OCT 3 2000

SANDIA REPORT

SAND2000-2171
Unlimited Release
Printed September 2000

Branch-and-Cut Algorithms for Independent Set Problems: Integrality Gap and an Application to Protein Structure Alignment

Robert D. Carr, Guiseppe Lancia, and Sorin Istrail

Sandia National Laboratories

SAND2000-2171
Unlimited Release
Printed September 2000

Branch–and–Cut Algorithms for Independent Set Problems: Integrality Gap and an Application to Protein Structure Alignment

Robert D. Carr and Giuseppe Lancia
Applied Mathematics Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1110

Sorin Istrail
Celera Genomics
Rockville, MD

Abstract

We discuss the effectiveness of branch and cut for solving large instances of the independent set problem. Typical LP formulations, even strengthened by clique inequalities, yield poor bounds for this problem. We prove that a strong bound is obtained by the use of the so called "rank inequalities", which generalize the clique inequalities. For some problems the clique inequalities imply the rank inequalities, and then a strong bound is guaranteed already by the simpler formulation. This is the case of the *contact map overlap problem*, which was proposed as a measure for protein structure alignments. We formalize this problem as a particular, large independent set problem which we solve by integer programming. We strengthen our formulation by the use of clique inequality cuts. Although there are exponentially many cliques, we show how to separate over them in polynomial time. Unprecedented computational results on real data show the effectiveness of our approach.

1 Introduction

The *Maximum Independent Set* (MIS) is one of the classic problems in combinatorial optimization. Both the *cardinality* version and the *weighted* version of MIS have been studied. The literature on this problem –or its twin, the *Maximum Clique*– is vaste and dates back to the beginning of the field. 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 maximum clique/independent set [20, 3, 6, 16]. 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]. The most successful approach to the exact solution of combinatorial optimization problems is probably *Integer Linear Programming*, which has been applied profitably in very many cases [19, 7, 17]. The Integer Programming 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, where the upper bound comes from the *linear programming relaxation.* The LP relaxation is the same question, only that the variables are not restricted to be integer, which makes it polynomially solvable. A formulation is as successful as the strength of its LP bound. That is, if we can prove that the value of the objective function over the relaxation is close to the value over the integers, then the bound, and hence the pruning of the search space, will be effective. It is often the case that in order to obtain better bounds, the formulation is reinforced by the use of additional constraints, called *cuts*, and the resulting approach is known as branch and cut. Cuts 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, nice formulation as an integer programming problem. Unfortunately, this formulation gives a terribly bad bound, e.g. the bound can be as big as $n/2$ for an instance with optimal value of 1. The formulation can be strengthened by the use of clique–inequalities cuts. These are constraints that say that each clique can have at most one node in common with any independent set. By using some concepts from Ramsey Theory, we will show that even with clique inequalities the gap between the LP value and the optimum can be very bad. In this paper we pinpoint the fundamental constraints for the Maximum Independent Set as the "rank inequalities". Their addition guarantees an $O(\log n)$ gap between the LP bound and the optimum. This does not contradict the known complexity results for MIS (stated in Section 2) since, in general, it is NP-complete to find all violated rank inequalities. However, the theory we develop here can be also useful in practice, since there are some fortunate cases in which a formulation implies the rank inequalities and hence we know that the bound will be strong. This is the case for a particular problem studied in this paper, namely the *maximum contact map overlap* (CMO) problem.

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 their euclidean distance is within a given threshold when the protein is folded. The contact map overlap problem tries to capture the similarity in the 3D folds of two proteins by comparing their contact maps. In this sense, it is a new way of aligning 3D structures. The value of an alignment of the residues of one protein vs. the other, is taken as the number of contacts in the first contact map whose endpoints are aligned with residues that also share a contact in the second contact map. The CMO, introduced in [9] and proved NP-hard in [12], is emerging as the most important practical measure of protein structure similarity.

