101 Optimal PDB Structure Alignments: a Branch–and–Cut Algorithm For The Maximum Contact Map Overlap Problem

Giuseppe Lancia[*][†] Robert Carr[‡] Brian Walenz[§] Sorin Istrail[¶]

Dedicated to the memory of Dr. Fred Howes

ABSTRACT

Structure comparison is a fundamental problem for structural genomics. A variety of structure comparison methods were proposed and several protein structure classification servers e.g., SCOP, DALI, CATH, were designed based on them, and are extensively used in practice. This area of research continues to be very active, being energized bi-annually by the CASP folding competitions, but despite the extraordinary international research effort devoted to it, progress is slow. A fundamental dimension of this bottleneck is the absence of rigorous algorithmic methods. A recent excellent survey on structure comparison by Taylor et.al. [23] records the state of the art of the area: *In structure comparison, we do not even have an algorithm that guarantees an optimal answer for pairs of structures ...*

In this paper we provide the first rigorous algorithm for structure comparison. Our method is based on developing an effective integer linear programming (IP) formulation of protein structure *contact maps overlap* (CMO), and a branch-and-cut strategy that employs lower-bounding heuristics at the branch nodes. Our algorithms identified a gallery of optimal and near-optimal structure alignments for pairs of proteins from the Protein Data Bank with up to 80 amino acids and about 150 contacts each – problems of instance size of about 300. Although these sizes also reflect our current limitations, these are the first provable optimal and near-optimal algorithms in the literature for a measure of structure similarity which sees extensive practical use. At the heart of our success in finding optimal alignments is a reduction of the CMO optimization to the maximum independent set (MIS) problem on special graphs. For CMO instances of size 300, the corresponding MIS graph instance contains about 10,000 nodes. While our algorithms are able to solve to optimality MIS problem of these sizes, the known optimal algorithms for the MIS on general graphs can at present only solve instances with up to a few hundred nodes. This is the first effective use of IP methods in protein structure comparison; the biomolecular structure literature contains only one other effective IP method devoted to RNA comparison, due to Lenhof et.al. [18].

The hybrid heuristic approach that worked well for providing lower bounds in the branch and cut algorithm was tried on large proteins in a test set suggested by Jeffrey Skolnick. It involved 33 proteins classified into four families: Flavodoxin-like fold CheY-related, Plastocyanin, TIM Barrel, and Ferratin. Out of the set of all 528 pairwise structure alignments, we have validated the clustering with a 98.7% accuracy (1.3% false negatives and 0% false positives).

[*]Celera Genomics, Rockville, MD, email: Giuseppe.Lancia@celera.com

[†]D.E.I., University of Padova

[‡]Sandia National Labs, Albuquerque, NM, email: bobcarr@cs.sandia.gov

[§]Celera Genomics, Rockville, MD, email: Brian.Walenz@celera.com

[¶]Celera Genomics, Rockville, MD, email: Sorin.Istrail@celera.com

1. INTRODUCTION

A fundamental dimension of the difficulties encountered in designing successful practical predictions methods for protein structure is the absence of rigorous algorithmic methods for basic problems in structure analysis. Here are several reasons for this situation.

(1) By their nature, three-dimensional computational problems are inherently more complex than the similar one-dimensional ones for which we have more effective solutions. The mathematics that can provide rigorous support in understanding models for structure prediction and analysis is almost nonexistent, as the problems are a blend of continuos-geometrical- and combinatorial-discrete-mathematics.

(2) Various simplified versions of the problems were shown NP-complete, e.g., for structure comparison see [11, 10].

(3) There is a dramatic difference between sequence

alignment and structure alignment. As opposed to the protein sequence alignment, where we are certain that there is a unique alignment to a common ancestor sequence, in structure comparison the notion of common ancestor does not exist. Similarity in folding structure is due to a different balance in folding forces, and there is not necessarily a one-to-one correspondence between positions in both proteins. In fact, for two homologous proteins that are distantly related, it is possible for the structural alignment to be entirely different from the correct evolutionary alignment. [6].

Pairwise structure comparison requires a structure similarity scoring scheme that captures biological relevance of the chemical and physical steric constraints involved in molecular recognition. The most used scoring schemes are based on three themes: *RMSD of rigid-body superposition* [17], *distance map similarity* [15] and *contact map overlap* (CMO) [8]. All these measures of similarity use distances between residues and raise computational issues that at present do not have effective computational solutions. The first two measures require a preset alignment for the equivalenced residues in the two proteins to be given.

In contrast, the CMO measure does not require a preset alignment. The measure was referred to as the "ultimate structure comparison measure" [19].

CMO is based on the basic notion of *contact* between two residues, notion of fundamental statistical mechanics and chemical significance, and at the core of many scoring schemes used in applications of protein structure analysis and simulation. They applications include: validation of protein models, identification of native folding motifs among many incorrect alternatives, identification of possible folds for a sequence of unknown structure, finding sequences compatible with given structures.

This paper focuses on the CMO scoring scheme, which was extensively studied and empirically validated by the Godzik-Skolnick group at the Scripps Institute [7, 8]. The work recorded here is part of our research program started in 1999 on the CMO optimization. Its goal is the development of the underlying mathematical structure of CMO, which will lead to practical rigorous algorithms for structure comparison, prediction and analysis [10, 11, 12].

The mathematical analysis and the algorithmic methods employed in this paper for the maximum CMO problem have feasible generalizations to other basic problems in structure comparison and annotation that do not have at present rigorous algorithms. These include: (1) the computation of the minimal distance similarity measures (used in the DALI classification); (2) the computation of optimal fit of contact-based structural patterns in protein structures, and (3) the development of rigorous assessment tools for analyzing predictions in the new competing category "Contacts" at CASP 2000.

1.1 Contact Maps

A *contact map* is an undirected graph giving a concise representation of the 3D fold of a protein. Each residue of a protein is a node, and there is an edge (called a *contact*) between two nodes if the Euclidean distance between the corresponding residues is within a given threshold. The distance between two residues is defined as the smallest distance between any pair of atoms in the residues.

An alignment between two contact maps specifies the residues that are considered equivalent. The value of an alignment between two contact maps is the number of contacts in the first map whose endpoints (residues) are aligned with residues that are also in contact in the second map. This value is called the *overlap* for the two proteins, and the optimization problem is to find the maximum overlap. Figure 1 shows two contact maps and their alignment.

The contact map overlap (CMO) problem tries to capture the similarity in the 3D folds of two proteins by comparing their contact maps. It was introduced in [7], proved NP-hard in [10], and is emerging as a very important practical measure of protein structure similarity.

