1001 optimal PDB structure alignments: Integer Programming methods for finding the maximum contact map overlap

Alberto Caprara* Robert Carr† Sorin Istrail‡ Giuseppe Lancia§

Brian Walenz¶

October 29, 2003

Abstract

Protein structure comparison is a fundamental problem for structural genomics, with applications to drug design, fold prediction, protein clustering and evolutionary studies. Despite its importance, there are very few rigorous methods and widely accepted similarity measures known for this problem.

In this paper we describe the last few years developments on the study of an emerging measure, the *Contact Map Overlap* (CMO), for protein structure comparison. A *contact map* is a list of pairs of residues which lie in 3-dimensional proximity in the protein's native fold.

Although in principle computationally hard to optimize, we show how this measure can in fact be computed with great accuracy for related proteins by Integer Linear Programming techniques. These methods have the advantage of providing certificates of near-optimality by means of upper bounds to the optimal alignment value. We also illustrate effective heuristics, such as Local Search and Genetic Algorithms.

We were able to obtain for the first time optimal alignments for large similar proteins (about 1000 residues and 2000 contacts) and used the CMO measure to cluster proteins in families. The clusters

*D.E.I.S., Università di Bologna, Viale Risorgimento, 2 40136 Bologna, Italy, acaprara@deis.unibo.it

†P.O. Box 5800, MS 1110, Sandia National Laboratories, Albuquerque, NM, 87185 USA, bobcarr@cs.sandia.gov

‡Informatics Research, Celera/Applied Biosystems, 45 W. Gude Drive, Rockville, MD, 20850, sorin.istrail@celera.com

§D.I.M.I., Università di Udine, Viale delle Scienze 206, 33100 Udine, Italy, lancia@dimi.uniud.it, Corresponding author

¶Informatics Research, Celera/Applied Biosystems, 45 W. Gude Drive, Rockville, MD, 20850, brian.walenz@celera.com

obtained were compared to SCOP classification in order to validate the measure. Extensive computational experiments showed that alignments which are at most off by 10% from the optimal value can be computed in a short time. Further experiments showed how this measure reacts to the choice of the threshold defining a contact, and how to choose this threshold in a sensitive way.

1 Introduction

The comparison of protein structures is a problem of paramount importance in structural genomics, and an increasing number of approaches for its solution were proposed over the past years. Several protein structure classification servers based on these methods were designed (e.g., SCOP [37], DALI [23], CATH [41], FSSP [25]), and are extensively used in practice. This area of research continues to be very active, being energized bi-annually by the CASP folding competitions [35, 36], but despite the extraordinary international research effort, progress is slow. A fundamental dimension of this bottleneck is the absence of rigorous algorithmic methods. A recent excellent survey on structure comparison [10] 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...*

This situation is due to several reasons, among which:

1. By their nature, three-dimensional computational problems are inherently more complex than the similar one-dimensional ones for which we have 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 continuous, geometric and combinatorial, discrete mathematics.

2. Various simplified versions of the problems were shown NP-hard, e.g., for structure comparison see [16, 17].

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 a 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 [12].

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* [27], *distance map similarity* [24] and *Contact Map Overlap* (CMO) [13]. 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.

CMO is based on the basic notion of *contact* between two residues, a 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. These 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, and detection of 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 [14, 13]. The work recorded here is part of our research program started in 1999 on 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 [17, 16, 33].

In this article we describe our work on the CMO problem which culminated in two papers, presented at the ACM Conference on Computational Molecular Biology, in 2001 [30] and 2002 [6]. We have developed an *Integer Linear Programming* (ILP) formulation of the protein structure CMO, and studied two approaches for its solution, i.e., *Branch-and-Cut* (B&C) and *Lagrangian Relaxation* (LR). This is the first effective use of ILP methods in protein structure comparison; the biomolecular structure literature contains only one other effective B&C method devoted to RNA comparison, due to [31]. Furthermore, to the best of our knowledge, LR techniques have never been used before neither for structure nor for sequence alignments. We remark

that LR techniques have proved very powerful for the solution of very large Combinatorial Optimization problems [40, 11].

We tested our algorithm on real proteins from Protein Data Bank (PDB) [4]. A comparison of B&C to LR shows how the latter outperforms the former, being capable of computing upper bounds of similar quality within computing times that are orders of magnitude smaller. By using the LR approach, we were able to solve optimally for the first time alignment problems for proteins with about 1000 residues and 2000 contacts. Furthermore, we were able to compare all 780 pairs in a testbed of 40 large proteins, suggested by Jeffrey Skolnick, within few hours. In 150 cases, our method found the optimal solution. The same proteins were then used in a clustering experiment aimed at determining the best distance-threshold to define a contact. From our results, it appears that by using a threshold smaller than 7Å the clusters in this data set cannot be retrieved with sufficient precision. Finally, we considered the 269 proteins from the literature [30] and compared within a few days, all the (about) 36000 corresponding pairs.

On the negative side, it has to be remarked that the optimal solution of instances associated with substantially different proteins seems completely out of reach not only for our algorithm, but also with the other methods method currently known in Combinatorial Optimization. (On the other hand, finding a provably optimal solution for completely different proteins appears to be of no practical interest.) Such a situation is analogous to the case of the Quadratic Assignment Problem, for which instances with 40 nodes (the "counterpart" of residues) are quite far from being solved to optimality. The Quadratic Assignment Problem [7] bears many similarities with the structure alignment problem. In particular, there are profits p_{ij} in the objective function which are attained when two binary variables x_i and x_j are *both* set to 1 in a solution.

As it will be shown in the sequel, 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 classical problem in Combinatorial Optimization, with a large literature. Despite its simple definition, this problem is one of the toughest to solve exactly. Many papers over the years have dealt with the exact solution of MIS or, equivalently, Maximum Clique [39, 3, 26], 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 [26]. Our work shows that we can solve the MIS on instances with 10,000 nodes and larger (i.e., proportional to the product of the number of contacts of the two maps), on the graphs derived from the CMO problem. Clearly, this approach is not general but exploits the particular characteristics of the problem at hand.

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.

The remainder of the paper is organized as follows. In Section 2 we describe contact maps and two possible strategies for optimizing their overlap, i.e., B&C and LR. In Section 3 we state an Integer Quadratic Programming formulation of the problem and show how to linearize and strengthen it. Section 4 is devoted to describing the B&C approach, with a characterization of the clique inequalities, and a description of a polynomial algorithm for their separation. Section 5 is devoted to describing the LR-based approach. We show how to decompose the problem and reduce it to a sequence of standard alignment problems, solved by Dynamic Programming. In Section 6 we describe the heuristic procedures designed for finding good feasible solutions, namely Steepest Ascent Local Search and Genetic Algorithms. Finally, in Section 7 we report the results of our computational experiments.

2 Contact Maps

[Figure 1 about here.]

A *contact map* [20] of a protein of n residues is a 0-1, symmetric $n \times n$ matrix C, whose 1–elements correspond to pairs of amino acids in three–dimensional "contact". A contact can be defined in many ways. Typically [34], one considers $C_{ij} = 1$ when the distance of two heavy atoms, one from the i–th amino acid and one from the j–th amino acid of the protein, is smaller than a given threshold (e.g., 5Å) in the protein's

native fold. The intuitive and simple contact map representation of proteins is already complex enough to capture the most important properties of the folding phenomenon. It was shown that it is relatively easy to go from a map to a set of possible structures to which it may correspond [20, 44]. This result opened the possibility of using contact maps to predict protein structure from sequence, by predicting contact maps from sequence instead.

Besides their use for protein fold prediction, contact maps are exploited to compare 3D structures. The basic idea is that, when two structures are similar, one should expect their contact maps to be similar as well. Hence, one can use an indirect method for structure comparison, i.e., contact map comparison.

We can regard the contact map of a protein p as the adjacency matrix of a graph G_p. Each residue is a node of G_p, and there is an edge between two nodes if the the corresponding residues are in contact. The CMO problem calls for determining an ordered (i.e., *noncrossing*) alignment of some residues in the first protein (nodes in G_1) and the second protein (nodes in G_2). The alignment specifies the residues that are considered equivalent in the two proteins. The goal is to find the alignment which highlights the largest set of common contacts, where the value of an alignment is given by the number of contacts (edges) in the first protein whose endpoints (residues) are aligned with residues that are also in contact in the second protein. This value is called the *overlap* for the two contact maps, and the optimization problem is to find the maximum overlap. Figure 1 shows two contact maps and an alignment.

The CMO problem tries to evaluate the similarity in the 3D folds of two proteins by determining the similarity of their contact maps (a high contact map similarity is a good indicator of high 3D similarity). This measure was introduced in [14], and its optimization was proved NP-hard in [17], thus justifying the use of sophisticated heuristics or enumerative methods.

2.1 CMO Optimization and Integer Linear Programming

Some of the most powerful algorithms for finding exact solutions of combinatorial optimization problems are based on ILP, which has been applied profitably in very many cases [40, 9]. The ILP approach consists in formulating a problem as the maximization of a linear function of some integer variables subject to linear inequalities and then solving it via Branch–and–Bound. When the gap between the upper bound and the

optimum is small, the pruning of the search space is effective. The upper bound can be mainly derived in two ways, Linear Programming Relaxation and Lagrangian Relaxation.

2.1.1 Linear Programming Relaxation

In the *Linear Programming* (LP) relaxation, the bound is obtained by relaxing the integrality constraint on the variables, which are allowed to take fractional values. The resulting LP is solvable in polynomial time [28, 29]. The success of this approach is witnessed by an enormous amount of applications, and there is a whole branch of applied mathematics, known as *Polyhedral Theory*, completely devoted to this topic. The key step in effectively solving a difficult problem by ILP lies in its formulation, i.e., in the choice of the variables and constraints. In order to obtain tight formulations (i.e., such that the gap between the LP optimum and the true optimum is small) it is often useful to introduce additional constraints, called *cuts*. 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 B&C approach is essentially a Branch-and-Bound in which cut constraints are added at each node of the search tree.

Oftentimes, there is an exponential number of possible inequalities, either in the basic formulation or in a family of cuts. For instance, in the formulations that we propose for CMO there is an exponential number of so-called *clique inequalities*. These are constraints saying that, out of a number of mutually incompatible ways of aligning two residues, at most one can be chosen in a solution. An exponential number of constraints must be dealt with only implicitly. The standard strategy for doing so is via a *Separation Algorithm*, a procedure which, given a fractional solution, finds a violated inequality, if one exists. If the inequality is found, it becomes a cut to add to the LP formulation. By a fundamental result of [19], a polynomial time separation algorithm leads to a polynomial time algorithm to solve an exponential-size LP relaxation. For CMO, separation of clique inequalities can be done in time $O(n_1 n_2)$ (where n_1 and n_2 are the number of residues in the two proteins).

