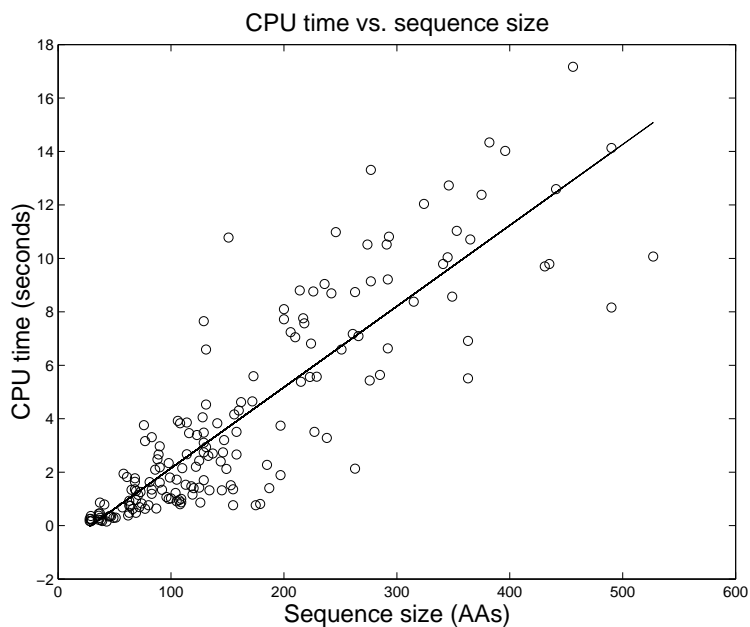


Fig. 1. CPU time of threading 170 sequences to template 119l.

4 Conclusion

This paper mainly introduces a branch-and-cut method to solve the integer program formulation employed by our protein structure prediction server RAPTOR. A surprising result during seeking for cutting planes is that two kinds of well-known cuts which might be useful for strengthening our formulation during the branch-and-cut process are already implied in our constraint set. Furthermore, The experimental results show that our integer program formulation is

so “tight” that approximately 99% of the relaxed linear programs solve to the optimal integral solutions without branching at all. It turns out that in practice, the threading alignment problem can be solved within polynomial time though theoretically it is NP-hard. However, our theoretical analysis can not provide a complete justification of the behavior of our integer program. A possible future work is to refine the model of the scoring function to gain more insight into the threading problem.

References

- [Akutsu & Miyano, 1999] Akutsu, T., & Miyano, S. 1999. On the approximation of protein threading. *Theoretical Computer Science*, **210**, 261–275.
- [Bryant, 1996] Bryant, S. 1996. Evaluation of threading specificity and accuracy. *Proteins: Structure, Function and Genetics*, **26**, 172–185.
- [Carr *et al.*, 2000] Carr, R., Lancia, G., & Istrail, S. 2000. *On the Integer Programming Approach to Independent Set Problems with Applications to Protein Structure Alignment*. Tech. rept. SAND2000-2171. Sandia National Labs, Albuquerque.
- [Chvátal, 1983] Chvátal, V. 1983. *Linear Programming*. NY, USA: W.H. Freeman.
- [Fischer *et al.*, 1996] Fischer, D., Elofsson, A., Bowie, J.U., & Eisenberg, D. 1996. Assessing the performance of fold recognition methods by means of a comprehensive benchmark. *Pages 300–318 of: Hunter, L., & Klein, T. (eds), Biocomputing: Proceedings of the 1996 Pacific Symposium*. Singapore: World Scientific Publishing Co.
- [Fischer *et al.*, 2003] Fischer, D., Rychlewski, L., Dunbrack, R.L., Ortiz, A.R., & Elofsson, A. 2003. CAFASP3: The Third Critical Assessment of Fully Automated Structure Prediction Methods. *Proteins: Structure, Function and Genetics*, **S6(53)**, 503–516.
- [Godzik *et al.*, 1992] Godzik, A., Kolinski, A., & Skolnick, J. 1992. A topology fingerprint approach to inverse protein folding problem. *Journal of Molecular Biology*, **227**, 227–238.
- [Holm & Sander, 1997] Holm, L., & Sander, C. 1997 (June). Decision support system for the evolutionary classification of protein structures. *Pages 140–146 of: Proceedings of the Fifth International Conference on Intelligent Systems for Molecular Biology*, vol. 5.
- [Jones, 1999] Jones, D.T. 1999. GenTHREADER: An Efficient and Reliable Protein Fold Recognition Method for Genomic Sequences. *Journal of Molecular Biology*, **287**, 797–815.
- [Jones *et al.*, 1992] Jones, D.T., Taylor, W.R., & Thornton, J.M. 1992. A new approach to protein fold recognition. *Nature*, **358**, 86–98.
- [Kelley *et al.*, 2000] Kelley, L.A., MacCallum, R.M., & Sternberg, M.J.E. 2000. Enhanced Genome Annotation using Structural Profiles in the Program 3D-PSSM. *Journal of Molecular Biology*, **299(2)**, 499–520.
- [Lindahl & Elofsson, 2000] Lindahl, E., & Elofsson, A. 2000. Identification of related proteins on family, superfamily and fold level. *Journal of Molecular Biology*, **295**, 613–625.
- [Schrijver, 1986] Schrijver, A. 1986. *Theory of Linear and Integer Programming*. John Wiley and Sons.
- [Shi *et al.*, 2001] Shi, J., Tom, L. B., & Kenji, M. 2001. FUGUE: Sequence-structure homology Recognition Using Environment-specific substitution Tables and Structure-dependent Gap penalties. *Journal of Molecular Biology*, **310**, 243–257.
- [Wolsey, 1998] Wolsey, L.A. 1998. *Integer Programming*. John Wiley and Sons.
- [Xu *et al.*, 2003] Xu, J., Li, M., Kim, D., & Xu, Y. 2003. RAPTOR: Optimal Protein Threading by Linear Programming. *Journal of Bioinformatics and Computational Biology*, **1(1)**, 95–117.