1.2 CMO Optimization

The CMO problem can be reduced to a (very large) *Maximum Independent Set* (MIS) problem on a suitable graph. An *independent set* is a set of vertices such that there is no edge between any two of them. The MIS is a classic problem in combinatorial optimization, with large literature. Although its definition is nice and simple, this problem is one of the toughest to solve exactly. Many papers over the years have dealt with the exact solution of the max independent set or, equivalently, the max clique [21, 1, 16], but the state of the art for this problem is that we cannot practically solve instances on dense graphs of more than a couple hundred nodes [16]. This fact makes the result achieved in our work record–setting, since the max independent set problems that we tackle and solve optimally in this paper have size proportional to the product of the number of contacts of the two maps (up to 10,000 nodes and more). Of course, this was only possible by exploiting the particular characteristics of the graph derived from the problem at hand.

Some of the most powerful algorithms for exact solution of combinatorial optimization problems are based on *Integer Programming* (IP), which has been applied profitably in very many cases [20, 3]. The IP approach consists in formulating a problem as the maximization of a linear function of some integer variables and then solving it via branch and bound. The upper bound comes from the *linear programming* (LP) *relaxation*, in which the variables are not restricted to be integer, and is polynomially solvable. When the LP–relaxation value is close to the value over the integers, then the bound, and hence the pruning of the search space, will be effective. In order to obtain better bounds, the formulation is often reinforced by the use of additional constraints, called *cuts* (from which the approach name, *Branch–and–Cut*): these are constraints that do not eliminate any feasible integer solution, but make the space of fractional solutions smaller, this way decreasing the value of the LP bound.

The maximum independent set has a natural formulation as an IP problem: denoting by x_v a binary variable for each vertex v, we seek to maximize $\sum_v x_v$, subject to $x_u + x_v \leq 1$ for all edges $\{u, v\}$. Unfortunately, this

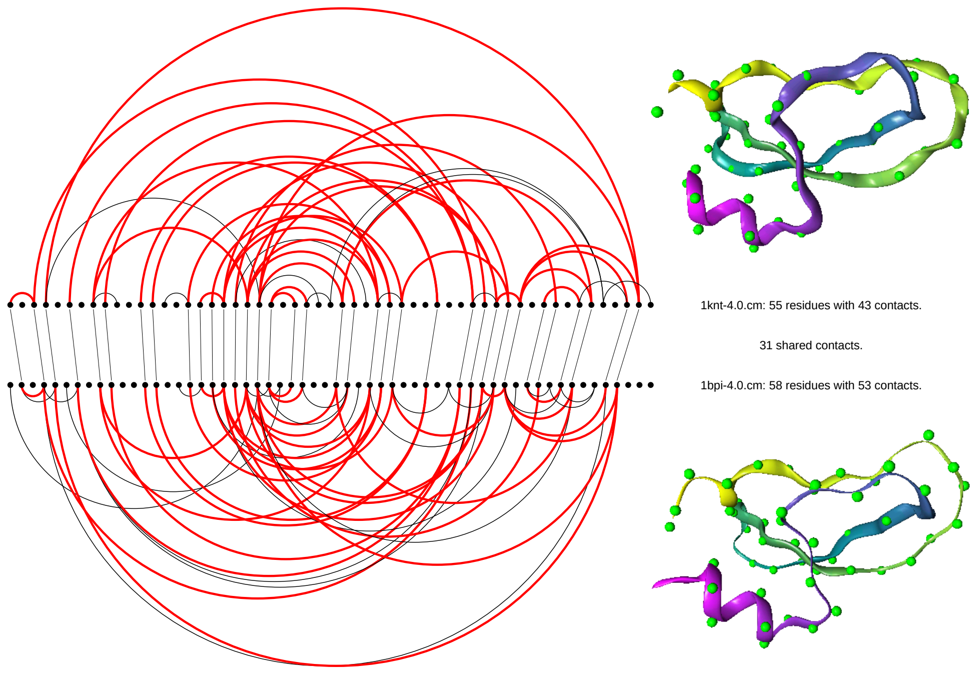
1knt-4.0.cm: 55 residues with 43 contacts.
31 shared contacts.
1bpi-4.0.cm: 58 residues with 53 contacts.

Figure 1: An optimal alignment of two 4Å threshold contact maps of proteins `1bpi` and `1knt`.

formulation gives a very weak bound. The formulation can be strengthened by the use of clique–inequalities cuts, such as $\sum_{v \in Q} x_v \leq 1$, which say that any clique Q can have at most one node in common with any independent set. The addition of these constraints can lead to tight formulations. This is exactly the case for the CMO problem studied here. In this paper we formulate the CMO problem as an Integer Program and solve it by Branch–and–Cut, where the cuts used are mainly clique–inequalities and some other described later. Although we show that there is an exponential number of different clique inequalities, we characterize them completely and show how to separate over them in fast polynomial time. That is, given a fractional solution, we can find in time $O(n^2)$ (where n is the size of the instance) the most violated clique inequality and add it to the LP formulation. By a fundamental result of Grötschel, Lovász and Schrijver [5], this is a necessary and sufficient condition for computing the value of an exponential–sized LP relaxation in polynomial time. Finding cliques in a graph is in general a difficult problem, but we show that in our case we can solve it effectively since the underlying graph is perfect.

1.3 Organization

In the remainder of the paper we will describe in detail our Branch–and–Cut algorithm. In Section 2 we formalize the problem as an Integer Program and describe the main inequalities used. Section 3 is devoted to characterize the clique inequalities, describe a polynomial algorithm for their separation and prove that the underlying graph is perfect.

Each Branch–and–Cut for a maximization problem, must rely on some sort of heuristic algorithm in order to obtain tight lower bounds (i.e. good feasible solutions). In fact, these are used to effectively prune the search space, by removing from consideration each subproblem whose LP upper bound is lower than the best feasible solution found so far. In our work we have implemented many different heuristics, which are run for several iterations at the root node and at each subproblem during the search. Our heuristics fall into two categories: *Steepest Ascent Local Search* and *Genetic Algorithms*. and are described in detail in section 5.

Our program has been run on real data coming from the PDB protein data base. This is the first time that exact solutions have been found for real instances of this problem. We have run our procedure on 597 protein pairs, with sizes ranging from 64 to 72 residues. In order to perform such a massive computation on a standard, single–processor Pentium PC, we have limited the time devoted to each individual instance (details can be found in Section 6 on computational results). Therefore, some problems have not be solved to optimality. However, even within the time limit of 1 hour per instance, we have been able to solve 42 problems optimally and for 362 problems (60 percent) the gap between the best solution found and the current upper bound was less than or equal to 5, thereby providing a strong certificate of near–optimality. These results also show the effectiveness of our lower bounding heuristic procedures, and in particular, of our genetic algorithm.

2. IP FORMULATION

A contact map is a graph giving a concise representation of the 3D fold of a protein. Each residue is a node, and there is an edge between two nodes if their euclidean distance is within a given threshold when the protein is folded. An *alignment* is a mapping, compatible with the linear ordering, of some residues of the first protein into some of the second. The value of an alignment is the number of edges in the first contact map whose endpoints are aligned with residues that also share an edge in the second contact map. The *maximum contact map overlap* problem calls for an alignment of maximum value.