2.1.2 Lagrangian Relaxation

The LR approach is particularly well suited for those cases in which the formulation of a problem consists of two sets of constraints: a set of "nice" constraints and a set of "bad" constraints, whose removal makes

the resulting problem, called the Lagrangian relaxed problem, easily solvable. The strategy then consists in removing the bad constraints from the formulation and putting them in the objective function, each weighed by some coefficient (*Lagrangian multiplier*). The weight for a constraint represents a penalty incurred by a solution which does not satisfy that constraint. To any choice of weights there corresponds a (relatively easy) problem whose solution yields an upper bound to the original problem. In the CMO case, this problem reduces to $O(n_1 n_2)$ standard sequence alignment problems, each solvable in $O(n_1\, n_2)$ time by Dynamic Programming. The core question of LR is then to determine the optimal weights, i.e., the Lagrangian multipliers yielding the best upper bound. In most cases, the determination of these multipliers is equivalent to solving a suitable LP, which would be too time consuming in practice. On the other hand, near-optimal multipliers can be found by a simple iterative procedure known as *subgradient optimization*, in which, at each iteration, the Lagrangian relaxed problem is solved and the multipliers are updated based on the corresponding solution.

Besides yielding an upper bound on the optimal solution of the original problem, the Lagrangian multipliers can be used to drive simple heuristic procedures (in most cases of greedy nature). These procedures typically produce substantially different solutions for different Lagrangian multipliers. Accordingly, if they are embedded within an iterative procedure to define near-optimal multipliers, namely they are called at each iteration with the current multipliers, the best solution found over all iterations tends to be near-optimal.

The theory of LR is a well established branch of Combinatorial Optimization, and has been used successfully in a large number of applications, in different domains [40]. Nowadays, LR is the most successful tool to tackle very large problems. For instance, the state of the art algorithms for the well known Set Covering Problem, one of the Combinatorial Optimization problems most frequently solved in real-world applications, are based on LR [5]. These algorithms are capable of finding near-optimal solutions to instances with millions of variables and thousands of constraints within minutes on a PC. However, to the best of our knowledge, this is the first time that a similar approach is used for an alignment problem arising in Computational Molecular Biology.

3 The Mathematical Formulation

As described in Section 2, we can use a graph to represent a protein (where each edge is a contact and each node is a residue) and so we will rephrase the CMO problem in graph–theoretic language. We are given two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, with $V_i = \{1, \ldots, n_i\}$ for $i = 1, 2$. We 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). For i in V_k, we denote by $\delta^+(i)$ the set $\{j \in i+1, \ldots, n_k : (i, j) \in E_k\}$, by $\delta^-(i)$ the set $\{j \in 1, \ldots, i-1 : (j, i) \in E_k\}$, and we let $N_k(i) := \delta^-(i) \cup \delta^+(i)$ be the set of neighbors of i in G_k.

To avoid confusion with the *edges* in G_1 and G_2, we hereafter call a pair $\{i, j\}$ with $i \in V_1$ and $j \in V_2$ a *line* (since it *aligns* i with j), and we denote it by $[i, j]$. Let $L = V_1 \times V_2$ be the set of all lines. An *alignment* of V_1 and V_2 is defined by a subset of lines $\{[i_1, j_1], \ldots, [i_p, j_p]\} \subset V_1 \times V_2$. An alignment is feasible if, for $q \neq r$, either $i_q < i_r$ and $j_q < j_r$ or $i_r < i_q$ and $j_r < j_q$, i.e., a feasible alignment corresponds to a *noncrossing matching* in the complete bipartite graph $(V_1 \cup V_2, L)$. Two edges $(h_1, l_1) \in E_1$ and $(h_2, l_2) \in E_2$ are *shared* by an alignment if there are $q, r \leq k$ such that $h_1 = i_q$, $l_1 = i_r$, $h_2 = j_q$ and $l_2 = j_r$. Each pair of shared edges contributes a *sharing* to the objective function. The problem consists of finding a feasible alignment which maximizes the number of sharings. It is easy to see that CMO is NP-hard since, if G_1 is a complete graph on $k = n_1$ vertices, the problem has a solution of value k if and only if G_2 contains a *Clique* of size k. More precisely, the problem call for the *Densest Subgraph* of size k of G_2 in this case. NP-hardness for special classes of graphs, closer to contact maps coming from real-life instances, is proved in [17].

We say that two lines $l = [i_1, j_1]$ and $m = [i_2, j_2]$ in L are *compatible* if there exists a feasible alignment that contains both lines (and *incompatible* otherwise). We say that incompatible lines *cross*, and *strictly* cross if they have no endpoint in common. We call $\mathcal{I}$ the (exponentially large) collection of all maximal sets of pairwise incompatible lines $I \subset L$.

3.1 A Quadratic Model

For $l = [i, j] \in L$ we introduce a binary variable, which we denote either as x_l or x_{ij}, equal to 1 if and only if i is aligned with j. We let X be the set of incidence vectors $x \in \{0, 1\}^L$ of all noncrossing matchings in L. For

$l = [i_1, i_2], m = [j_1, j_2] \in L$, let $a_{lm} = 1$ if $(i_1, j_1) \in E_1$ and $(i_2, j_2) \in E_2$, $a_{lm} = 0$ otherwise. The objective function in our first formulation contains terms of the form $a_{lm}x_l x_m$, indicating that a contribute of a sharing is achieved if the three terms in the product take the value 1. In fact, in order to illustrate our method, it is necessary to introduce separately products $x_l x_m$ and $x_m x_l$ in the objective function. To this end, we define separate profits b_{lm} for $x_l x_m$ and b_{ml} for $x_m x_l$. Of course, we have to ensure $b_{lm} + b_{ml} = a_{lm} (= a_{ml})$ for all $l, m \in L$. For instance, a valid (initial) choice is $b_{lm} = b_{ml} = a_{lm}/2$. Later, it will be clear how crucial is for our method to split a_{lm} into b_{lm} and b_{ml} "optimally", even if we will not fix this splitting *a priori*, but rather use an iterative method to find it.

The problem can now be stated as

$$\max \sum_{l \in L} \sum_{m \in L} b_{lm} x_l x_m \tag{1}$$

subject to

$$x \in X. \tag{2}$$

Actually, we know how to represent X with linear constraints. Recalling the definition of $\mathcal{I}$, (2) is equivalent to

$$\sum_{l \in I} x_l \leq 1, \ \forall I \in \mathcal{I} \tag{3}$$

$$x_l \geq 0, \text{ integer}, \ \forall l \in L. \tag{4}$$

Problem (1), (3), (4) is a *Binary Quadratic Program*. Note in passing that, although the *convex hull* of the vectors in X is given by (3) and the non-negativity constraints, as illustrated in Section 4, and therefore the integrality conditions could be removed if (1) were linear (see, e.g., [40]), in this case we must keep these conditions (it is false in general that the maximization of a quadratic function over a polytope has a solution which is a vertex of the polytope). The formulation shows that the problem is closely related to the Quadratic Assignment Problem. Here, the difference is that the problem is in maximization form, the matching to be found does not have to be perfect and it must be noncrossing (see, e.g., [8]).

3.2 Linearization and strengthening

To linearize (1) we use a standard technique, introducing variables y_{lm}, for $l, m \in L$, and replacing the product $x_l x_m$ by y_{lm}. Note that we introduce separately y_{lm} and y_{ml}. (In practice, the number of y

variables that we will have to introduce explicitly is much smaller, namely $2|E_1||E_2|$, as it can be assumed without loss of generality that the remaining variables take the value 0. However, for the sake of illustration we will describe the model as if all the y variables were present.) The new (linear) objective function becomes

$$\max \sum_{l \in L} \sum_{m \in L} b_{lm} y_{lm}.$$

Moreover, the nonlinear relation $y_{lm} = x_l x_m$ can be replaced by the linear constraints

$$y_{lm} \leq x_m, \ \forall l, m \in L \tag{5}$$

$$y_{lm} = y_{ml}, \ \forall l, m \in L, l < m, \tag{6}$$

where "$<$" denotes an (arbitrarily defined) total ordering of the lines in order to avoid repeating the same constraint twice. In particular, note that no condition is required to enforce y_{lm} (and y_{ml}) to 1 if both x_l and x_m are 1, since the maximization will guarantee this condition (actually, if $a_{lm} = 0$, $x_l = x_m = 1$ and $y_{lm} = y_{ml} = 0$, setting $y_{lm} = y_{ml} = 1$ yields a feasible solution of the same value). In fact, we do not use (5) but stronger linear conditions. For this purpose, we adopt a standard procedure used in the convexification of ILPs [1, 32, 2].

If we multiply a constraint (3) associated with set I by x_m for some $m \in L$ and replace $x_l x_m$ by y_{lm}, we get

$$\sum_{l \in I} y_{lm} \leq x_m.$$

Note that, if $m \in I$, we can write x_m instead of y_{mm}, since the variables are restricted to be binary, getting $\sum_{l \in I \setminus \{m\}} y_{lm} \leq 0$, i.e., the condition that all y_{ml} must be 0 if l and m are incompatible. By doing the above for all constraints in (3) and variables x_m, $m \in L$, we get our new model:

$$\max \sum_{l \in L} \sum_{m \in L} b_{lm} y_{lm} \tag{7}$$

subject to

$$\sum_{l \in I} x_l \leq 1, \ \forall I \in \mathcal{I} \tag{8}$$

$$\sum_{l \in I} y_{lm} \leq x_m, \ \forall I \in \mathcal{I}, \ m \in L \tag{9}$$

$$y_{lm} = y_{ml}, \ \forall l, m \in L, l < m \tag{10}$$

$$x_l, y_{lm} \geq 0, \text{integer}, \ \forall l, m \in L. \tag{11}$$

Note that relations (5) are not required, as their are implied by (9). An interesting property of formulation (7)-(11) is that the problem obtained by removing constraints (10) is as difficult as the maximization of a *linear* function subject to (2) (or, equivalently, (3) and (4)). This property is common to other integer quadratic programs, such as the Quadratic Assignment Problem [7].

[Figure 2 about here.]

4 The Branch-and-Cut Approach

In this section we describe the B&C approach to the solution of CMO, which originally appeared in [30].

The formulation of the problem is given by (7)-(11), but with the constraints (9) replaced by

$$\sum_{l \in I} y_{lm} \leq x_m, \ \forall I \in \mathcal{I}', \ m \in L \tag{12}$$

where $\mathcal{I}' \subseteq \mathcal{I}$ is the collection of sets given by $\{\{i\} \times V_2 \ | i \in V_1\} \cup \{V_1 \times \{i\} \,|\, i \in V_2\}$. Moreover, we only have variables y_{lm} for pairs (l, m) of lines such that $a_{lm} = 1$, and identify variables y_{lm} and y_{ml} removing constraints (10). A similar formulation was adopted by [31] for the solution of the RNA sequence-structure alignment problem.