The CMO problem can be reduced to a very large MIS problem on a suitable graph. In this paper we formulate the CMO problem as an Integer Program and use clique–inequalities and some other cuts to strengthen the bound. 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 $O(n^2)$ the most violated clique inequality and add it to the LP formulation. Finding cliques in a graph is in general a difficult problem. However, in our case we can solve it effectively since we will show that the underlying graph is perfect. For this type of graphs, the clique inequalities imply the rank inequalities, which gives a theoretical explanation of the practical good performance of our algorithm. We have implemented our ideas in a computer program, which has been run on some 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 been able to align optimally several pairs of proteins with contact maps of 50 to 100 residues/contacts. These values are typical of small and moderate–sized proteins, of which there is in abundance in PDB.

2 The Maximum Independent Set Problem

Given an undirected graph $G = (V, E)$, $|V| = n$, a subset $V' \subseteq V$ of nodes is an *independent* or *stable* set if no two nodes in V' are joined by an edge. The Maximum Independent Set problem consists in determining an independent set of maximal weight or cardinality. Given a weight function c on the vertices ($c := 1$ in the cardinality version), the standard LP formulation associates a binary variable x_v to each node $v \in V$:

$$\max\left\{\sum_{v\in V} c_v x_v \mid x_u + x_v \leq 1 \ \forall\{u,v\} \in E,\ x_v \in \{0,1\} \forall v \in V\right\}. \tag{1}$$

The LP relaxation of the cardinality version of (1) has the optimal value $\lceil n/2 \rceil$, achieved when $x_v = 1/2$ for all $v \in V$. This bound can be $O(n)$ times larger than the true optimum: e.g. if G is a clique, the optimum is 1. Since any clique $Q \subseteq V$ can have at most one node in common with any independent set, the following *clique–inequality* constraints can be added to (1):

$$\sum_{v\in Q} x_v \leq 1 \qquad \forall Q \text{ clique in } G \tag{2}$$

Adding contstraints (2) presents two types of difficulites. First, findig cliques in a graph is itself a hard problem. 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 inequalities (2), tells us if the solution is in fact feasible for all the inequalities (2), or else returns us a violated clique inequality. In the next section we will argue that even in this case we are not guaranteed that the LP relaxation of (1) and (2) will in fact be a good approximation of the optimum.

The complexity theory results for the MIS paint a bleak picture as far as our ability to approximate this problem. Since the maximum independent set in a graph is the maximum clique in the complement graph, the complexity results are the same as those for the maximum clique problem. If $P \neq NP$, then MIS cannot be approximated to within a factor of $O(n^\epsilon)$, where ϵ is a fixed positive constant defined for MIS, [10, 2, 1]. Under stronger complexity assumptions, MIS cannot be approximated to within a factor of $O(n^{0.5-\epsilon})$, [13]. The best approximation factor for MIS found so far is a mere $O(\frac{n}{\log^2 n})$ for the cardinality version, [5].

2.1 Ramsey Theory and the Integrality Gap

We just saw that one cannot expect to ever be able to find a reasonable approximation for MIS. A tacit polyhedral combinatorics axiom is that if there is a reasonable approximation for MIS, there should be an LP relaxation for MIS which is solvable in polynomial time and has a reasonable size integrality gap. Such an LP relaxation would not allow any of the clique inequalities to be violated by too large a factor. Call the LP relaxation that consists of the clique inequalities for all of the maximal cliques the *clique relaxation*. In order for this relaxation to be polynomially solvable, we need an efficient separation algorithm for these exponentially many clique inequalities. Let x^* be our current fractional solution. With the vertices of G weighted by x^*, this separation algorithm must be able to find a clique of weight more than 1 if such a clique exists. Hence, our separation algorithm must be powerful enough to solve the maximum weighted clique problem. However, maximum weighted clique is as difficult to approximate as MIS.

Suppose we eliminated this obstacle to a small integrality gap by imagining we have a separation oracle that finds a clique of weight more than 1 if such a clique exists. What can we say then about the integrality gap? We argue here that the results from a branch of graph theory called *Ramsey theory* make it unlikely that the gap is a constant even in this case.