We can rephrase this problem in graph–theoretic language as follows: We are given two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, with $n_i = |V_i|$ and $m_i = |E_i|$, for $i = 1, 2$. A total order is defined on $V_1 = \{a_1 < \ldots < a_{n_1}\}$ and $V_2 = \{b_1 < \ldots < b_{n_2}\}$ (we may identify V_i with $\{1, \ldots, n_i\}$ for $i = 1, 2$). It is customary to draw such a graph with the vertices arranged increasingly on a line. We distinguish a tail and a head for each edge $\{i, j\}$, where the tail is the left endpoint and the head is the right endpoint. Therefore, we denote an edge by an ordered pair (i, j).

A non–crossing map of V_1 in V_2 is defined by any two subsets of the same size k, $\{i_1, \ldots, i_k\} \subseteq V_1$ and $\{u_1, \ldots, u_k\} \subseteq V_2$, where $i_1 < i_2 \ldots < i_k$ and similarly for the u_h's. In this map, u_h is the image of i_h for $1 \leq h \leq k$. Two edges $(i, j) \in E_1$ and $(u, v) \in E_2$ are *shared* by the map if there are $l, t \leq k$ s.t. $i = i_l$, $j = i_t$, $u = u_l$ and $v = u_t$ (see Figure 2). Each pair of shared edges contributes a *sharing* to the objective function. The problem consists in finding the non–crossing map which maximizes the number of sharings. This problem is closely related to the maximum edge–induced common subgraph problem [9, 4], with the additional constraint that the isomorphism of the subgraphs must preserve the ordering of the nodes. Also, a somewhat similar problem is the RNA sequence structure alignment, to which Lenhof, Reinert and Vingron applied and IP approach in [18]. In a different way, they obtain results akin to ours for clique inequalities in their solution space, i.e. RNA letter–letter alignments. Here, we fully characterize cliques via an if–and–only–if characterization, and we show that the underlying graph is a perfect graph. Note that noncrossing maps are in one-to-one correspondence with noncrossing matchings in the complete bipartite graph W having vertex sets V_1 and V_2 and edge set $V_1 \times V_2$.

2.1 IP Formulation

We denote by y_{ef} a binary variable for $e \in E_1$ and $f \in E_2$, which is 1 iff the edges e and f are a sharing in a feasible solution. The objective function is

$$\max \sum_{e \in E_1, f \in E_2} y_{ef}. \tag{1}$$

The sharings (e_1, f_1) and (e_2, f_2) can be both achieved by a noncrossing map if and only if they are *compatible*, i.e. no two of the lines betweens the tails of e_1 and f_1, the tails of e_2 and f_2, the heads of e_1 and f_1 and the heads of e_2 and f_2 intersect at a single point (or, as we will say, *cross*). Then the constraints for the problem are simply

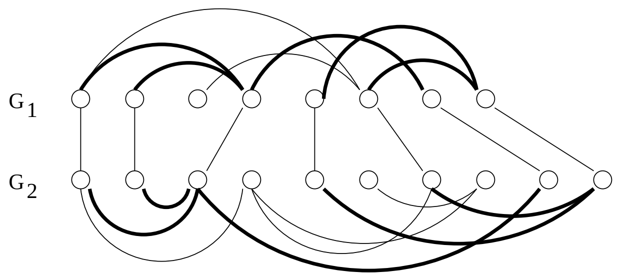

Figure 2: A noncrossing map of value 5

$$y_{e_1 f_1} + y_{e_2 f_2} \leq 1 \tag{2}$$

for all $e_1, e_2 \in E_1$, $f_1, f_2 \in E_2$ s.t. (e_1, f_1) and (e_2, f_2) are not compatible. Although it is possible to list all pairs of incompatible sharings and solve the corresponding IP with this formulation, the resulting LP bound would be very weak unless we strengthen it with cuts, which are hard to separate, and it would also have too many constraints. Therefore we introduce a new set of binary variables x_{iu} for $i \in V_1$ and $u \in V_2$, which represent the actual map, and constraints such that the support graph of x must be a non–crossing matching. Then, we bound the y variables by means of the x variables so that the edges (i, j) and (u, v) can be shared only if i is mapped to u and j to v. For $i \in V_1$ (and analogously for $i \in V_2$), denote by $\delta^+(i)$ the set $\{j \in i+1, \ldots, n_1 : (i, j) \in E_1\}$ and $\delta^-(i) = \{j \in 1, \ldots, i-1 : (j, i) \in E_1\}$. Then we have the following constraints:

$$\sum_{j \in \delta^+(i)} y_{(i,j)(u,v)} \leq x_{iu} \tag{3}$$

$$\sum_{j \in \delta^-(i)} y_{(j,i)(u,v)} \leq x_{iv} \tag{4}$$

for all $i \in V_1$, $(u, v) \in E_2$, and of course analogous constraints for $i \in V_2$ and $(u, v) \in E_1$. We call these *activation* constraints. Finally, the *noncrossing* constraints are of the form:

$$x_{iu} + x_{jv} \leq 1 \tag{5}$$

for all $1 \leq i \leq j \leq n_1$, $1 \leq v \leq u \leq n_2$ s.t. $i \neq j \vee u \neq v$. We define a relation of compatibility for the x variables similarly as for the y. A matching in W is a set of lines connecting nodes of V_1 and V_2 in the usual drawing of W, with V_1 drawn on the top and V_2 on the bottom. We denote such a line for $i \in V_1$ and $j \in V_2$ by $[i, j]$. We say that two lines: *cross* if their intersection is a point; *strictly* cross if they cross at a point other than an endpoint; are *compatible* if they do not cross. A set of sharings is *feasible* if they are all mutually compatible, otherwise it is *infeasible*. Similarly we define a feasible and infeasible set of lines. If we draw the lines connecting the endpoints of an infeasible set of sharings, we have an infeasible set of lines.

The connection to the independent set problem is now clear. We define two graphs G_x and G_y as follows. In G_x there is a node N_{iu} for each line $[i,u]$ with $i \in V_1$ and $u \in V_2$ and two nodes N_{iu} and N_{jv} are connected by an edge iff $[i,u]$ and $[j,v]$ cross. Similarly, in G_y there is a node N_{ef} for each $e \in E_1$ and $f \in E_2$ and two nodes N_{ef} and $N_{e'f'}$ are connected by an edge iff the sharings (e,f) and (e',f') are not compatible.

Then a selection of x variables feasible for all non-crossing constraints corresponds to an independent set in G_x and a feasible set of sharings is an independent set in G_y. The maximum independent set in G_y is the solution sought after. All cuts valid for the independent set problem can be applied to the x and y variables. The most notable cuts for the independent set problem are the clique inequalities: Any independent set and any clique can have at most one element in common. Another (weaker) class of inequalities are the *odd–hole inequalities*: Any odd cycle of length $2l+1$ can have at most l nodes in an independent set. We will show that G_x has no odd holes, while G_y may contain them.