Since the number of inequalities (12) is $O(n_1 n_2 \max\{n_1, n_2\})$, we have them all explicitly in the LP. As to inequalities (8), their number is $\Theta(2^{n_1+n_2})$ ([30, 31]) and we have to resort to a separation algorithm. Both [30] and [31] present Dynamic Programming based $O(n_1 n_2)$ algorithms for separation of these inequalities. The algorithm of [31], which is simpler to describe, is reported in Section 4.1. In Section 4.2 we illustrate a formulation of the CMO problem as a MIS problem. This formulation suggests some valid inequalities, which we used in our B&C approach. The overall B&C algorithm is briefly outlined in Section 4.3.

4.1 Separation of clique inequalities

Recall that X is the set of incidence vectors of all noncrossing matchings in L (see Figure 3(a)). A noncrossing matching in L corresponds to a stable set in a new graph, G_L (the *conflict graph* of lines), defined as follows.

Each line $l \in L$ is a vertex of G_L, and two vertices l and h are connected by an edge if the lines l and h cross.

We now show that the graph G_L is *perfect*, by proving that in fact it belongs to a special class of perfect graphs, i.e., its complement $\overline{G}_L$ is a comparability graph. A *comparability graph* is an unoriented graph for which there exist a way to orient the edges which yields a transitive and acyclic digraph. I.e., there are no directed cycles and, if (i,j) and (j,k) are arcs of the digraph, also (i,k) is an arc of the digraph. For a definition of perfectness and a proof that comparability graphs are perfect, see [18].

Theorem 1 *The graph $\overline{G}_L$ is a comparability graph.*

Proof Two lines l and m are connected by an edge in $\overline{G}_L$ if and only if they are compatible. Let $\prec$ be the partial ordering on L defined by $[a,b] \prec [c,d]$ if $a < c$ and $b < d$. Note that for compatible lines l and m, either $l \prec m$ or $m \prec l$. Orient each edge $\{l,m\}$ in $\overline{G}_L$ such that $l \prec m$ from l to m, obtaining a set of arcs A. Then, if $(l,m) \in A$ and $(m,q) \in A$, it follows that l and q are compatible, and $l \prec q$, so that $(l,q) \in A$. Hence, $\overline{G}_L$ is a comparability graph. $\diamond$

The perfectness implies that a complete characterization of the convex hull of X in terms of linear inequalities is given by non-negativity inequalities and (8). Note that each set $I \in \mathcal{I}$ corresponds to a maximal clique in G_L.

Given weights x_l^* for each line l, the separation problem for (8) consists in finding a clique I in G_L such that $\sum_{l \in I} x_l^* > 1$. Since G_L is perfect, this problem can be solved in polynomial time by finding the maximum x^*-weight clique in G_L. Instead of resorting to general algorithms for comparability graphs, based on network flow techniques, the separation problem can be solved directly by Dynamic Programming, as described in [31] and, independently and with a different construction, in [30]. Both algorithms have complexity $O(n_1 n_2)$. We describe here the construction of [31], which appears to be simpler.

[Figure 3 about here.]

Consider L as the vertex set of a directed grid, in which vertex $[1, n_1]$ is put in the lower left corner and vertex $[n_2, 1]$ in the upper right corner (see Figure 3(b)). At each internal node $l = [i,j]$, there are two incoming arcs, one directed from the node below, i.e. $[i, j+1]$, and one directed from the node on the left, i.e. $[i-1, j]$. Each such arc has associated a length equal to x_l^*.

Theorem 2 *([31, 30]) The nodes on a directed path P from $[1, n_1]$ to $[n_2, 1]$ in the grid correspond to a maximal clique I in G_L and vice versa.*

Then the most violated clique inequality can be found by taking the longest $[1, n_1]$-$[n_2, 1]$ path in the grid. There is a violated clique inequality if and only if the length of the path (plus $x^*_{1n_1}$) is greater than 1.

Closely related to the problem of finding a largest-weight clique in G_L is the problem of finding a largest-weight stable set in G_L (i.e., a maximum weight noncrossing matching in L). Although not needed in the B&C approach, this problem is instrumental to the LR approach, described in Section 5. Also for this problem there is a Dynamic Programming solution, of complexity $O(n_1 n_2)$, discussed in detail in Section 5.6. Incidentally, this algorithm coincides with the basic algorithm for computing the *edit distance* of two strings, rediscovered independently by many authors (see, e.g., [38, 43]).

4.2 A connection to MIS

Two variables y_{lm} and $y_{l'm'}$ can be both 1 in a noncrossing map if and only if no two of the lines in $\{l, m, l', m'\}$ cross. Let $M = \{(l, m) \in L \times L : l < m,\, a_{lm} = 1\}$ be the set of line pairs associated with variables in the LP. We define a graph $G_M = (M, E_M)$ by putting an edge between the node corresponding to two variables y_{lm} and $y_{l'm'}$ if they cannot be both 1 in a noncrossing matching. Then the CMO problem is nothing other than the MIS problem in G_M.

Although it would be possible to formulate and solve a standard ILP corresponding to this MIS problem, i.e.,

$$\max \sum_{(l,m)\in M} y_{lm} \tag{13}$$

$$y_{lm} + y_{l'm'} \leq 1,\ \forall\{(l, m), (l', m')\} \in E_M \tag{14}$$

$$y_{lm} \geq 0,\ \text{integer}\ \ \forall (l, m) \in M, \tag{15}$$

the resulting LP bound would be very weak unless we strengthen it with cuts, which are often hard to separate. This formulation would also have too many constraints, i.e., $\Theta(|E_1|^2|E_2|^2)$.

Since this MIS formulation uses the same variables as the original CMO formulation that we outlined ((7)-(11)), all valid inequalities for the MIS formulation can be used as cuts for the CMO. The most notable classes of inequalities are derived from odd-holes and cliques in G_M.

An *odd hole* is an odd cycle with no chords. By Theorem 1, G_L has no odd holes, while G_M may contain them. The *odd-hole inequalities* for the MIS problem say that for any odd-hole H there can be at most $\lfloor |H|/2 \rfloor$ nodes in an independent set, i.e.,

$$\sum_{(l,m)\in H} y_{lm} \leq \lfloor |H|/2 \rfloor, \ \forall H \in \mathcal{H}, \tag{16}$$

where $\mathcal{H}$ denotes the (exponentially large) collection of the odd holes in G_M. There is a known polynomial time algorithm for separating odd-holes ([40]), whose time complexity is $O(|E_1|^3|E_2|^3)$, that is in practice much better than in the worst case since only the nodes corresponding to strictly positive y variables can be considered in the separation. This algorithm was tried in our code. However, the improvement in the upper bound due to the addition of these constraints is negligible.

The *clique inequalities* say that the sum of y variables for a clique in G_M cannot be greater than 1, i.e.,

$$\sum_{(l,m)\in C} y_{lm} \leq 1, \ \forall C \in \mathcal{C}, \tag{17}$$

where $\mathcal{C}$ denotes the (exponentially large) collection of maximal cliques in G_M. The solution of the LP relaxation obtained by adding these inequalities seems at present completely out of reach even for very small size instances. Accordingly, we next describe a specific class of clique inequalities for our case, that we tried in our algorithm since they can be separated within reasonable time. Consider two edges both in the same graph, say G_1 (the same conclusions and inequalities will apply to G_2 as well). They can either have no endpoint in common and not intersect (cases A1 and A2 in Figure 4), or have one common endpoint (cases B1, B2, B3) or no endpoint in common and intersect (case C). For an edge $e_1 \in E_1$ and R one of A1, A2, A3, B1, B2, C, call $R(e_1)$ the set of edges which are in the relation R with e_1. In any feasible solution in which e_1 and $e_2 \in E_2$ are shared and $f_1 \in E_1, f_2 \in E_2$ are shared, f_2 must be in the same relationship to e_2 as f_1 is to e_1. Therefore, considering edges $e_1 = (i_1, j_1) \in E_1$, $f_1 = (h_1, l_1) \in E_1$ such that $f_1 \in R(e_1)$, and $e_2 = (i_2, j_2) \in E_2$, the following is a valid clique inequality:

$$y_{[i_1,i_2][j_1,j_2]} + \sum_{(h_2,l_2)\in E_2 \setminus R(e_2)} y_{[h_1,h_2][l_l,l_2]} \leq 1, \ \forall e_1, f_1 \in E_1, f_1 \in R(e_1), e_2 \in E_2. \tag{18}$$

As already mentioned, the same inequalities hold if the roles of G_1 and G_2 are exchanged. It can be easily proved that the clique inequalities relative to the cases B1, B2, B3 and C are implied by constraints (8)

and (9), and hence cannot be used as cuts in our formulation. The remaining inequalities however are not implied. Consider for instance the case $e_1 = (2,3) \in E_1$ and $e_2 = (2,5) \in E_2$. The edge $f_1 = (1,4) \in E_1$ is in relation A2 with e_1. However, $f_2 = (1,4) \in E_2$ and $g_2 = (3,6) \in E_2$, are not in the relation A2 with e_2. The solution $y_{[2,2][3,5]} = y_{[1,1][4,4]} = y_{[1,3][4,6]} = 1/2$ and $x_{11} = x_{13} = x_{22} = x_{35} = x_{44} = x_{46} = 1/2$ is feasible for constraints (8) and (9), but violates a constraint (18). It is possible to describe a similar example for the case R=A1. Inequalities (18) can be separated by simply cycling over all possible e_1, f_1, e_2, considering then each edge $f_2 \in E_2$ and verifying if $f_2 \in R(e_2)$. The corresponding running time is $O(|E_1|^2|E_2|^2)$.

[Figure 4 about here.]

Our computational experiments have shown that these clique inequalities in the y variables are actually very weak, and very seldom does their use give an improvement to the bound value. For instance, in the above example, adding the clique inequality does not change the objective function value (3/2) but simply the solution, which becomes $x_{1i} = x_{2i} = x_{3j} = x_{4j} = 1/4$ for $i = 1,2,3$ and $j = 4,5,6$, and $y_{[2,1][3,4]} = y_{[2,3][3,6]} = y_{[2,2][3,5]} = y_{[1,1][4,4]} = y_{[1,3][4,6]} = y_{[1,2][4,5]} = 1/4$. This solution is now feasible for all cliques in both x and y variables.

4.3 The overall B&C algorithm

In the following we give the basic features of our overall B&C algorithm, referring to [40] for details about the general structures of these algorithms.

The initial LP contains all variables and, besides non-negativity inequalities, constraints (12), recalling that (10) are implicitly imposed. Constraints (8) are separated as described above, whereas, as already discussed, constraints (16) and (18) are not particularly useful and hence not used in the final version. Branching is performed by choosing the most fractional x variable and generating two subproblems by fixing it to 1 and 0, respectively. At each B&C node, we use a greedy heuristic that considers the x variables according to decreasing LP values and, for each variable x_l it sets it to 1 with probability equal to the associated value. If x_l is set to one, all the incompatible variables (corresponding to lines crossing l) are set to 0. The final solution obtained is then used as starting point for the local search procedures illustrated in Section 6. The B&C nodes are considered according to a *best first* policy.