Ramsey theory in part studies the occurrence of large cliques and large independent sets in arbitrary graphs. An idea in Ramsey theory is that if a graph is large enough and has no large cliques (independent sets), then it must have large independent sets (cliques). More formally, there is a smallest integer $R(k, l)$ such that every graph with at least $R(k, l)$ vertices has either a clique of k vertices or an independent set of l vertices. This number is called the (k, l)–th *Ramsey number.*

Ramsey numbers are notoriously difficult to compute, and seem to grow reasonably quickly. We are particularly interested in their growth when the clique number $k = 3$, and conjecture the following.

Conjecture 1

$$\lim_{l\to\infty} \frac{R(3,l)}{l} = \infty. \tag{3}$$

An integrality gap result follows as a corollary to this conjecture.

Corollary 1 *If conjecture 1 holds, there is no constant integrality gap between MIS and the clique LP relaxation.*

Proof Let $r \in \mathbf{N}$ be given. We now produce a graph where the integrality gap exceeds $r/2$. Choose $l_r \in \mathbf{N}$ such that $\frac{R(3,l_r)-1}{l_r-1} > r$. Choose a graph G_r having $R(3, l_r) - 1$ vertices that has no clique of size 3 and no independent set of size l_r. An optimal solution to the clique LP relaxation is $x_i^* = 1/2$ for all $i \in V_r$. The cost of this solution is $\frac{R(3,l_r)-1}{2}$. The MIS solution has cost $l_r - 1$. Hence, this integrality gap exceeds $r/2$, as required. $\diamond$

Ramsey theory thus shows that inequalities other than the clique inequalities are needed in an LP relaxation for there to be a constant integrality gap. The analysis above suggests which additional inequalities are needed: For each $H \subset V$, we have the valid inequality

$$x(H) \leq r(H), \tag{4}$$

where the *rank function* $r(H)$ is the maximum size of any independent set contained in H. These inequalities are called the *rank inequalities* of MIS. We call the LP relaxation consisting precisely of all of the rank inequalities the *rank LP relaxation.*

Theorem 1 *The integrality gap between weighted MIS and its rank LP relaxation is at most* $\log n + 1$.

Proof Denote a maximizer for the rank LP relaxation by x^*. We will demonstrate that $\frac{x^*}{\log n+1}$ is less than or equal to a convex combination of incidence vectors of independent sets of G. From an averaging argument, one of these independent sets costs more than $\frac{c \cdot x^*}{\log n+1}$, from which our integrality gap result follows.

We first find the smallest integer k so that for every vertex $v \in V$, $x_v^* = \frac{l_v}{k}$, where $l_v \in \mathbf{N}$. We will decompose x^* into a linear combination of incidence vectors of independent sets with linear multipliers $\frac{1}{k}$. We start by having $x^* = y^0 + z^0$, where $y^0 := x^*$ and $z^0 := 0$. We successively form y^i, z^i for $i = 1, 2, \ldots$ by taking $\frac{1}{k}$ times an incidence vector of an independent set away from y^{i-1} and adding it to z^{i-1}. Hence, $y^i + z^i = x^*$ for $i = 1, 2, \ldots$. We terminate our process at an integer *last* when $y^{last} = 0$ and $z^{last} = x^*$. We thus have z^{last} expressed as a linear combination of independent sets. Define

$$V_i := \{v \in V \,:\, y_v^{i-1} > 0\} \text{ for } i = 1, 2, \ldots.$$

Consider a maximum cardinality indepent set x^i for $G[V_i]$. Define

$$y^i := y^{i-1} - \frac{1}{k}x^i,\ z^i := z^{i-1} + \frac{1}{k}x^i \text{ for } i = 1, 2, \ldots.$$