[Xu *et al.*, 1998] Xu, Y., Xu, D., & Uberbacher, E.C. 1998. An Efficient Computational Method for Globally Optimal Threadings. *Journal of Computational Biology*, **5**(3), 597–614.

5 Appendix

5.1 Objective Function Formulation

In our previous paper [Xu *et al.*, 2003], the objective function of the threading alignment problem is defined as:

$$\begin{aligned} & \min W_m E_m + W_s E_s + W_p E_p + W_g E_g + W_{ss} E_{ss}, \\ E_m &= \sum_{i=1}^M \sum_{l \in D[i]} [x_{i,l} \times \sum_{r=0}^{len_i-1} Mutation(head_i + r, l + r)], \\ E_s &= \sum_{i=1}^M \sum_{l \in D[i]} [x_{i,l} \times \sum_{r=0}^{len_i-1} Fitness(head_i + r, j + r)], \\ E_{ss} &= \sum_{i=1}^M \sum_{l \in D[i]} [x_{i,l} \times \sum_{r=0}^{len_i-1} SS(head_i + r, j + r)], \\ E_p &= \sum_{1 \leq i < j \leq M, (c_i, c_j) \in E(CMG)} \sum_{l \in D[i]} \sum_{k \in R[i, j, l]} y_{(i, l), (j, k)} P(i, j, l, k), \\ P(i, j, l, k) &= \sum_{u=0}^{len_i-1} \sum_{v=0}^{len_j-1} \delta(t_{head_i+u}, t_{head_j+v}) Pair(l + u, k + v) \\ E_g &= \sum_{i=1}^M \sum_{l \in D[i]} \sum_{k \in R[i, i+1, l]} y_{(i, l), (i+1, k)} G(i, l, k), \end{aligned}$$

Where mutation score E_m , environment fitness score E_s , and secondary structure compatibility score E_{ss} are linear combinations of x variables; pairwise interaction score E_p and gap score E_g are linear combinations of y variables; and $W_m, W_s, W_p, W_g, W_{ss}$ are weight factors determined by a combined Genetic Algorithm and Local Pattern Search method. $Mutation(i, l)$ denotes the mutation score between the i^{th} template residue and the l^{th} sequence residue. $Fitness(i, l)$ denotes the environment fitness score of the l^{th} sequence residue being put into environment of the i^{th} template residue. $SS(i, l)$ denotes the consistency between the secondary structure of the i^{th} template residue and the predicted secondary structure of the l^{th} sequence residue. $Pair(l_1, l_2)$ denotes the pairwise interaction score between the l_1^{th} sequence residue and the l_2^{th}

sequence residue. $G(i, l, k)$ denote the gap score between i^{th} core and $(i + 1)^{th}$ core when they are aligned to sequence position l and k respectively. It is calculated by a simple dynamic programming method. Please refer to our paper [Xu *et al.* , 2003] for a detailed description.

5.2 An Sequence-Template Alignment Example

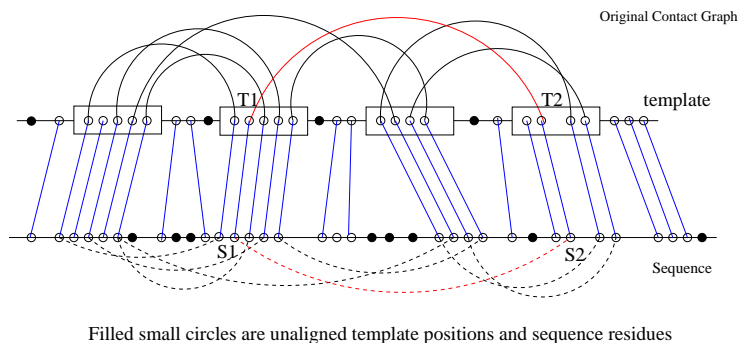


Fig. 2. An example of sequence-template alignment. A small circle represents one residue. A solid arc in the original contact graph represents an interresidue contact. A dashed arc indicates that if two sequence residues are aligned to two interacted template residues, then the interaction score of these two sequence residues must be considered in the scoring function. Filled small circles in the figure are some gaps in the sequence-template alignment.

5.3 An Example Figure of $D[i]$ and $R[i, j, l]$

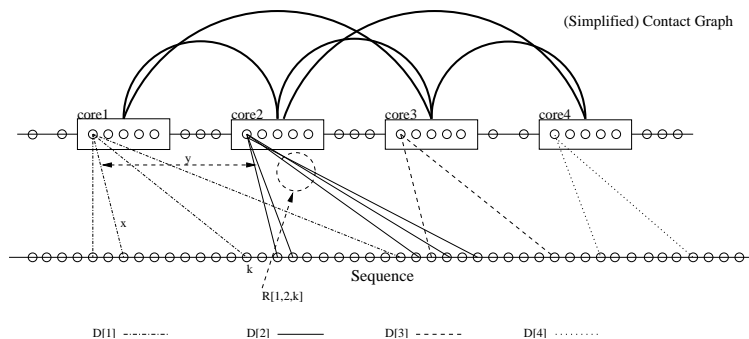


Fig. 3. Example of $D[i]$ and $R[i, j, l]$.