5 The Lagrangian Relaxation approach

In this section we show how the Lagrangian relaxation of constraints (10) leads to a problem that is efficiently solvable, yielding upper bounds that are generally better than those found by the B&C method. In Section 5.1 we show how to solve the problem defined by (7), (8), (9), and (11). We then address in Section 5.2 the Lagrangian relaxation of constraints (10), as opposed to their removal, discussing how to find near-optimal Lagrangian multipliers in Section 5.3. Sections 5.4 and 5.5 describe a heuristic and an exact enumerative algorithm based on LR, whereas Section 5.6 shows how these algorithms can be implemented efficiently.

5.1 Eliminating constraints (10)

Theorem 3 *The problem defined by (7), (8), (9), and (11) can be solved in $O(|E_1||E_2|)$ time.*

Proof After the removal of equations (10), each variable y_{lm} appears only in the constraints (9) associated with x_m (besides being constrained to be nonnegative and integer by (11)). For each $m \in L$, this implies that, if variable x_m takes the value 0 all variables y_{lm} take the same value, whereas, if variable x_m takes the value 1, the optimal choice of y_{lm} for $l \in L$ amounts to solving the following:

$$\max \sum_{l \in L} b_{lm} y_{lm} \tag{19}$$

subject to

$$\sum_{l \in I} y_{lm} \leq 1, \ \forall I \in \mathcal{I} \text{ such that } m \notin I \tag{20}$$

$$\sum_{l \in I} y_{lm} \leq 0, \ \forall I \in \mathcal{I} \text{ such that } m \in I \tag{21}$$

$$y_{lm} \geq 0, \text{ integer}, \ \forall l \in L. \tag{22}$$

In other words, the profit achieved if $x_m = 1$, say p_m, is given by the optimal solution of (19)–(22). This is a simple *alignment problem* that can be solved by Dynamic Programming, as explained in Section 4.1. Once profits p_m have been computed for all $m \in L$, we can find the optimal solution to our problem (without (10)) by solving

$$\max \sum_{m \in L} p_m x_m \tag{23}$$

subject to

$$\sum_{l \in I} x_m \leq 1, \ \forall I \in \mathcal{I} \tag{24}$$

$$x_m \geq 0, \text{ integer}, \ \forall m \in L, \tag{25}$$

which is again an alignment problem.

Overall, an optimal solution $(\overline{x}, \overline{y})$ of the relaxed problem is obtained by:

(i) For each $m \in L$, computing an optimal solution $\hat{y}_{lm}$ $(l \in L)$ to problem (19)–(22), letting p_m be the associated profit;

(ii) Computing an optimal solution $\overline{x}$ to problem (23)–(25);

(iii) Letting $\overline{y}_{lm} := \hat{y}_{lm} \cdot \overline{x}_m$ $(l, m \in L)$.

In particular, note that variable y_{lm} takes the value 1 in the solution of the overall problem if and only if it takes the value 1 in the solution of (19)–(22) and x_m takes the value 1 in the solution of (23)–(25).

We conclude the proof by analyzing the running time of the method. Of course, we can explicitly consider y_{lm} with $l = [i_1, j_1]$ and $m = [i_2, j_2]$ only if l and m are compatible, $(i_1, i_2) \in E_1$ and $(j_1, j_2) \in E_2$, since the other y_{lm}'s can be assumed to be 0 without loss of generality. Accordingly, for each $m = [i, j] \in L$ we can solve (19)–(22) by simply considering the subgraphs of G_1 and G_2 induced, respectively, by $N_1(i)$ and $N_2(j)$ – the sets of neighbors of i and j. For these subgraphs, we have to find the maximum-weight (with respect to weights b) matching, without taking any line incompatible with m.

For each $m = [i, j] \in L$ we should find (separately) the maximum-weight noncrossing matching among the "left" neighbors of i and j as well as among the "right" neighbors of i and j. This can be done in time $O(|N_1(i)||N_2(j)|)$ by Dynamic Programming, as explained in detail in in Section 5.6. The solution of (23)–(25) can be derived in time $O(n_1 n_2)$ again by Dynamic Programming, leading to an overall complexity of

$$O(n_1 n_2 + \sum_{i \in V_1} \sum_{j \in V_2} |N_1(i)||N_2(j)|) = O(|E_1||E_2|).$$

$\diamond$

5.2 Lagrangian relaxation

Although the quality of the upper bound that we obtain by solving the relaxation without constraints (10) is typically quite poor, we can get much better bounds with a perfectly identical approach by relaxing these constraints in a *Lagrangian* way. This amounts to assigning a *Lagrangian multiplier* λ_{lm} to each constraint (10) and adding to the original objective function (7) a linear combination of constraints (10), each weighed by the associated Lagrangian multiplier, obtaining the Lagrangian objective function

$$\max \sum_{l \in L} \sum_{m \in L} b_{lm} y_{lm} + \sum_{l \in L} \sum_{\substack{m \in L: \\ l<m}} \lambda_{lm}(y_{lm} - y_{ml}). \tag{26}$$

The corresponding Lagrangian relaxed problem requires the maximization of (26) subject to (8), (9), and (11). The resulting value is an upper bound on the optimal value of (7)–(11) since, for each feasible solution of the latter, the contribution to (26) of the new term is null.

Defining for convenience $\lambda_{ml} := -\lambda_{lm}$ for $l < m$ (and $\lambda_{mm} := 0$), (26) can be rewritten as

$$\max \sum_{l \in L} \sum_{m \in L} (b_{lm} + \lambda_{lm}) y_{lm}. \tag{27}$$

Then, one can see immediately that the effect of Lagrangian relaxation is to re–distribute the profit a_{lm} between the two terms in the objective function associated with y_{lm} and y_{ml}. Clearly, the Lagrangian relaxed problem can be solved as before, after replacing b_{lm} by $b_{lm} + \lambda_{lm}$ for $l, m \in L$. Let $U(\lambda)$ be the resulting upper bound.

The best upper bound that can be obtained by Lagrangian relaxation is $U(\lambda^*) := \min_\lambda U(\lambda)$, where λ^* denotes the best Lagrangian multiplier vector. By the above discussion, finding λ^* is the same as splitting profits a_{lm} between b_{lm} and b_{ml} so that the optimal value of the relaxation considered in Theorem 3 is minimized.

It is interesting to compare upper bound $U(\lambda^*)$ to the upper bounds obtainable by the B&C approach. The first, denoted by U_1 is the one that is actually computed by the final version of the B&C algorithm, and is associated with LP relaxation (7), (8), (10), (12), and the non-negativity constraints. The second, denoted by U_2, is the value of the LP relaxation of (13)–(15) with the addition of (17). Recall that this LP relaxation is (at present) apparently impossible to solve for realistic-size instances.

Theorem 4 $U_1 \geq U(\lambda^*) \geq U_2$.

Proof It is easy to observe that the optimal solution of the relaxed problem (26), (8), (9), and (11) does not change if the integrality constraints are removed. In this case, it is known [11] that $U(\lambda^*)$ coincides with the optimal solution value of the LP relaxation defined by (7)–(9) plus the non-negativity conditions. This bound is stronger than U_1 since (12) are a subset of (9). This shows $U_1 \geq U(\lambda^*)$.

To show that $U(\lambda^*) \geq U_2$, we prove that every feasible solution of the LP relaxation yielding U_2 is also feasible for the LP relaxation yielding $U(\lambda^*)$. Indeed, consider a nonnegative $\overline{y}$ satisfying (17), and define

$$\overline{x}_m := \max_{\substack{I \in \mathcal{I}: \\ m \notin I}} \sum_{l \in I} \overline{y}_{lm} \ (m \in L).$$

Clearly, all constraints (9) are satisfied by $(\overline{x}, \overline{y})$. What remains to be shown is that all constraints (8) are satisfied by $\overline{x}$. For a generic $I \in \mathcal{I}$, we have

$$\sum_{l \in I} \overline{x}_l = \sum_{l \in I} \max_{\substack{Q \in \mathcal{I}: \\ l \notin Q}} \sum_{q \in Q} \overline{y}_{ql}.$$

To see that this cannot be more than 1, consider for each $l \in L$ an arbitrary clique $Q(l) \in \mathcal{I}$ with $l \notin Q(l)$. It is easy to check that the set defined by $\{(q, l) : l \in I, q \in Q(l)\}$ belongs to the collection $\mathcal{C}$ of cliques in G_M. Since $\overline{y}$ satisfies (17), the claim follows.

$\diamond$

5.3 Finding near–optimal multipliers

Our approach determines a near-optimal λ by a standard *Subgradient Optimization* procedure; see [21]. The procedure generates a series $\lambda^0, \lambda^1, \lambda^2, \ldots$ of multiplier vectors, where $\lambda^0 := 0$ (with $b_{lm} = b_{ml} = a_{lm}/2$ for $l, m \in L$), and, for $k \geq 0$, λ^{k+1} is defined from λ^k as follows. Let $(\overline{x}, \overline{y})$ denote an optimal solution of the Lagrangian relaxed problem associated with λ^k. The corresponding *subgradient vector* is given by

$$s_{lm} := \overline{y}_{lm} - \overline{y}_{ml}, \ l, m \in L, l < m.$$

Using the technique proposed in [21], we compute the new multipliers by

$$\lambda_{lm}^{k+1} := \begin{cases} \lambda_{lm}^k & \text{if} \quad s_{lm} = 0 \\ \max(\lambda_{lm}^k - \gamma, -b_{lm}) & \text{if} \quad s_{lm} = 1 \\ \min(\lambda_{lm}^k + \gamma, b_{ml}) & \text{if} \quad s_{lm} = -1 \end{cases}$$

for $l, m \in L, l < m$. Here, the *step size* γ is defined by

$$\gamma = \mu \frac{UB - LB}{\sum_{l,m} s_{l,m}^2}$$

where μ is a suitable parameter, while UB and LB are the values of the best upper bound and feasible solution found so far, respectively. In our implementation, μ is initially set to 1, and halved if the upper bound does not decrease within 50 iterations (halving μ is customary within subgradient optimization). The number of iterations is limited by $\max\{1000, 10 \cdot \max\{|E_1|, |E_2|\}\}$, since we experimentally observed that afterwards no substantial improvement occurs. The overall complexity of the upper bound computation is therefore $O(|E_1||E_2| \max\{|E_1|, |E_2|\})$.

A first heuristic procedure that we use to compute feasible solutions to the problem is very simple: we simply take the $\overline{x}$ vector corresponding to the Lagrangian relaxed solution, found at each iteration. The associated value is of course given by (1).

In the following we illustrate methods to find better solutions, namely a more sophisticated greedy heuristic, also applicable at every iteration of the procedure, and a branch–and–bound algorithm based on the Lagrangian relaxation. In both methods, we will *fix* some lines in the solution and solve the Lagrangian relaxed problem (26), (8), (9), (11) keeping into account this choice (and without changing the Lagrangian multipliers). We will let $U(S)$ denote the corresponding upper bound, where $S \subset L$ is the set of (pairwise compatible) lines fixed in the solution. We will also consider in S "pseudo" lines of the form $[i, \emptyset]$, meaning that we fixed that *no node* in V_2 will be aligned with node i in V_1 in the solution.