Because x^* satisfies the rank inequalities for the vertex sets V_i, we have that

$$x^*(V_i) \leq x^i(V_i) \text{ for } i = 1, 2, \ldots.$$

We can now determine a lower bound on how much y^i decreases in each iteration, and thus an upper bound on how many terms are in our linear decomposition of x^* into independent sets. We have

$$\begin{array}{rcl} y^i(V) & = & y^{i-1}(V) - \frac{1}{k}x^i(V) \leq y^{i-1}(V) - \frac{1}{k}x^*(V) \\ & \leq & y^{i-1}(V) - \frac{1}{k}y^{i-1}(V) = (1 - \frac{1}{k})y^{i-1}(V). \end{array}$$

We determine inductively that

$$\begin{array}{rcl} y^i(V) & \leq & (1 - \frac{1}{k})^i y^0(V) = (1 - \frac{1}{k})x^*(V) \\ & \leq & (1 - \frac{1}{k})n. \end{array}$$

Hence, we have

$$\begin{array}{rcl} y^k & \leq & (1 - \frac{1}{k})^k n \leq \frac{n}{e}, \\ y^{k \log n} & \leq & e^{-\log n} n \leq 1. \end{array}$$

After at most $k \log n$ iterations have been done, we are very nearly done. It is easy to see that at most k more iterations are needed to reach the integer *last* satisfying $y^{last} = 0$. This means that $last \leq k(\log n + 1)$. Hence, we have

$$\frac{x^*}{\log n + 1} \leq \frac{kx^*}{last} = \frac{1}{last}\sum_{i=1}^{last} x^i.$$

As previously discussed, by an averaging argument, one of these x^i's will cost at least as much as $\frac{c \cdot x^*}{\log n+1}$, which establishes our theorem. ◇

3 Subgraph Isomorphism of Sorted Graphs and The Contact Map Overlap Problem

We are given two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Let $n_i = |V_i|$ and $m_i = |E_i|$, for $i = 1, 2$. The graphs are *sorted* i.e. a total order is defined on $V_1 = \{a_1, \ldots, a_{n_1}\}$ and $V_2 = \{b_1, \ldots, b_{n_2}\}$, so that $a_1 < \ldots < a_{n_1}$ and $b_1 < \ldots < b_{n_2}$. Because of this total ordering, we may identify V_i with $\{1, \ldots, n_i\}$ for $i = 1, 2$, and resort to a_i and b_i notation only when there is a possible confusion for a node i being in V_1 or V_2. It is customary to draw a sorted graph with the vertices arranged increasingly on a line. Although the graph is undirected, we distinguish a tail and a head for each edge $\{i, j\}$, where the tail is the left endpoint (i.e. $\min\{i, j\}$) and the head is the right endpoint (i.e. $\max\{i, j\}$). Therefore, we will denote an edge by an ordered pair (i, j) where i is the tail and j the head of the edge.

A non–crossing map of V_1 in V_2 is defined by 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 1).

Each pair of shared edges contributes a *sharing* to the objective function. The *maximum subgraph isomorphism* for sorted graphs 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 [11, 8], with the additional constraint that the isomorphism of the subgraphs must preserve the ordering of the nodes.

Also, a similar problem is the RNA sequence structure alignment, to which Lenhof, Reinert and Vingron applied and IP approach in [18]. Note that non–crossing maps are in one-to-one correspondence with non–crossing matchings in the complete bipartite graph W having vertex sets V_1 and V_2 and edge set $V_1 \times V_2$. The complete bipartite graph W will be extensively referred to in the arguments to follow.

The problem is largely motivated by its application to computational biology named in the introduction, i.e. the *maximum contact map overlap* problem. A contact map is a graph giving a concise representation of the 3D fold of a protein: for each residue there is a node, and there is an edge (called a *contact*) between two nodes if their euclidean distance is within a given threshold when the protein is folded. The value of an alignment of the residues of one protein vs the other, is the number of contacts in the first contact map whose endpoints are aligned with residues that also share a contact in the second contact map. Since an alignment must preserve the order of the residues, to find the best alignment one has to solve a Maximum Subgraph Isomorphism of Sorted Graphs.