Adding clique inequalities presents two types of difficulties. First, findig cliques in a graph is in general a hard problem itself. Second, there may be exponentially many different cliques. We can overcome these difficulties if we have a *separation oracle*: that is a black box that, given a solution to an LP with only some of the clique inequalities, tells us if the solution is in fact feasible for all the clique inequalities, or else returns us a violated clique inequality.

In the next section we will describe an $O(n^2)$ separation oracle for the exponentially large ($O(2^{2n})$) set of all cliques in the x variables. Although we do not characterize all cliques for the y variables, we identify several classes of cliques in G_y and show that satisfying the clique inequalities for the x variables implies also satisfying the clique inequalities for the y variables for all but two classes of cliques. Furthermore, the cuts for x–cliques are so strong that adding the two non-implied classes of y–cliques yields only a tiny improvement in the bound, while increasing the running time in a way that makes their use dispensable. For these reasons, we focus primarily on cliques in the x variables, i.e. sets of lines in the bipartite graph W which are all mutually crossing.

Our final IP formulation for the max CMO problem is given by Equations (1), (3), (4) and (5), where x_{ij} and y_{ef} are binary variables for all $i \in V_1, j \in V_2, e \in E_1, f \in E_2$. The formulation is strengthened by the aforementioned cuts.

3. POLYNOMIAL SEPARATION OF MAXIMAL CLIQUES

In this section we study the problem of characterizing all cliques of G_x, i.e. sets of lines in the bipartite graph W which are all mutually crossing.

We define the following notion of a *triangle* in W. $T(i,j|u) := \{[i,u],[i+1,u],\ldots,[j-1,u],[j,u]\}$ where $i \le j \in V_1$ and $u \in V_2$, and $T(i|j,u) := \{[i,j],[i,j+1],\ldots,[i,u-1],[i,u]\}$ where $i \in V_1$ and $j \le u \in V_2$. Clearly a triangle corresponds to a clique, so that

$$x(T(i,j|u)) \le 1 \quad \text{and} \quad x(T(i|j,u)) \le 1 \qquad (6)$$

are valid inequalities for each i, j and u. These includes the standard matching constraints, which are just $T(i|1,n_2)$ and $T(1,n_1|u)$. We have the following lemma.

LEMMA 1. *Given a fractional LP solution x^*, we can compute the value of any t triangles in time $O(n^2+t)$.*

Proof We start with a preprocessing in which we compute $x^*(T(i|1,v))$ and $x^*(T(1,i|v))$ for each $i \in V_1$ and $v \in V_2$ in total time $O(n^2)$. This is done by first fixing i, letting $x^*(T(i|1,1)) = x^*_{i1}$ and noticing that $x^*(T(i|1,v)) = x^*(T(i|1,v-1)) + x^*_{iv}$ (and similarly we obtain all $x^*(T(1,i|v))$). Now, given say any $i_1 \le i_2 \in V_1$ and $u \in V_2$, we get $x^*(T(i_1,i_2|u)) = x^*(T(1,i_2|u)) - x^*(T(1,i_1-1|u))$ in time $O(1)$. ◇

We call the algorithm computing the $O(n^2)$ basic triangles of the proof SETUP. It will be the preliminary step to our fast separation algorithms.

Call a_1, a_{n_1}, b_1, and b_{n_2} the set of *terminal* nodes. Consider a path P which passes through all the terminal nodes, and alternates nodes of V_1 and V_2 in a zig–zag fashion: That is, we can orient the path so that a_1 is the first of the nodes of V_1 visited by the path, and if a_k has been visited by the path, then all of the nodes in V_1 visited after a_k are to its right. Similarly, b_{n_2} is the first of the nodes of V_2 visited by the path, and if b_h has been visited by the path, then all of the nodes in V_2 visited after b_h are to its left. Note that any such path must start and end at a terminal node (see figure 3), and must always include the lines $[a_1, b_{n_2}]$ and $[a_{n_1}, b_1]$. There are only two possibilities after we orient the path as described before: The path starts at a_1 and b_{n_2} is the second node or it starts at b_{n_2} and a_1 is the second node. For each node of degree two in P a triangle is defined by considering the set of lines incident on the node and contained within the two lines of the path. Let $T_A(P)$ ($T_B(P)$) be the set of triangles defined by P with tip in the nodes of V_1 (V_2) having degree two in P. We define $T(P) := T_A(P) \cup T_B(P)$. The following theorem characterizes completely all cliques in G_x.

THEOREM 1. *A set Q of lines is a maximal clique in G_x if and only if there exists a zigzag path P such that $Q = T(P)$.*

Proof (Sketch) *If:* Let Q be a set of lines and P a zigzag path such that $Q = T(P)$. Take any two lines in $T(P)$ and show they cross. Hence $T(P)$ is a clique. Now, for any line not in $T(P)$ show there is a line in $T(P)$ "parallel" to it. Hence, $T(P)$ is a maximal clique. *Only if:* Let Q be a maximal clique. Let $a_1^Q < \ldots < a_k^Q$ be the nodes of V_1 in Q. For each t, consider the leftmost and the rightmost lines of Q out of a_t^Q. Show that the union of these lines defines a zigzag path P, and $Q \subset T(P)$ by construction. But $T(P)$ is a clique, and Q is maximal. So $Q = T(P)$. ◇

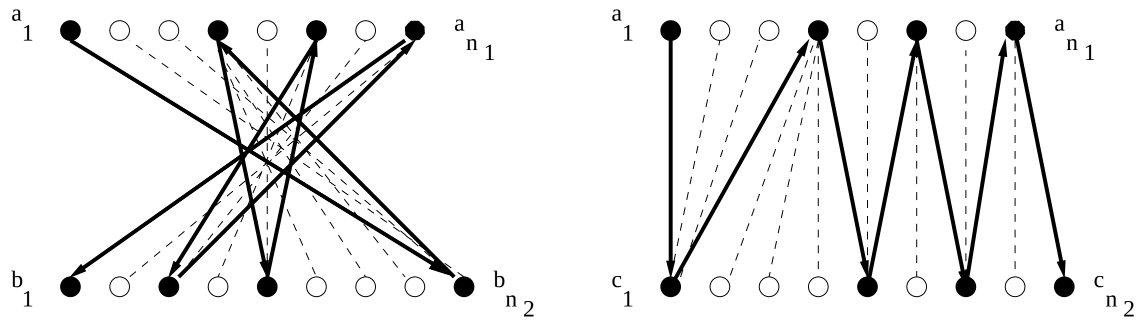

Figure 3: Left: A zigzag path P (bold) and the set $T(P)$. Right: Same path after flipping $V2$.

The inequalities $x(T(P)) \leq 1$ for all zigzag paths P are therefore the strongest clique cuts for this particular independent set problem. We now show that they can be separated in time $O(n^2)$. To make the following argument easier, we rename the nodes of V_2 as $\{c_1, \ldots, c_{n_2}\}$, so that the leftmost node c_1 is b_{n_2} and the rightmost, c_{n_2}, is b_1 (i.e., we flip the nodes of V_2 with respect to the usual drawing). Having done this, two lines were not crossing in the original drawing of W if and only if now they are *strictly* crossing. Furthermore, a zigzag path P now looks as a path wich goes from left to right both in V_1 and V_2. We call such a path a *leftright* path.