Since the computational bottleneck of both procedure is the determination of $U(S)$, we will also discuss how to perform this efficiently.

5.4 A Lagrangian greedy heuristic

A natural way to proceed in the construction of a solution is to choose a line which maximizes a suitable "score", fix it in the solution and iterate. As to the score, a natural decision would be to use the Lagrangian profit p_m for each line m. For our problem, solutions of much better quality are obtained if the score is computed in a more accurate way as follows: choose the line m such that $U(\{m\})$ is maximum, i.e., the line that, fixed in the solution, produces the smallest decrease in the value of the Lagrangian relaxed problem.

Another crucial choice is whether to consider the addition of any line (compatible of course with those previously fixed) to the solution, or to follow some order in the construction of the solution. For our instances, we found that constructing the solution "from left to right" (or, equivalently, "from right to left") is faster and produces solutions which, on average, are as good as those obtained without following any particular order.

Based on the two main issues above, our greedy heuristic proceeds like this: first of all, we decide whether node 1 in V_1 should be aligned with some node j in V_2 or not, computing $U(\{[1,j]\})$ for $j = 1, \ldots, n_2$ and for $j = \emptyset$. The choice corresponding to the best value is fixed, and the procedure is iterated with node 2 in V_1, and so on. If line $[i,j]$ is added to the solution S, we have that, for each node $l < j$ in V_2, either l is already incident with a line in S, or it will not be aligned in the solution. The corresponding pseudo–code implementation is given in Figure 5. There, S is the set of lines in the solution and f_2 is the leftmost node in V_2 which is not aligned and may still be aligned.

[Figure 5 about here.]

The greedy heuristic is called at each iteration of the subgradient optimization procedure.

5.5 A Branch–and–Bound Algorithm

The branch–and–bound algorithm is organized in a simple way. First of all, the current solution is constructed "from left to right" as in the greedy heuristic, fixing first the node $j \in V_2$ aligned with node $1 \in V_1$ (possibly 1 is not aligned), then the node $l \in V_2$ aligned with node $2 \in V_1$, and so on. After each choice the upper bound $U(S)$ corresponding to the lines already fixed in the solution is computed, and backtracking is performed if $\lfloor U(S) \rfloor$ is not larger than the best feasible solution found so far.

Actually, since the main purpose of branch–and–bound is to provide tighter upper bounds, letting $U(\emptyset)$ denote the upper bound when nothing is fixed (computed by the subgradient optimization procedure), we initially call the procedure with a *target solution value* $t := \lfloor U(\emptyset) \rfloor$, backtracking as soon as $\lfloor U(S) \rfloor \leq t$. If a solution of value t is found, it is clearly optimal, otherwise no such solution exists and we can call the procedure with $t := \lfloor U(\emptyset) \rfloor - 1$, and so on. Each time the procedure finds no solution of value t, we have a valid upper bound on the optimum which is one unit smaller.

In Figure 6 we report the pseudo–code recursive implementation of the procedure. The procedure is called initially as branch–and–bound$(1, 1, \emptyset)$, specifying the value of t. The status **success** means that a solution of value t was found. Moreover f_1 and f_2 denote, respectively, the leftmost node in V_1 and V_2 that are not aligned but may still be aligned in the solution.

[Figure 6 about here.]

In our implementation, in order to further speed–up the procedure, we compute a simple upper bound on $U(S \cup \{[f_1, j]\})$ before the first recursive call as follows. For each node $l > j$ in V_2, let $h(l)$ be the node in V_1 such that $h(l) > f_1$ and the Lagrangian profit $p_{[h(l),l]}$ is maximum. A solution whose Lagrangian value is certainly not worse than $U(S \cup \{[f_1, j]\})$ is obtained by adding to $S \cup \{[f_1, j]\}$ the lines $\{[h(l), l] : l = j + 1, \ldots, n_2\}$. If the associated (rounded down) value is not larger than t, we can skip the recursive call. Once all $h(l)$s have been determined in $O(n_1 n_2)$ time before the **for** loop, this test can be performed in constant time for each value of j, and we can exit the **for** loop as soon as the answer to the test is positive.

5.6 Parametric solution of the Lagrangian relaxed problem

As discussed above, both in the greedy heuristic and in the branch–and–bound procedure we keep on doing the following: add a line $[i, j]$ to the leftmost part of the solution S and compute $U(S)$ (without changing the Lagrangian multipliers). In this section, when we refer to subproblem (19)–(22), it is understood that we mean the version with objective function modified according to the Lagrangian multipliers.

We next show how to skip the explicit solution of each subproblem (19)–(22), finding the associated optimal value p_m in *constant* time. This means that the bottleneck in the parametric determination of $U(S)$ is the solution of (23)–(25), that takes $O(n_1 n_2)$ time, which is much better than $O(|E_1||E_2|)$, the time that would be required to compute it from scratch.

We first illustrate the simple Dynamic Programming procedure to solve problem (23)–(25). In particular, letting $\pi_{i,j}$ denote the maximum profit that can be achieved by selecting (pairwise compatible) noncrossing lines whose endpoints are $\leq i$ in V_1 and $\leq j$ in V_2, for $i = 0, \ldots, n_1$ and $j = 0, \ldots, n_2$, we have $\pi_{0,0} := 0$ and

$$\pi_{i,j} := \max\{\pi_{i,j-1},\ \pi_{i-1,j},\ \pi_{i-1,j-1} + p_{[i,j]}\},$$

for $i = 1, \ldots, n_1$ and $j = 1, \ldots, n_2$. The optimal solution is given by π_{n_1,n_2}. In graph theoretic terms, finding an optimal solution corresponds to finding the *longest path* from vertex $\{0,0\}$ to vertex $\{n_1, n_2\}$ in the directed acyclic graph $A = (W, F)$ with vertex set

$$W := (V_1 \cup \{0\}) \times (V_2 \cup \{0\})$$

and arc set $F := F_V \cup F_H \cup F_D$ where

$$F_V := \{(\{i-1, j\}, \{i, j\}) : i \in V_1, j \in V_2 \cup \{0\}\}$$

is the set of *vertical arcs*,

$$F_H := \{(\{i, j-1\}, \{i, j\}) : i \in V_1 \cup \{0\}, j \in V_2\}$$

is the set of *horizontal arcs*, and

$$F_D := \{(\{i-1, j-1\}, \{i, j\}) : i \in V_1 \cup \{0\}, j \in V_2 \cup \{0\}\}$$

is the set of *diagonal arcs* (elements in W are called *vertices* to distinguish them from the *nodes* in G_1 and G_2). Arcs in $F_V \cup F_H$ have no profit, whereas the profit of each arc $(\{i-1, j-1\}, \{i, j\}) \in F_D$ is equal to $p_{[i,j]}$. We let

$$d(\{h, l\}, \{i, j\}), \ (0 \leq h \leq i \leq n_1; 0 \leq l \leq j \leq n_2)$$

denote the length of the longest path from $\{h, l\}$ to $\{i, j\}$ in A. The optimal value of (23)–(25) is then $d(\{0,0\}, \{n_1, n_2\})$.

The above discussion implies that the optimal solution of subproblem (19)–(22) associated with line $m = [i, j]$ is obtained by defining the profits for the arcs in F_D according to (19) and computing the longest paths in A from $\{0,0\}$ to $\{i-1, j-1\}$ and from $\{i, j\}$ to $\{n_1, n_2\}$, the corresponding value being

$$d(\{0,0\}, \{i-1, j-1\}) + b_{mm} + d(\{i, j\}, \{n_1, n_2\}).$$

Suppose now the *leftmost* part of the solution is fixed, namely lines $[h_1, l_1]$, $[h_2, l_2]$, ..., $[h_q, l_q]$, with $h_1 < h_2 < \ldots < h_q$ and $l_1 < l_2 < \ldots < l_q$. Considering again subproblem (19)–(22) associated with line $m = [i, j]$ $(i > h_q, j > l_q)$, its optimal solution keeping into account the lines fixed has value

$$\sum_{r=1}^{q} b_{[h_r,l_r],m} + d(\{h_q, l_q\}, \{i-1, j-1\}) + b_{mm} + d(\{i, j\}, \{n_1, n_2\}).$$

This value can be found in constant time if a table with the longest path value from every vertex $\{h, l\}$ with $h \leq i-1$ and $l \leq j-1$ to $\{i-1, j-1\}$ is available (along with the scalar $d(\{i,j\},\{n_1,n_2\})$). Actually, we can compute and store only the values for $h \in N_1(i)$ and $l \in N_2(j)$, since the value for a generic vertex $\{h, l\}$ coincides with the one for vertex $\{h', l'\}$ where $h' = \min\{r \in V_1 : r \geq h, r \in N_1(i)\}$ and $l' = \min\{r \in V_2 : r \geq l, r \in N_2(j)\}$. This implies that computation and storage of the tables for all lines $m \in L$, which is done once for all when the Lagrangian multipliers have been fixed, has an overall time and space complexity of $O(|E_1||E_2|)$.

6 Metaheuristics

Several different heuristics based on genetic algorithms and steepest ascent local search were found to be effective. The effectiveness of each heuristic varies based on the size of the problem (small or large contact maps) and on the level of similarity between the two maps. In particular, heuristics that perform well for aligning similar (dissimiliar) contact maps do not perform well for aligning dissimilar (similiar) contact maps. A good alignment between two similar contact maps will generally map a contiguous set of residues in the first graph to a contiguous set of residues in the second graph, whereas between dissimilar contact maps the alignment has fewer edges and is less structured. Figure 1 shows an optimal alignment of two similar proteins.

6.1 Genetic Algorithm

The genetic algorithm [22, 15] mimics an evolutionary process whereby there is a population of individuals that, by the process of mutation and recombination, gradually improve over time. For optimization problems, an individual is a candidate solution, mutation is a slight perturbation of the solution parameters, and recombination is a "merge" of two candidate solutions to form a new candidate solution.

Classical applications of the genetic algorithm encode candidate solutions as bit strings, which would then be decoded to form the solution parameters for evaluation. The genetic operators are simple and independent of the problem – mutation is a bit-flip and recombination copies blocks of bits from existing candidate solutions to form new candidate solutions. However, unless great care is taken in choosing an

encoding scheme, doing so will frequently produce infeasible solutions.

Our application of the genetic algorithm avoids infeasible solutions by using the alignment edge, not the bit, as the fundamental element, and using special mutation and recombination operators. An alignment edge associates a residue in one contact map graph with a residue in the other. A mutation will slightly shift one edge, while recombination will pick edges out of two candidate solutions and create a new candidate solution using those edges.

[Figure 7 about here.]