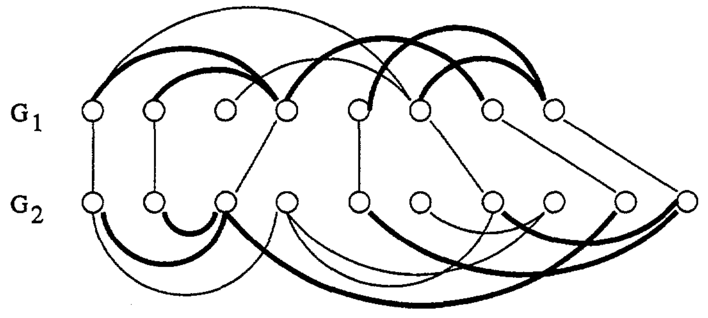

Figure 1: A noncrossing map of value 5

4 Integer Programming 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 then

$$\max \sum_{e \in E_1, f \in E_2} y_{ef} \tag{5}$$

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 do intersect at a single point (or, as we will say, *cross*). Then the constraints for the problem are simply

$$y_{e_1 f_1} + y_{e_2 f_2} \leq 1 \qquad \forall e_1, e_2 \in E_1, f_1, f_2 \in E_2 : \ (e_1, f_1), (e_2, f_2) \text{ are not compatible.} \tag{6}$$

Although it would be possible to list all pairs of incompatible sharings and solve the corresponding IP with this formulation, there are two reasons why we choose not to proceed this way. First, the LP bound is very weak (see the preceding discussion on the independent set problem) unless we strengthen it with cuts, which are not easy to deal with in the space of only y variables. Second, throwing in all the pairs of incompatible sharings may result in too many constraints, which will slow down unacceptably the solving algorithm.

Therefore we decide to 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. Instead of using the predicate PC we then bound the y variables by means of the x variables. That is, the edges (i, j) and (u, v) can be shared only if i is mapped to u and j to v :

$$y_{(i,j)(u,v)} \leq x_{iu} \qquad \text{and} \qquad y_{(i,j)(u,v)} \leq x_{jv} \tag{7}$$

We can immediately strengthen considerably these constraints as follows: For $i \in V_1$ (and analogously for $i \in V_2$), denote by $\delta^+(i)$ the set $\{j \in i+1, \ldots, n_1 : (i,j) \in E_1\}$ of heads of edges of which i is the tail. Similarly, $\delta^-(i) = \{j \in 1, \ldots, i-1 : (j,i) \in E_1\}$ are the tails of edges with head in i. Then, in place of (7), we can write the following constraints:

$$\sum_{j \in \delta^+(i)} y_{(i,j)(u,v)} \leq x_{iu} \quad \text{and} \quad \sum_{j \in \delta^-(i)} y_{(j,i)(u,v)} \leq x_{iv} \quad \forall i \in V_1, (u,v) \in E_2 \tag{8}$$

and of course analogous constraints for $i \in V_2$ and $(u,v) \in E_1$. We call these *activation* constraints.

The *noncrossing* constraints are of the form:

$$x_{iu} + x_{jv} \leq 1 \qquad \forall 1 \leq i \leq j \leq n_1, 1 \leq v \leq u \leq n_2 : i \neq j \vee u \neq v. \tag{9}$$

Noncrossing and activation constraints are sufficient to model completely the problem. Note that we can define a relation of compatibility for the x variables in a similar way as we did for the y. A matching in W can be seen as a set of lines connecting nodes of V_1 and V_2 in the usual drawing of W in which V_1 is drawn on the top and V_2 is drawn 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, and that they *strictly* cross if they cross at a point other than an endpoint. Two lines 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. In the following section we will show that there is a very effective way of finding infeasible sets of lines in W. This sets will correspond to cuts in the IP which will in turn bound the infeasible sets of sharings. There is a well known notion in combinatorial optimization for finding a set of compatible elements, which is the *stable set*, or *independent set* problem.