It is easy to see that there are $O(2^{2n})$ leftright paths. However, there is a low–degree polynomial algorithm for finding the leftright path P with largest $x^*(T(P))$, making separation of this class of inequalities very effective in the practical solution of the problem. With respect to the new drawing of W, orient each line $[a, c]$ in the two possible ways and then define the length for each arc $(a, c) \in V_1 \times V_2$ and $(c, a) \in V_2 \times V_1$ as follows: $l(a, c) = x^*(T(a|1, c)) - x^*(T(1, a|c))$ and $l(c, a) = x^*(T(1, a|c)) - x^*(T(a|1, c))$. The lengths of four special arcs are defined separately, as $l(a_1, c_1) = 0$, $l(c_1, a_1) = 0$, $l(a_{n_1}, c_{n_2}) = x^*(T(a_{n_1}|1, c_{n_2}))$ and $l(c_{n_2}, a_{n_1}) = x^*(T(1, a_{n_1}|c_{n_2}))$. Now, consider a leftright path P starting with either the arc (a_1, c_1) or (c_1, a_1) and ending with either the arc (a_{n_1}, c_{n_2}) or (c_{n_2}, a_{n_1}). Call $l(P)$ the standard length of this path, i.e. the sum of arcs lengths. We then have the following lemma.

Lemma 2. *For a leftright path P, $l(P) = x^*(T(P))$.*

Proof Suppose $P = (a_1, c_{h_1} = c_1, a_{j_2}, c_{h_2}, \ldots, a_{j_l} = a_{n_1}, c_{h_l} = c_{n_2})$. A similar argument applies in the other three cases. From tedious but simple algebra, we have $l(P) =$

$$
\begin{aligned}
&= l(a_1, c_1) + \textstyle\sum_{t=1}^{l-1} l(c_{h_t}, a_{j_{(t+1)}}) \\
&\quad + \textstyle\sum_{t=2}^{l-1} l(a_{j_{(t+1)}}, c_{h_{(t+1)}}) + l(a_{n_1}, c_{n_2}) \\
&= \textstyle\sum_{t=1}^{l-1} \Big(x^*(T(1, a_{j_{(t+1)}}|c_{h_t})) - x^*(T(a_{j_{(t+1)}}|1, c_{h_t})) \Big) \\
&\quad + \textstyle\sum_{t=2}^{l-1} (x^*(T(a_{j_t}|1, c_{h_t})) - x^*(T(1, a_{j_t}|c_{h_t}))) \\
&\quad + x^*(T(a_{n_1}|1, c_{n_2})) \\
&= \textstyle\sum_{t=1}^{l-1} x^*(T(1, a_{j_{(t+1)}}|c_{h_t})) - \textstyle\sum_{t=1}^{l-1} x^*(T(a_{j_{(t+1)}}|1, c_{h_t})) \\
&\quad + \textstyle\sum_{t=2}^{l-1} x^*(T(a_{j_t}|1, c_{h_t})) - \textstyle\sum_{t=2}^{l-1} x^*(T(1, a_{j_t}|c_{h_t})) \\
&\quad + x^*(T(a_{n_1}|1, c_{n_2})) \\
&= x^*(T(1, a_{j_2}|c_{h_1})) + \textstyle\sum_{t=2}^{l-1} x^*(T(1, a_{j_{(t+1)}}|c_{h_t})) \\
&\quad - \textstyle\sum_{t=1}^{l-2} x^*(T(a_{j_{(t+1)}}|1, c_{h_t})) - x^*(T(a_{j_l}|1, c_{h_{(l-1)}})) \\
&\quad + \textstyle\sum_{t=2}^{l-1} x^*(T(a_{j_t}|1, c_{h_t})) - \textstyle\sum_{t=2}^{l-1} x^*(T(1, a_{j_t}|c_{h_t})) \\
&\quad + x^*(T(a_{n_1}|1, c_{n_2})) \\
&= x^*(T(1, a_{j_2}|c_{h_1})) + \textstyle\sum_{t=2}^{l-1} x^*(T(a_{j_t} + 1, a_{j_{(t+1)}}|c_{h_t})) \\
&\quad + \textstyle\sum_{t=2}^{l-1} x^*(T(a_{j_t}|1, c_{h_{(t-1)}} + 1, c_{h_t})) \\
&\quad - x^*(T(a_{j_l}|1, c_{h_{(l-1)}})) + x^*(T(a_{n_1}|1, c_{n_2})) \\
&= x^*(T(1, a_{j_2}|c_{h_1})) + \textstyle\sum_{t=2}^{l-1} x^*(T(a_{j_t} + 1, a_{j_{(t+1)}}|c_{h_t})) \\
&\quad + \textstyle\sum_{t=2}^{l-1} x^*(T(a_{j_t}|1, c_{h_{(t-1)}} + 1, c_{h_t})) \\
&\quad + x^*(T(a_{n_1}|c_{h_{(l-1)}} + 1, c_{n_2})) \\
&= x^*(T(P)).
\end{aligned}
$$

The same kind of computations can be carried for the other three possibilities for starting/ending arcs. ◇

The longest leftright path can be found effectively. In fact we have

Theorem 2. *There is an $O(n^2)$ algorithm for finding the longest leftright path in a complete bipartite oriented graph.*

Proof We use dynamic programming to find such a path. Call $V_{\searrow}(i, j)$ the length of a longest leftright path starting at a_i and using nodes of V_2 only within $c_j, c_{j+1}, \ldots, c_{n_2}$. Also, call $V_{\nearrow}(i, j)$ the length of a longest zigzag path starting at c_j and using nodes of V_1 only within $a_i, a_{i+1}, \ldots, a_{n_1}$. We have the following recurrences:

$$V_{\searrow}(i, j) = \max\{l(a_i, c_j) + V_{\nearrow}(i + 1, j), V_{\searrow}(i, j + 1)\},$$

$$V_{\nearrow}(i, j) = \max\{l(c_j, a_i) + V_{\searrow}(i, j + 1), V_{\nearrow}(i + 1, j)\}.$$

The only boundary conditions we need are the values of $V_{\searrow}(n_1, n_2)$ and $V_{\nearrow}(n_1, n_2)$, which are easily set to $V_{\searrow}(n_1, n_2) = l(a_{n_1}, c_{n_2})$ and $V_{\nearrow}(n_1, n_2) = l(c_{n_2}, a_{n_1})$. The recurrence can be then solved backwards starting at (n_1, n_2), in time $O(n^2)$. At the end, $V_{\searrow}(1, 2)$ is the length of the longest leftright path starting with arc (c_1, a_1) and $V_{\nearrow}(2,1)$ is the length of the longest leftright path starting with arc (a_1, c_1). The maximum of the two is the longest leftright path. $\diamond$

The following corollary says that we have a polynomial oracle for the clique relaxation of CMO.