The mutation operator shifts one side of a set of alignment edges. Any shifted edges that intersect unshifted edges are removed, and edges are randomly added in any available space. The edges to apply the shift to are determined as follows. For each edge, the probability of mutation is tested. If the test passes, we randomly choose four things: (1) The contact map to shift in (the top or bottom graph); (2) A set of nodes (all nodes to the left or right of the current edge); (3) A direction to shift (left or right); and (4) A distance to shift.

Figure 7(a) shows a candidate solution before two mutation events. The first mutation shifts the alignment edges on the circled nodes in Figure 7(a) to the right by one position, causing the right-most edge to be removed and a new edge to be be inserted. The result of this mutation is in Figure 7(b). The second mutation shifts the circled edges in Figure 7(b) 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.

[Figure 8 about here.]

The recombination operator creates a new candidate solution (the child) from two existing solutions (the parents). The parents are selected by a standard genetic algorithm method, i.e., randomly, but biased towards good solutions. A set of contiguous edges is randomly selected from one of the parents and is copied directly to the child. Next, all edges from the second parent that do not cross edges in the child are copied to the child. Finally, new edges are added in any available positions, as done for the mutation operator. In Figure 8(d), the dashed edges came from the first parent, the dotted edges from the second parent, and the

thick edge was added randomly.

The genetic algorithm seems to be stable across a wide range of sizes, however, it needs to be tuned specifically for similar or dissimilar contact maps. Two heuristics are formed:

GA similiar has a small population, initialized by several applications of the mutation operator to the identity alignment, with a high probability of crossover and low rate of mutations.

GA dissimiliar has a large population, initialized by inserting random alignment edges until no more edges can be added, with a small probability of crossover, and a high rate of mutation.

6.2 Local Search

The local search heuristic algorithms follow the standard approach: Let s be a feasible current solution. The *neighborhood of* s is the set of solutions which can be obtained by applying a *move* to s. If all solutions in the neighborhood are no better than the current solution s, the solution s is a local optimum and the search is terminated. Otherwise, the move that results in the best solution value is applied to s, and the search continues. Since converging to a local optimum is very fast, the search can be repeated many times, each time starting from a random feasible solution.

A feasible contact map alignment solution is identified by a pair of lists of the same size, (A, B), each containing an increasing sequence of vertices in a contact map graph. We denote by $A = (a_1 < \ldots < a_k)$ the list of vertices in G_1, and by $B = (b_1 < \ldots < b_k)$ the list of vertices in G_2. The contact map graph alignment is found by mapping residue a_i in the first graph to residue b_i in the second graph. Figure 9(a) shows a feasible solution, with $A = (1, 3, 4, 6, 8)$ and $B = (2, 3, 6, 7, 8)$. The fourth line is $[a_4, b_4] = [6, 8]$.

[Figure 9 about here.]

Our two local search algorithms differ only in the definition of the moves that generate a neighborhood.

LS dissimilar use one move that adds a single specific line to the solution, and removes any lines that cross the new line. The move results in big "jumps" in the solution space, i.e., by introducing very skewed lines and by removing many lines at once. The move $M(a, b)$ is defined for all $a \in G_1 - A$ and $b \in G_2 - B$ and will add the line $[a, b]$, removing all crossing lines $[a_j, b_j]$. Figure 9(b) shows the line $[7, 4]$ being added,

which forces the removal of two lines, $[a_2, b_2]$ and $[a_3, b_3]$. The resulting solution is $A = (1, 3, 7, 8)$ and $B = (2, 3, 4, 8)$.

LS similar uses two moves, a *decreasing* move and an *increasing* move, both of which do not introduce very skewed lines easily and are thus suited for similar proteins, in which good solutions are made of many parallel lines. The decreasing move, $M^-(a, b)$, defined for $a \in A$ and $b \in B$, removes a from A and b from B. Figure 9(d) shows the decreasing move $M^-(6, 3)$, which removes a_4 and b_2 from their respective lists. This has the effect of modifying all lines $[a_i, b_i], 2 \leq i \leq 4$ in the original solution. The increasing move, $M^+(a, b)$, where $a \in G_1 - A$ and $b \in G_2 - B$, adds a into A and b into B. Figure 9(c) shows the increasing move $M^+(7, 4)$ which will insert 7 into A before a_5 and 4 into B after b_2, similar to the decreasing move, this modifies all lines $[a_i, b_i], 3 \leq i \leq 4$ in the original solution.

7 Computational results

The algorithms were implemented in C and run on a Pentium PC. For our tests, we used protein structures from the Protein Data Bank (PDB) [4].

In a first experiment we selected a set of small proteins (about 70 residues each) which were compared using the B&C and LR approaches. The results, discussed in Section 7.1 show how the latter outperforms the former.

Our second experiment explores how well the CMO measure correlates with other parameters used to determine if two protein structures are similar, e.g., those adopted by the SCOP protein structure server. The CMO itself has a "hidden" parameter, the contact threshold δ below which two residues are considered in contact, and to use the CMO value as a classifier, we define a similarity threshold τ, above which two proteins are considered to belong to the same family. In our experiment, we used a set of proteins chosen from four SCOP families, and studied for which values of τ and δ our program was able to correctly classify pairs of proteins. This experiment is described in detail in Section 7.2.

Finally, we use the pairwise classification from the previous experiment to retrieve the four protein families. This experiment is described in Section 7.3.

7.1 Selected proteins from PDB

[Table 1 about here.]

We ran our methods on a set of 269 proteins with 64 to 72 residues and 80 to 140 contacts each, using a contact threshold δ of 5Å. The set was chosen to contain both a large number of similar proteins (alpha helices), as well as a large number of dissimilar proteins (small beta-sheets). An all-against-all computation would require 36,046 alignments, however, we selected a subset of 597 alignments, so as to have approximately the same number of similar and dissimilar pairs.

For the B&C approach, 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, so that in every instance we solved the LP relaxation at the root node even if took more than 1 hour. The heuristics were applied at every node, and were limited to at most five minutes per node.

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 LP is to solve.

In Table 1 we sort the instances by the value of the gap between best solution found from a heuristic and the upper bound, where 0 is represents instances solved optimally. For each gap level, we report the total instances solved, average and maximum number of residues and contacts, and the performance of the heuristics. For each heuristic, we report how many instances it found the best solution. The "GA dissimilar" heuristic is clearly superior to the others, finding 52 of the 55 optimal solutions, and finding more than twice as many best solutions as the other heuristics at all gap levels.

The same 597 pairs were then compared by using the LR approach. For each instance we allowed a maximum running time of 1 minute. Despite the smaller running time the LR method solved 72 instances to optimality and only for 172 was the gap larger than 5. For all instances, the upper bound computed by LR was at least as good as that computed by B&C, however, most of the time the bounds were equal. Note that we are not finding the best Lagrangian multipliers (Sections 5.2 and 5.3) and hence, in principle, our upper bound may be worse than U_1.

[Table 2 about here.]

By using the LR approach we then compared, in a weekend on a desktop PC, all 36,046 pairs. To speed up the computation, we only explored the root node of the search tree and we did not apply the greedy heuristic. Note that running the B&C method on all these instances, with a time limit of 1 hour/problem, would have taken about four years.

For 14,912 pairs of proteins the gap between the upper and lower bounds was 10 or less, with 1,680 instances solved to optimality.[1] Table 2 shows the gap distribution.

The fact that for most instances the gap is larger than 10 reflects the present difficulty to solve instances associated with substantially different proteins, even of relatively small size. On the other hand, the LR method can find optimal maps for pairs of similar proteins, even of very large size, within few seconds. For instance, in less than one minute we have optimally aligned `1hkbA` to `1hkcA`, with 891 and 887 contacts and 1944 and 1973 contacts, respectively.

7.2 Setting the thresholds: the Skolnick clustering test

In order to assess the quality of the CMO as a protein family classifier, we define a function $sim(i,j) \in \{true, false\}$ that decides if a pair of proteins is in the same family or not. For the CMO measure, we have chosen to use

$$sim(i,j) = \frac{CMO(i,j)}{\min\{c_i, c_j\}} > \sigma$$

where $CMO(i,j)$ is the CMO value of proteins i and j, and c_i and c_j are the number of contacts of the two maps.

A first question arises about how robust this threshold is across various inputs, that is, how strongly does σ depend on the proteins being classified? This issue deserves further investigation, possibly from those with a more sophisticated knowledge of what it means for two proteins to be biologically similar.

Instead, we focus on how such a classifier can be used to assess a very delicate parameter, on whose setting the whole utility of the CMO measure ultimately depends, the contact threshold δ. To the best of our knowledge, the setting of this critical threshold has never been studied in a parametric way before.

It is clear that for a very small δ there are no contacts, and all contact maps are the same – the empty

[1] Thereby justifying the title of this paper!

graph. The same phenomenon happens at the opposite extreme: for large δ, all pairs of residues are in contact, and again all contact maps are the same – the complete graph. Values of δ between these extreme cases start differentiating contact maps for different proteins, and want to find the best δ, for which dissimilar proteins produce dissimilar contact maps, while similar proteins produce similar contact maps.

[Table 3 about here.]

[Table 4 about here.]

Our experiment used a test set of large proteins suggested to us by Jeffrey Skolnick [42]. The set contains 33 proteins with a total of 40 domains classified by SCOP into four families, as shown in Table 3. The CMO score for all 780 pairs of proteins, for $4.5 \leq \delta \leq 8.75$, was computed using the LR method and the four heuristics. At each contact threshold δ, we found the similarity threshold σ that minimized the total number of classification errors. The result of the experiment is that contact thresholds δ smaller than 5.5Å are too small to limit the classification error to less than 10%. The best contact threshold δ is 7.5Å paired with a contact similarity of σ of .636. The full results are shown in Table 4.

[Table 5 about here.]

In the above experiment, a different similarity threshold σ was obtained for each contact threshold δ. In order to check how well a constant similarity threshold would have performed, we reran our analysis using σ of .636. The results, reported in Table 5, show that for all maps with a δ of at least 6.25Å the error is below 10%, and for maps with δ at least 7.0Å the error is below 5%. We conclude that the choice of σ is not strongly dependent on the choice of a contact threshold δ.

7.3 Retrieving the families

Given a perfect classifier $sim(i, j)$, recovering the family clustering from a set of proteins is a trivial task. One can form a new cluster using some protein a from the set of unclustered proteins, add all proteins i such that $sim(a, i)$ is true to the cluster, remove the proteins in the cluster from the data and repeat until there are no unclustered proteins.

Unfortunately, the classifier we described in the previous section gives only an approximation of the perfect classifier. In the presence of false positives and false negatives, the transitive property $sim(i,j) \wedge sim(j,k) \mapsto sim(i,k)$ may not apply, and hence there is a computational problem of retrieving the clusters from the pairwise classifications. One way of doing this is to use any of the many known algorithms for clustering in the literature, which we will not describe here.