4.1 The max independent set problems

We define two new 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 noncrossing 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. Hence, all cuts which are valid for the independent set problem can be applied to the x and y variables. The use of these cuts is vital in this problem. Without them, it is easy to build a trivial solution in which all fractional values are 1/2 and which achieves the useless bound of $\min\{m_1, m_2\}$. The most notable cuts for the independent set problem are the clique inequalities: An independent set and a clique can share at most one element. Therefore, we want to determine cliques in G_x and G_y. Another class of inequalities for the max independent set is the odd-holes. We will show that G_x has no odd holes, while G_y may contain them.

In the remainder of the paper, we will describe a very fast method $((O(n^2))$ for separating over the exponentially large $(O(2^{2n}))$ set of all cliques in the x variables. Although we cannot 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 will be so strong that adding the two non-implied classes of cliques for the y variables yields only a tiny improvement in the bound, while incrasing the running time in a way that makes their use dispensable. We will elaborate on this in the section on computational results. For all these reasons, we will focus primarily on cliques in the x variables, i.e. sets of lines in the bipartite graph W which are all mutually crossing. Note that the usual matching constraints are just some such cliques (all lines share an endpoint) :

$$\sum_{u \in V_2} x_{iu} \leq 1 \quad \forall i \in V_1 \quad \text{and} \quad \sum_{i \in V_1} x_{iu} \leq 1 \quad \forall u \in V_2$$

Instead of separating on the $O(n)$ matching constraints, we would like to have them in our basic formulation. However they are not maximal and can be strengthened. We will show that each *maximal* clique in W will correspond to a path which we call *zigzag*. Let ZZ be the set of all zigzag paths and $ZZ(k)$ be the zigzag paths with k internal nodes. The matching constraints are then contained in the set of zigzag paths with only one internal node. $ZZ(2)$ and $ZZ(3)$ will also turn out to be useful, since it appears from our computational experiments that separating over $ZZ(2)$ and $ZZ(3)$ first and only after failure looking at all cliques in ZZ results in a faster branch and bound algorihtm in practice.

4.2 The final IP formulation

Our final IP formulation for the max subgraph isomorphism of sorted graphs is as follows:

$$
\begin{array}{llll}
z & = \max \sum_{e\in E_1, f\in E_2} y_{ef} & & \\
s.t. & & & \\
& \sum_{v\in\delta^+(u)} y_{(i,j)(u,v)} & \le x_{iu} & \forall (i,j)\in E_1, u\in V_2 \\
& \sum_{u\in\delta^-(v)} y_{(i,j)(u,v)} & \le x_{ju} & \forall (i,j)\in E_1, v\in V_2 \\
& \sum_{j\in\delta^+(i)} y_{(i,j)(u,v)} & \le x_{iu} & \forall (u,v)\in E_2, i\in V_1 \\
& \sum_{i\in\delta^-(j)} y_{(i,j)(u,v)} & \le x_{ju} & \forall (u,v)\in E_2, j\in V_1 \\
& \sum_{l\in Q_x} x_l & \le 1 & \forall Q_x \text{ clique in } G_x \\
& \sum_{(e,f)\in Q_y} y_{ef} & \le 1 & \text{for some } Q_y \text{ clique in } G_y \\
& \sum_{(e,f)\in O_y} y_{ef} & \le |O_y|/2 & \forall O_y \text{ odd holes in } G_y \\
& x_{ij}, y_{ef} \in \{0,1\} & & \forall i\in V_1, j\in V_2, e\in E_1, f\in E_2
\end{array}
$$

The remainder of the paper is organized as follows:

5 Cliques in the X variables

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

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

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

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 $O(1)$. $\diamond$

We call the algorithm computing the $O(n^2)$ basic triangles of the proof SETUP. It will be the preliminary step to the fast separation algorithms for the classes of inequalities described later. Note that there are only a polynomial number of triangles ($O(n^3)$) overall so that their value can trivially be computed in polynomial time. However it would cost $O(tn)$ to compute the value of any t triangles in a direct way, so that the lemma gives us an $O(n)$ speed-up factor for $t \geq n^2$, which is going to be the case (e.g. some inequalities will be defined –implicitely– over $O(n^3)$ triangles, and the separation could take time $O(n^4)$). Going from n^2 to n^3 and n^4 makes a *huge* practical difference in running times. We will show that all our separation algorithms have complexity $O(n^2)$, i.e. are very efficient.