COROLLARY 1. *There is an $O(n^2)$ algorithm for separating the class of all maximal clique inequalities.*

Proof From theorem 1, a clique inequality is violated if and only if there is a zigzag path P such that $x^*(T(P)) > 1$. Since a zigzag path corresponds to a leftright path via lemma 2, we can simply find the longest leftright path and check if it has length > 1. By lemma 1, we can compute in time $O(n^2)$ the lengths l for all the arcs of the complete bipartite oriented graph since each arcs requires the value of only two triangles. Together with theorem 2, this concludes the proof. $\diamond$

4. OTHER VALID INEQUALITIES AND STRENGTH OF RELAXATION

The odd–holes inequalities for the independent set problem say that for any odd–hole C there can be at most $\lfloor |C|/2 \rfloor$ nodes in an independent set. The fact that we can find (weighted) cliques in G_x in polynomial time hinted us to proving that G_x is in fact perfect. In this case, G_x has no odd holes. A graph G is *weakly triangulated* if neither G nor G^c have (induced) chordless cycles of length greater than four Note that in G_x there may be holes of size 4 (G_x is not chordal); e.g. $[a_1, b_3]$, $[a_2, b_4]$, $[a_3, b_1]$ and $[a_4, b_2]$ define a chordless cycle. A result by Hayward ([13]) states that weakly triangulated graphs are perfect.

THEOREM 3. *The graph G_x is weakly triangulated.*

Proof We have to prove that there are no chordless cycles of length ≥ 5 in (i) G_x and (ii) G_x^c.

(i) Consider a cordless cycle of length $k \geq 5$ in G_x. It corresponds to a set $l_1, \ldots, l_k$ of lines in W, such that l_i crosses l_{i-1} and l_{i+1} only. Since l_1 does not cross l_3, wlog assume l_3 lies completely to the right of l_1. We distinguish two cases. If $k \geq 6$, for $i = 4, \ldots, k-1$, since l_i crosses l_{i-1} but does not cross l_1, also l_i lies completely to the right of l_1. Use the same argument starting from l_{k-1} and knowing that l_1 is completely to its left. Then for $i = 2, \ldots, l_{k-3}$ we deduce that the line l_i is completely to the left of l_{k-1}. Hence the nonempty set $L = \{l_3, \ldots, \ldots, l_{k-3}\}$ lies completely within the lines l_1 and l_{k-1}. But l_k crosses both l_1 and l_{k-1} and so it must cross all the lines in L. So l_k cannot have degree 2 in the cycle. If $k = 5$ we reason as follows. l_1 does not cross l_3, l_3 does not cross l_5 but l_5 crosses l_1. So l_3 is to the right of both l_1 and l_5, written $3 \in R(1,5)$. Then, since l_5 does not cross l_2 nor l_3 but l_2 crosses l_3, l_5 is to the left of both l_2 and l_3, i.e. $5 \in L(2,3)$. Continuing we get $2 \in R(4,5)$, $4 \in L(1,2)$ and $1 \in R(3,4)$. A contradiction, since we started with l_3 to the right of l_1 and ended with l_1 to the right of l_3.

(ii) We start by embedding the contact maps of each of the two proteins into the plane as follows. The vertices of the first protein V_1 are placed on a horizontal line according to their order, with the first amino acid in the protein being the left-most vertex. The contacts between vertices are then drawn in as curved edges, but do not affect G_x. The vertices of the second protein V_2 are placed on a horizontal line below this first line and also according to their order in the protein.

The vertices of G_x correspond exactly to the edges in the embedded complete bipartite graph K_{n_1,n_2}. Two vertices in G_x are adjacent whenever their corresponding edges cross. We wish to eliminate the case where the edges intersect at a point (i.e. do not strictly cross) to make our analysis easier. We do this by constructing a graph isomorphic to G_x from a non-complete bipartite graph G_{n_1,n_2} with more vertices than K_{n_1,n_2} as follows. Make a group of nodes for each vertex in V_1 that consists of n_2 copies of that vertex in V_1, and likewise form a group of n_1 copies of a vertex for each vertex in V_2. We do not overlap any of these vertex groups and maintain the order of these vertex groups on the line. Then an edge ij in K_{n_1,n_2} which is the k_1st edge from the left incident to i and is the k_2nd edge from the right incident to j is assigned the new endpoints of the k_1st rightmost copy of i and the k_2nd leftmost copy of j. Then the graph isomorphic to G_x is obtained by considering the crossing edges of G_{n_1,n_2} analogously as in our first construction of G_x.

After having constructed G_x from G_{n_1,n_2}, let an antihole of size 5 or more be given. Denote the vertex in this antihole whose corresponding edge has the leftmost endpoint in V_1 by l. Denote its neighboring vertices by $l-2, l-1, l+1, l+2$ consistently with the order that these 5 vertices appear in this antihole. The vertices $l, l+1$, etc. correspond to edges in G_{n_1,n_2}, whose endpoints in V_1 and V_2 are denoted by $l_1, l_2, (l+1)_1, (l+1)_2$, etc. Since the edges for l and $l+2$ must intersect, and the edge for $l+1$ must not intersect either of these, the left-to-right order in V_1 for $l_1, (l+1)_1, (l+2)_1$ must be $l_1, (l+2)_1, (l+1)_1$. Also, the left-to-right order of the endpoints in V_2 must be $(l+2)_2, l_2, (l+1)_2$. The edge for $l-1$ must intersect the edges for $l+1$ and $l+2$, but not the edge for l. Hence, the new left-to-right orders are $l_1, (l-1)_1, (l+2)_1, (l+1)_1$ and $(l+2)_2, l_2, (l+1)_2, (l-1)_2$. The edge for $l-2$ is required to intersect the edges for l and $l+1$. As a result, this edge will also intersect the edge for $l-1$, which contradicts the definition of an antihole.

$\diamond$

Since G_x is perfect, it is no surprise we could find weighted cliques in polynomial time. In fact, there are algorithms for finding a max weighted clique in a weakly triangulated graph of time $O(|V|^5)$, due to Hayward, Hoang, Maffray [14] and Raghunathan [22]. Our $O(n^2)$ result for this specific graph makes a huge difference in the practical solution of the problem. Finally, we note that since G_x is perfect, the clique inequalities and non-negativity provide a complete polyhedral description for the non-crossing bipartite matching polytope that the

x variables are constrained to be in.

The situation is different as far as the graph G_y is concerned. In fact, G_y can contain odd holes. There is a known polynomial time algorithm for separating odd-holes ([20]), which we used in our code. Finally, we mention some cliques for the y variables. Consider two edges both in the same graph. They can either *(i)* have no endpoint in common and not intersect, or *(ii)* have one common endpoint or *(iii)* no endpoint in common and intersect. For an edge e and R one of *(i), (ii), (iii)*, call $R(e)$ the set of edges which are in the relation R with e. Then, in any feasible solution in which e is mapped into e' and f is mapped into f', f' must be in the same relationship to e' as f is to e. That is, if $f \in R(e)$ then the set $\{\{e,f\}\} \cup \{\{e',g\} : g \notin R(e')\}$ is a clique in G_y. It can be proved that the clique inequalities relative to the cases *(ii)* and *(iii)* are implied by the activation constraints together with the matching constraints for the nodes, and hence cannot be used as cuts. The remaining inequalities however are not implied. Our computational experiments have shown that these clique inequalities in the y variable are actually very weak, and very seldom does their use give an improvement to the bound value.