Instead, we experimented with the following reduction to the Hamming TSP problem. For a δ of 7.5Å and σ of .636, we obtain a 0-1 matrix A, with rows and columns indexed by the proteins, such that $A[i,j] = 1$ if proteins i and j are classified in the same family, and 0 otherwise. The matrix A can then be used to cluster the proteins. Under a perfect clustering, there would exist a permutation for the rows (and columns) which exhibits a block-diagonal structure,and hence, also the consecutive-1 property in each row. In presence of errors, the best arrangement, the one that minimizes the number of times that a false positive is put in a cluster or a false negative is not put in a cluster, can be found by solving a TSP in which the distance between two columns of A is given by their hamming distance.

As the instance size was small, such a TSP could be defined and solved to optimality The result, reported in Figure 10 shows how the clusters are neatly defined in the resulting best arrangement for rows and columns. By a close inspection of the solution, we can also see how 33 of the 34 false negatives at are, in fact, due to just one protein, `1rn1`, that has three domains. The remaining false negative is due to only one pair, `1dps-1b71`.

[Figure 10 about here.]

References

[1] W.P. Adams and H.D. Sherali. A tight linearization and an algorithm for zero-one quadratic programming problems. *Management Science*, 32:1274–1290, 1986.

[2] E. Balas, S. Ceria, and G. Cornuejols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, 58:295–324, 1993.

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

[4] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000. The PDB is at `http://www.rcsb.org/pdb/`.

[5] A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. *Operations Research*, 47:730–743, 1999.

[6] A. Caprara and G. Lancia. Structural alignment of large-size proteins via lagrangian relaxation. In *Proceedings of the Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 100–108, New York, NY, 2002. ACM Press.

[7] P. Carraresi and F. Malucelli. A reformulation scheme and new lower bounds for the QAP. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 147–160. AMS Press, 1994.

[8] E. Çela. *The Quadratic Assignment Problem: Theory and Algorithms.* Kluwer Academic Publishers, 1998.

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

[10] I. Eidhammer, I. Jonassen, and W. R. Taylor. Structure comparison and structure prediction. *J. Comp. Bio.*, 5(5):685–716, 2000.

[11] M. L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 27:1–18, 1981.

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

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

[14] A. Godzik, J. Skolnick, and A. Kolinski. A topology fingerprint approach to inverse protein folding problem. *Journal of Molecular Biology*, 227:227–238, 1992.

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

[16] D. Goldman. PhD thesis, U.C. Berkeley, 2000.

[17] D. Goldman, S. Istrail, and C. Papadimitriou. Algorithmic aspects of protein structure similarity. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 512–522, New York, NY, 1999. IEEE.

[18] M. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, NY, 1980.

[19] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.

[20] T. F. Havel, I. D. Kuntz, and G. M. Crippen. Effect of distance constraints on macromolecular conformation. *Biopolymers*, 18, 1979.

[21] M. Held and R. M. Karp. The traveling salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1:6–25, 1971.

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

[23] L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *J. Mol. Biol*, 233:123–138, 1993.

[24] L. Holm and C. Sander. 3-D lookup: fast protein structure searches at 90% reliability. In *Proceedings of the Annual International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 179–187, Menlo Park, CA, 1995. AAAI Press.

[25] L. Holm and C. Sander. Mapping the protein universe. *Science*, 273:595–602, 1996.

[26] D. S. Johnson and M. A. Trick, editors. *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. AMS Press, 1996.

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

[28] L. G. Kachian. A polynoimial algorithm for Linear Programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.

[29] N. Karmarkar. A new polynomial time algorithm for linear programming. *Combinatorica*, 4:375–395, 1984.

[30] G. Lancia, R. Carr, B. Walenz, and S. Istrail. 101 optimal PDB structure alignments: A branch-and-cut algorithm for the maximum contact map overlap problem. In *Proceedings of the Annual International Conference on Computational Biology (RECOMB)*, pages 193–202, New York, NY, 2001. ACM Press.

[31] H.-P. Lenhof, K. Reinert, and M. Vingron. A polyhedral approach to RNA sequence structure alignment. *Journal of Computational Biology*, 5(3):517–530, 1998.

[32] L. Lovász and A. Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM Journal on Optimization*, 1:166–190, 1991.

[33] A. Lucas, K. Dill, and S. IStrail. Contact maps and the computational statistical mechanics aspects of protein folding. *In preparation.*

[34] L. Mirny and E. Domany. Protein fold recognition and dynamics in the space of contact maps. *Proteins*, 26:391–410, 1996.

[35] J. Moult, T. Hubbard, S. Bryant, K. Fidelis, J. Pedersen, and Predictors. Critical assessment of methods of proteins structure prediction (CASP): Round II. *Proteins*, Suppl.1:dedicated issue, 1997.

[36] J. Moult, T. Hubbard, K. Fidelis, and J. Pedersen. Critical assessment of methods of proteins structure prediction (CASP): Round III. *Proteins*, Suppl.3:2–6, 1999.

[37] A.G. Murzin, S.E. Brenner, T. Hubbard, and C. Chothia. Scop: a structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.*, 247:536–540, 1995.

[38] S. B. Needleman and C. D. Wunsch. An efficient method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

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

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

[41] C.A. Orengo, A.D. Michie, S. Jones, D.T. Jones, M.B. Swindells, and J.M. Thornton. CATH- a hierarchic classification of protein domain structures. *Structure*, 5(8):1093–1108, 1997.

[42] J. Skolnick. Personal communication. 2000.

[43] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[44] M. Vendruscolo, E. Kussell, and E. Domany. Recovery of protein structure from contact maps. *Fold. Des.*, 2:295–306, 1997.

Acknowledgments

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

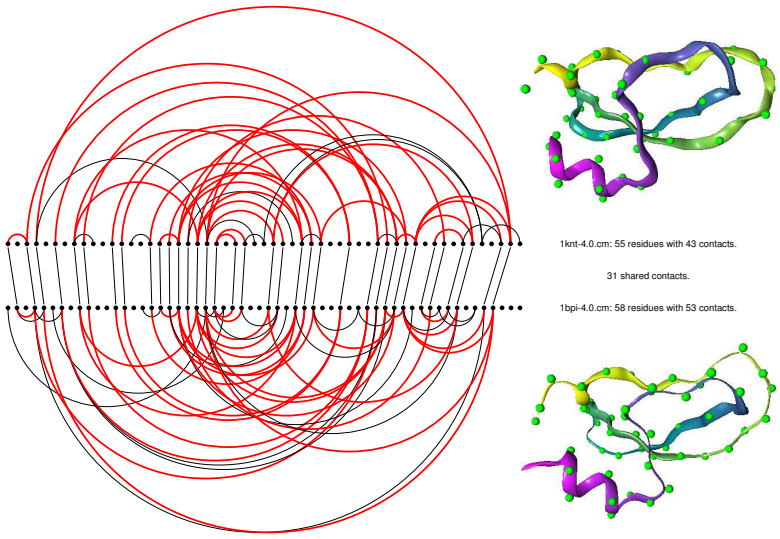

Figure 1: An optimal alignment of two 4Å threshold contact maps of proteins `1bpi` and `1knt`.

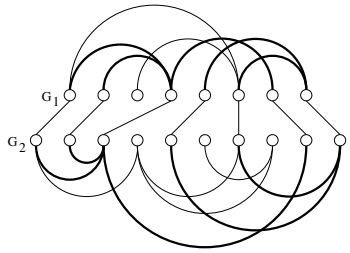

Figure 2: An alignment of value 5.

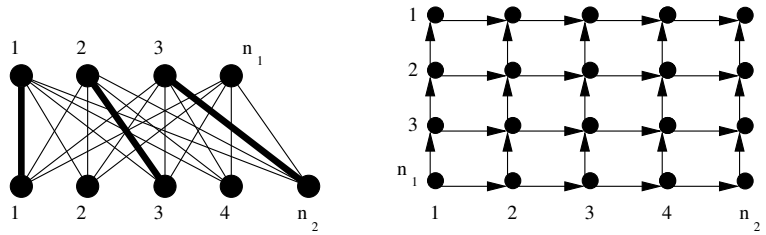

Figure 3: (a) A noncrossing matching (bold). (b) The directed grid.

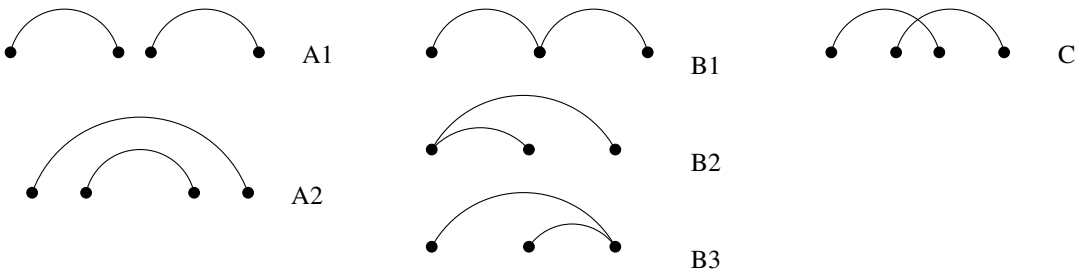

Figure 4: Relationships between edges.

```
procedure greedy;
begin
    S := ∅; f_2 := 1;
    for i := 1 to n_1 do
        U_max := 0;
        for j := f_2 to n_2 do
            if U(S ∪ {[i, j]}) > U_max then
                j_max := j; U_max := U(S ∪ {[i, j]});
            end if
        end for
        if U(S ∪ {[i, ∅]}) ≤ U_max then
            S := S ∪ {[i, j_max]}; f_2 := j_max + 1;
        else
            S := S ∪ {[i, ∅]};
        end if
    end for
end.
```

Figure 5: Pseudo–code implementation of the greedy heuristic.

```
procedure branch–and–bound(f_1, f_2, S);
begin
    if ⌊U(S)⌋ ≤ t then return;
    if f_1 = n_1 + 1 then success;
    for j := f_2 to n_2 do
        branch–and–bound(f_1 + 1, j + 1, S ∪ {[f_1, j]});
    end for
    branch–and–bound(f_1 + 1, f_2, S ∪ {[f_1, ∅]});
end.
```

Figure 6: Pseudo–code implementation of the branch–and–bound procedure for a given target solution value t.

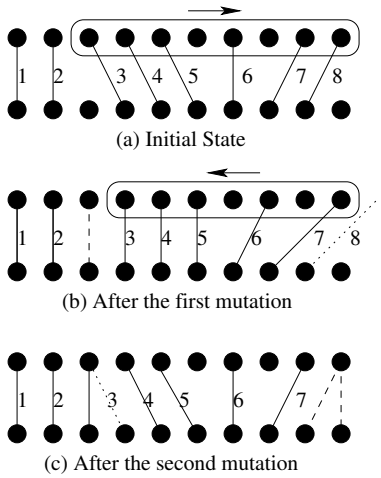

Figure 7: The mutation operator. Dotted lines are edges that have been removed by a mutation, dashed lines are edges that have been added after a mutation is performed.