5.1 Polynomial Separation of all Maximal Cliques

Call a_1, a_{n_1}, b_1, and b_{n_2} the set of *terminal* nodes. Consider a simple (i.e. without repeated nodes) 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 if 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 the right" (i.e. larger) of a_k. 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 the left" (i.e. smaller) of b_h. Note that any such path must start and end at a terminal node (see Figure 2, left), and must always include the lines $[a_1, b_{n_2}]$ and $[a_{n_1}, b_1]$. Since a_1 and b_{n_2} cannot have both degree two in such a path, there are only two possibilities after we orient the path as described before: Either 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. For example, if $\ldots, a_{h_1}, b_{k_1}, a_{h_2}, b_{k_2}, \ldots$ is part of such a path, then we consider, among others, the triangles $T(a_{h_1}, a_{h_2}|b_{k_1})$ and $T(a_{h_2}|b_{k_2}, b_{k_1})$. Let $T_A(P)$ be the set of triangles defined by P with tip in the nodes of V_1 having degree two in P. Similarly, let $T_B(P)$ be the set of triangles defined by P with tip in nodes of V_2 of degree two in P. We define $T(P)$ as the union of all the triangles defined by P, i.e. $T(P) = T_A(P) \cup T_B(P)$.

Theorem 2 *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 (If) Let Q be a set of lines and P a zigzag path such that $Q = T(P)$. Let $[a_i, b_j]$ and $[a_k, b_h]$ be two lines in $T(P)$. If $[a_i, b_j]$ and $[a_k, b_h]$ are in a same triangle of $T_A(P)$ or $T_B(P)$, then they cross. Otherwise, assume wlog $a_i < a_k$. Depending on the two lines being in a triangle of T_A or T_B, there are four possibilities, which can all be checked similarly. If $[a_i, b_j] \in T_A(P)$ and $[a_k, b_h] \in T_A(P)$, then the path contains $a_i, b, \ldots, a_k$ with $b_j \geq b$ and $b_h \leq b$. Then $b_h \leq b_j$ and so the lines cross.

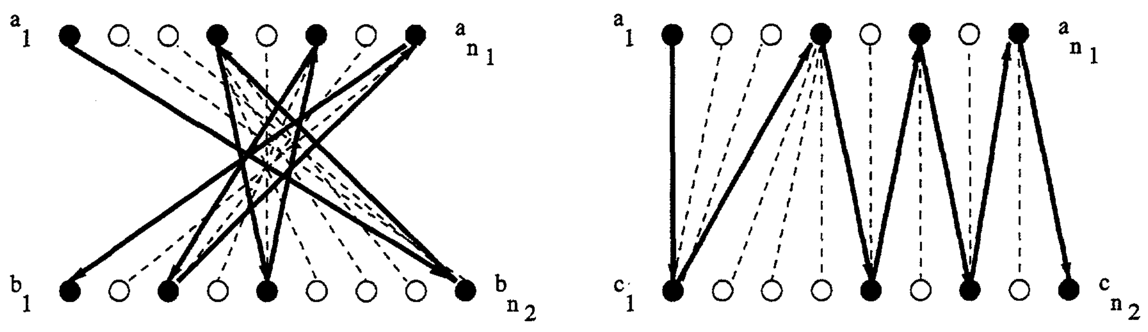

Figure 2: Left: A zigzag path P (bold) and the set $T(P)$. Right: Same path after flipping $V2$.