5. LOWER BOUND HEURISTICS

5.1 Genetic algorithms

The genetic algorithm (GA) paradigm has been described in great detail elsewhere [25, 24]. Classical applications of the GA encode candidate solutions as bit strings (the genome), which is then decoded for evaluation. However, unless great care is taken in choosing an encoding scheme, this method will frequently produce infeasible solutions. Instead of using the bit as the fundamental element in our genome, we use the alignment edge. An alignment edge associates a residue in one protein with a residue in the other. *Mutation* is the operation of translating one side of a set of contiguous edges. For each edge, the probability of mutation is tested. If the test passes, a mutation is performed by randomly choosing a protein (top or bottom), a set of edges (all edges to the left or right of the current edge), a direction to shift in (left or right) and a distance to shift. After the shift is performed, new edges are added in any available positions. In Figure 4 we show two mutations: The top alignment shows the initial state, the middle alignment shows the state after the first mutation and before the second mutation, the bottom alignment shows the state after both mutations. Dashed lines are edges that have been added after a mutation is performed, dotted lines are edges that have been removed by a mutation. The first mutation shifts the circled edges to the right by one position, causing the right-most edge to be removed and a new edge the be inserted. The second mutation shifts the circled edges to the left by one position, causing the left-most shifted edge to be removed and a new edge to be randomly inserted on the right end – exactly one of the dashed lines is inserted.

Recombination merges edges from two parents to create a new candidate solution. First, a random set of

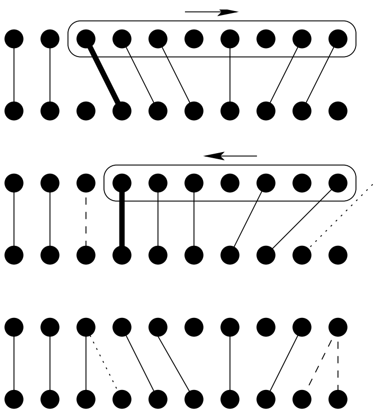

Figure 4: The mutation operator.

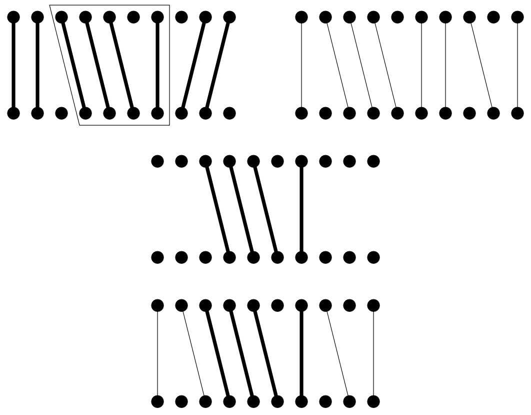

Figure 5: The recombination operator.

contiguous edges is copied from a parent to the child. Then, as many edges as possible are copied from the other parent into the child. Finally, new edges are added in any available positions. An example is shown in Figure 5. The top two alignments are the two parents, the middle alignment is the partially constructed child and the bottom alignment is the fully constructed child.

5.2 Local Search

The local search heuristic algorithms follow the standard approach: Given a feasible solution s of value $v(s)$, a *neighborhood* $N(s)$ of solutions which can be obtained by s through a valid *move* is explored. Let $s' \in N(s)$ such that $v(s') = \max\{v(u), u \in N(s)\}$. If $v(s') > v(s)$, s' replaces s and the search continues, otherwise s is a local optimum. Since converging to a local optimum is very fast, the search can be repeated many times, each time starting from a random feasible solution.

In the contact map problem, a feasible solution is identified by a pair (A, B) of sets of vertices in G_1 and G_2 such that $|A| = |B|$. If $A = \{a_1 < \ldots < a_k\}$ and $B = \{b_1 < \ldots < b_k\}$, the feasible solution they define is that in which a_i is mapped to b_i, for $i = 1, \ldots, k$.

We have two algorithms, which differ in the way $N(s)$

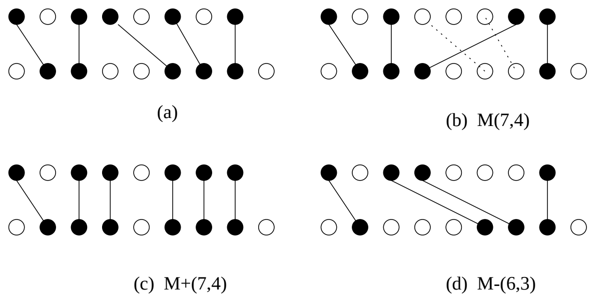

Figure 6: Moves for the two algorithms. (a) starting solution s. (b) solution $s' \in N(s)$ for 1st algorithm. (c) and (d) solutions $s' \in N(s)$ for 2nd algorithm.

is defined. In the first algorithm, a move $M(a, b)$ involves two elements $a \in G_1 - A$ and $b \in G_2 - B$. The move adds the line $[a, b]$ to the current mapping, and so some lines (i.e. those indexed by J, where $[a_j, b_j]$ crosses $[a, b]$ for all $j \in J$), must be removed. The new solution is $(A \cup \{a\} - \{a_j, j \in J\}, B \cup \{b\} - \{b_j, j \in J\})$. This allows for big "jumps" in the solution space (i.e. removal of many lines, and introduction of very skewed lines) and is suitable for instances in which the contact maps are quite different (Figure 6(b)).

While in the first algorithm the number of lines after a move can increase, decrease or stay the same, in the second we have increasing moves and decreasing moves. An increasing move $M^+(a, b)$ is defined, as before, for $a \notin A$ and $b \notin B$. Such move simply changes (A, B) into $(A \cup \{a\}, B \cup \{b\})$ (Figure 6(c)). A decreasing move $M^-(a, b)$ is defined for $a \in A$ and $b \in B$. The new solution is $(A - \{a\}, B - \{b\})$ (Figure 6(d)). This algorithm does not introduce very skewed lines easily and so is suited for similar proteins, in which good solutions are made of many parallel (and typically vertical) lines.