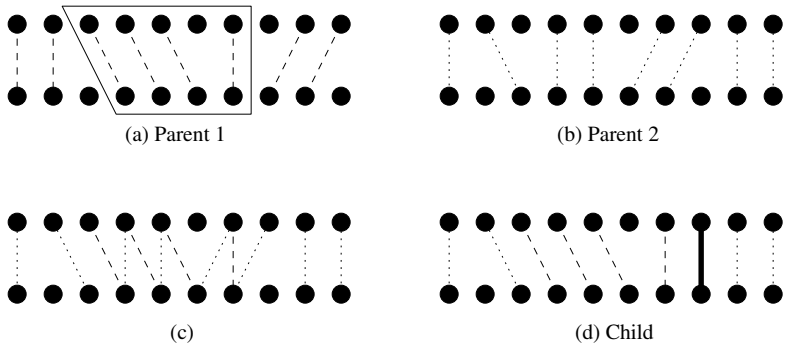

Figure 8: The recombination operator. (a) The first parent, showing the set of edges selected for inclusion in the child. (b) The second parent. (c) The partially constructed child, showing the edges from the first parent and all edges from the second parent. (d) The fully constructed child, with conflicting edges from the second parent removed, and a single new edge added.

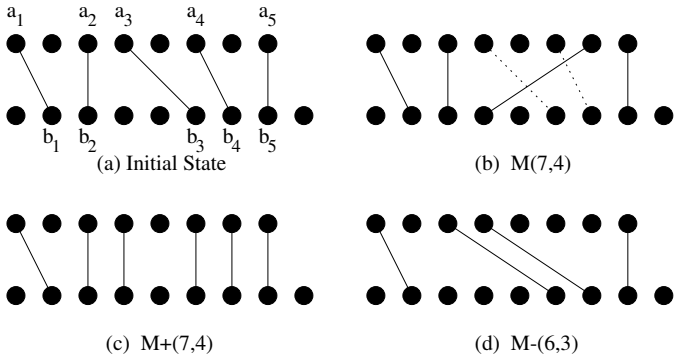

Figure 9: Moves for the two algorithms. (a) Starting solution s. (b) A move from the LS dissimiliar algorithm applied to s. (c) An increasing move applied to s. (d) A decreasing move applied to s.

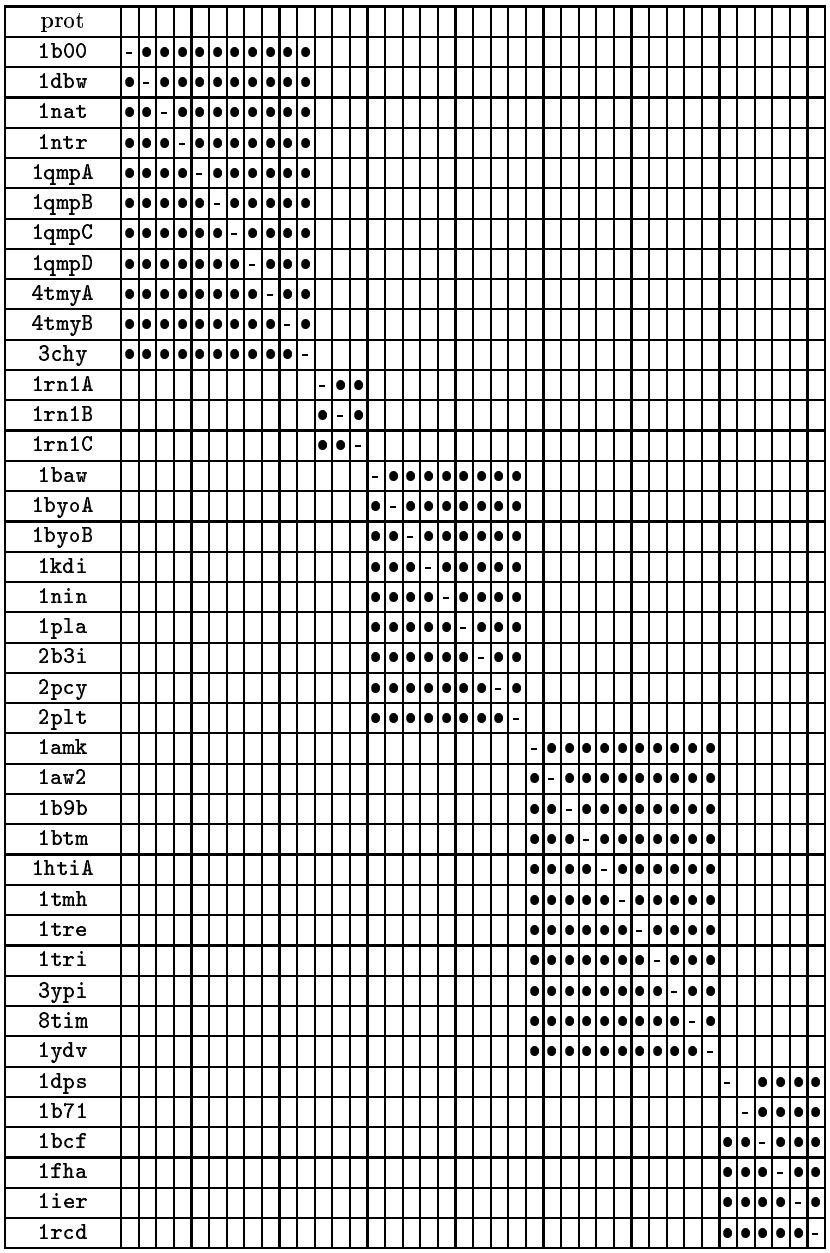

Figure 10: Clustering the proteins of the Skolnick data set. The permutation found by TSP (shown for the rows) is applied to both rows and columns.

Optimality Gap	0	1	2	3	4	5	>5
n. instances	55	62	80	75	77	72	176
	9.2%	10.4%	13.4%	12.6%	12.9%	12.1%	29.5%
avg n. residues	66.4	66.8	66.7	67.0	67.03	66.8	66.8
max n. residues	69	72	71	72	71	72	72
avg n. contacts	61.1	56.3	57.3	59.7	61.5	64.7	71.4
max n. contacts	92	89	93	95	88	89	133
GA similar	23	14	19	17	10	11	41
GA dissimilar	52	59	77	73	76	69	161
LS similar	25	20	35	31	33	35	82
LS dissimilar	5	0	0	1	5	12	53

Table 1: Branch–and–Cut performance. Number of instances solved, and the number of times each heuristic found the best solution. More than one heuristic can find the best solution.

Optimality Gap	0	1	2	3	4	5	6	7	8	9	10	>10
n. instances	1680	215	373	491	640	937	1233	1701	2235	2558	2849	21134
	4.6%	0.5%	1.0%	1.3%	1.7%	2.6%	3.4%	4.7%	6.2%	7.1%	7.9%	58.5%
avg n. residues	57.2	55.6	55.6	55.6	55.5	55.6	55.7	55.9	56.0	56.2	56.4	57.7
max n. residues	68	66	66	68	68	68	68	68	68	68	68	68
avg n. contacts	98.1	81.6	81.1	80.1	79.1	81.5	83.2	85.3	87.3	89.6	92.2	101.4
max n. contacts	137	137	137	137	137	137	137	137	137	137	137	137
GA similar	1240	179	316	435	585	851	1121	1565	2046	2313	2601	18678
GA dissimilar	707	30	45	80	65	116	130	188	246	276	280	2217
LS similar	723	18	21	31	29	31	57	78	116	145	157	1698
LS dissimilar	406	3	4	3	3	8	24	33	48	77	126	1845

Table 2: LR results for 36,046 pairs of small proteins.

	Fold	Family	Residues	Seq. Sim.	RMSD	Proteins
1	Flavodoxin-like	CheY-related	124	15-30%	< 3Å	1b00, 1dbw, 1nat, 1ntr, 1qmp (A,B,C,D), 1rn1 (A,B,C), 3chy, 4tmy (A,B)
2	Cupredoxins	Plastocyanin/ azurin-like	99	35-90%	< 2Å	1baw, 1byo (A,B), 1kdi, 1nin, 1pla, 2b3i, 2pcy, 2plt
3	TIM beta/ alpha-barrel	Triosephosphate isomerase	250	30-90%	< 2Å	1amk, 1aw2, 1b9b, 1btm, 1hti, 1tmh, 1tre, 1tri, 1ydv, 3ypi, 8tim
4	Ferratin-like	Ferritin	170	7-70%	< 4Å	1b71, 1bcf, 1dps, 1fha 1ier, 1rcd

Table 3: The Skolnick set.

δ	σ	Errors	Correct	False Positives	False Negatives	True Positives	True Negatives
4.50	.925	231 (29%)	549	60	171	26	523
4.75	.930	188 (24%)	592	0	188	9	583
5.00	.730	157 (20%)	623	2	155	42	581
5.25	.545	128 (16%)	652	34	94	103	549
5.50	.569	66 (8%)	714	9	57	140	574
5.75	.517	68 (6%)	712	6	62	135	577
6.00	.533	50 (6%)	730	1	49	148	582
6.25	.538	51 (6%)	729	1	50	147	582
6.50	.564	45 (5%)	735	0	45	152	583
6.75	.559	41 (5%)	739	0	41	156	583
7.00	.593	36 (4%)	744	0	36	161	583
7.25	.596	38 (4%)	742	1	37	160	582
7.50	.636	34 (4%)	746	0	34	163	583
7.75	.624	37 (4%)	743	0	37	160	583
8.00	.630	37 (4%)	743	0	37	160	583
8.25	.637	37 (4%)	743	0	37	160	583
8.50	.663	37 (4%)	743	0	37	160	583
8.75	.636	37 (4%)	743	0	37	160	583

Table 4: The number of errors for the best similarity threshold σ at each contact threshold δ.

δ	σ	Errors	Correct	False Positives	False Negatives	True Positives	True Negatives
4.50	.636	424 (54%)	356	382	42	155	201
4.75	.636	321 (41%)	459	239	82	115	344
5.00	.636	202 (25%)	578	97	105	92	486
5.25	.636	145 (18%)	635	2	143	54	581
5.50	.636	103 (13%)	677	1	102	95	582
5.75	.636	130 (16%)	650	0	130	67	583
6.00	.636	81 (10%)	699	0	81	116	583
6.25	.636	72 (9%)	708	0	72	125	583
6.50	.636	57 (7%)	723	0	57	140	583
6.75	.636	56 (7%)	724	0	56	141	583
7.00	.636	43 (5%)	737	0	43	154	583
7.25	.636	41 (5%)	739	0	41	156	583
7.50	.636	34 (4%)	746	0	34	163	583
7.75	.636	39 (5%)	741	0	39	158	583
8.00	.636	38 (4%)	742	0	38	159	583
8.25	.636	39 (5%)	741	0	39	160	581
8.50	.636	37 (4%)	743	0	37	160	583
8.75	.636	37 (4%)	743	0	37	160	583

Table 5: Performance for $\sigma = .636$.