6. COMPUTATIONAL RESULTS

6.1 Branch–and–Cut

We ran our branch–and–cut program on a set of 269 proteins with 64 to 72 residues and 80 to 140 contacts each, selected from the Protein Data Bank. The set was chosen to contain both a large number of similar proteins, as well as a large number of dissimilar proteins. An all-against-all computation would have resulted in 36046 alignments; we selected a subset of 597 alignments, so that there would be roughly as many pairs of similar proteins as well as dissimilar. The results are reported in Table 7. We set a maximum limit of 1 hour or 15 nodes in the search tree per instance. We also set a minimum limit of 1 node in the search tree (i.e. at least one LP is solved even if takes more than 1 hour). We found that the LP time at each search node could take from 1 minute to 2 hours, depending on the instance size and similarity of the proteins (the more similar, the easier the LPs). In Table 7 we sort the instances by the value of the final gap (gap between best solution found from a heuristic and the upper bound, 0=optimal solution). For each gap level, we report total instances, average and max number of residues (n) and contacts (m), and performance of the heuristics, which are run at each node of the search tree for a maximum time of 5 minutes for all nodes. For each heuristic, we report how many times it found the final best solution. From the table, it appears that the GA heuristic is clearly superior to the others.

We have been able to solve 42 problems optimally and for 362 problems (60 percent) the gap between the best solution found and the current upper bound was less than or equal to 5, thereby providing a strong certificate of near–optimality.

6.2 The Skolnick clustering test

The hybrid heuristic approach that worked well for providing lower bounds in the branch and cut algorithm was also tried on large proteins in a test set suggested by Jeffrey Skolnick. It involved 33 proteins classified into four families: 1) Flavodoxin-like fold CheY-related, 2) Plastocyanin, 3) TIM Barrel, and 4) Ferratin.

We evaluated the validation of this clustering using our structure alignments. We performed 528 alignments and the cluster was retrieved with 1.3% false negatives, and with 0% false positives. Considering the percentagte of the number of contacts in the alignment with respect to the number of contacts in the smaller protein, the range 0.314 – 0.99 contained 119 "within clusters" alignments with a 0% false negatives; the range 0.0 – 0.313 contained 409 "between clusters" alignments with a 1.3% false negatives.

7. REFERENCES

[1] E. Balas and C. S. Yu, Finding a maximum clique in an arbitrary graph, *SIAM J. on Comp.*, 15(4):1054–1068, 1986.

[2] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I.N. Shindyalov, P.E. Bourne, The Protein Data Bank, *Nucleic Acids Research*, 28 pp. 235-242, 2000.

[3] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank and A. Schrijver, *Combinatorial Optimization*, John Wiley and Sons, New York, 1998.

[4] P. Crescenzi and V. Kann, *A compendium of NP optimization problems*, `http://www.nada.kth.se/~viggo`, the web.

[5] M. Grötschel, L. Lovász and A. Schrijver, "The Ellipsoid Method and its Consequences in Combinatorial Optimization", *Combinatorica* 1 (1981), 169–197.

[6] A. Godzik, The structural alignment between two proteins: Is there a unique answer ?, *Protein Science*, 5:1325-1338, 1996.

[7] A. Godzik, J. Sklonick and A. Kolinski, A topology fingerprint approach to inverse protein folding problem, *J. Mol. Biol.*,227:227–238, 1992.

[8] A. Godzik and J. Skolnick, Flexible algorithm for direct multiple alignment of protein structures and sequences, *CABIOS*, 10, (6) 587–596, 1994.

[9] Garey and Johnson, *Computers and intractability: A Guide to the Theory of NP–Completeness*, Freeman, 1979.

Gap=	0	1	2	3	4	5	>5
n. inst	42	48	72	71	76	95	193
avg n	66.4	66.8	66.7	67.0	67.03	66.8	66.8
max n	69	72	71	72	71	72	72
avg m	61.1	56.3	57.3	59.7	61.5	64.7	71.4
max m	92	89	93	95	88	89	133
GA	38	44	63	61	64	74	155
LS1	25	20	35	31	33	35	82
LS2	5	0	0	1	5	12	53

Figure 7: Branch–and–Cut performance. More than one heuristic can find the best value.

Family	Style	Residues	Seq. Sim.	RMSD	Proteins
1	alpha-beta	124	15-30%	< 3Å	1b00, 1dbw, 1nat, 1ntr, 1qmp, 1rnl, 3cah, 4tmy
2	beta	99	35-90%	< 2Å	1baw, 1byo, 1kdi, 1nin, 1pla, 2b3i, 2pcy, 2plt
3	alpha-beta	250	30-90%	< 2Å	1amk, 1aw2, 1b9b, 1btm, 1hti, 1tmh, 1tre, 1tri, 1ydv, 3ypi, 8tim
4		170	7-70%	< 4Å	1b71, 1bcf, 1dps, 1fha, 1ier, 1rcd

Figure 8: The Skolnick set.

[10] D. Goldman, S. Istrail and C. Papadimitriou, Algorithmic Aspects of Protein Structure Similarity, *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, 512–522, 1999.

[11] D. Goldman, PhD. Thesis, Dept. of Computer Science, UC Berkeley, 2000.

[12] A. Lucas, K. Dill and S. Istrail, Contact maps and the computational statistical mechanics aspects of protein folding (in preparation).

[13] R. B. Hayward, Wealky Triangulated Graphs, *J. of Comb. Theory, Series B*, (39)200–209, 1985.

[14] R.B. Hayward, C. Hoang and F. Maffray, Optimizing Wealky Triangulated Graphs, *Graphs and Combinatorics*, 1987.

[15] L. Holm and C. Sander, 3-D lookup: fast protein structure searches at 90% reliability, *Proceedings of the ISMB 1995*, p. 179-187, AAAI., 1995.

[16] D. S. Johnson and M. A. Trick eds, *Cliques, Coloring, and Satisfiability*, Dimacs Series in Discrete Mathematics and Theoretical Computer Science, the American Mathematical Society, 1996.

[17] Kabash-W., A solution for the best rotation to relate two sets of vectors, *Acta Cryst.* A32, 922-923, 1978.

[18] H. P. Lenhof, K. Reinert, M. Vingron, A Polyhedral Approach to RNA Sequence Structure Alignment, *J. Comp. Biol.*, 5(3):517–530, 1998.

[19] A. Lesk, 11th Lipari International Summer School in Computational Biology, 1999.

[20] G. L. Nemhauser and L. Wolsey, *Integer and Combinatorial Optimization*, J. Wiley and Sons, 1988.

[21] G. L. Nemhauser and L. E. Trotter, Vertex packings: Structural properties and algorithms, *Mathematical Programming*, 8:232–248, 1975.

[22] A. Raghunathan, Algorithms for Weakly Triangulated Graphs, UC. Berkeley, Tech. Rep. CSD-89-503, 1989.

[23] I. Eidhammer and I. Jonassen and W. R. Taylor, Structure Comparison and Structure Prediction, to appear *J. Comp. Biol.*, x(x), 2000.

[24] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning. Reading: Addison-Wesley, 1989.

[25] J. H. Holland, Adaptation in Natural and Artificial Systems, Cambridge, MA: MIT Press, 1992.

Acknowledgments

Thanks to Alberto Caprara for useful discussions, and to Ross Lippert for helping throughout the project. The work of the first author was partially supported through project MURST on Computational Biology, while visiting Sandia National Labs, Albuquerque, NM.