After checking similarly the remaining cases for being in T_A or T_B, we conclude that $T(P)$ is a clique. To show it is maximal, let $[a, b]$ be any line not in $T(P)$. Either $a > a_1$ or $a < a_{n_2}$. Assume the latter (a similar argument applies in the other case). Let a_k, b_h, a_t be three consecutive nodes of the path such that $a_k \leq a < a_t$. It cannot be $b = b_h$ or else $[a, b] \in T(P)$. Now if $b < b_h$ the lines $[a, b]$ and $[a_t, b_h]$ do not cross, while if $b > b_h$ the lines $[a, b]$ and $[a_k, b_h]$ do not cross. Hence $[a, b]$ cannot be added to $T(P)$ to still obtain a 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. Since Q is maximal, it includes the lines $[a_1, b_{n_2}]$ and $[a_{n_1}, b_1]$, so $a_1^Q = a_1$ and $a_k^Q = a_{n_1}$. For each t, let $b_{l(t)}^Q$ and $b_{r(t)}^Q$ be in $V_2 \cap Q$ such that $[a_t^Q, b_{l(t)}^Q]$ is the leftmost line out of a_t^Q in Q and $[a_t^Q, b_{r(t)}^Q]$ is the rightmost. For Q to be a clique it must be $b_{r(t+1)}^Q \leq b_{l(t)}^Q$ (or else the lines $[a_t^Q, b_{l(t)}^Q]$ and $[a_{r(t+1)}^Q, b_{r(t+1)}^Q]$ do not cross), and to be maximal, it must be in fact $b_{r(t+1)}^Q = b_{l(t)}^Q$ (or else lines from a_{t+1}^Q or a_t^Q to points between $b_{r(t+1)}^Q$ and $b_{l(t)}^Q$ could be added). Now the union of all lines $(a_t^Q, b_{l(t)}^Q)$ and $(a_t^Q, b_{r(t)}^Q)$ defines a zigzag path P, and $Q \subset T(P)$ by construction. But $T(P)$ is a clique, and Q is maximal. So $Q = T(P)$. ◇

The inequalities $x^*(T(P)) \leq 1$ for all zigzag paths P are therefore the strongest clique cuts for this particular maximum independent set problem. We now show that they can be separated in time $O(n^2)$. In order to make the following argument easier, we first 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 (that is, we flip the nodes of V_2 with respect to the usual drawing). Having done this, two lines were compatible (i.e. 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. In Figure 2, right, we show the leftright path of

Figure 2, left, after flipping V_2. A set Q was a clique in the original graph if and only if Q is now a set of triangles in which no two lines are stricIty crossing.

Since to define a leftright path we may pick any k internal (i.e. non terminal) nodes in one graph and k or $k+1$ in the other, there are $O(2^{2n})$ such paths. However, there is an algorithm for finding the leftright path P with largest $x^*(T(P))$ which is not just polynomial, but of very low degree, 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 in the two possible ways and define the length for each arc thus obtained as follows.

$$l(a,c) = x^*(T(a|1,c)) - x^*(T(1,a|c)) \tag{10}$$

and

$$l(c,a) = x^*(T(1,a|c)) - x^*(T(a|1,c)). \tag{11}$$

The lengths of four special arcs are defined separately, as $l(a_1,c_1) = l(c_1,a_1) = 0$, $l(a_{n_1},c_{n_2}) = x^*(T(a_{n_1}|1,c_{n_2}))$ 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 e.g. that $P = (a_1, c_{h_1} = c_1, a_{j_2}, c_{h_2}, \ldots, a_{j_l} = a_{n_1}, c_{h_l} = c_{n_2})$. A similar argument applies in the other three cases. From tedious but simple algebra, we have

$$\begin{aligned}
l(P) &= l(a_1, c_1) + \textstyle\sum_{t=1}^{l-1} l(c_{h_t}, a_{j_{(t+1)}}) + \sum_{t=2}^{l-1} l(a_{j_{(t+1)}}, c_{h_{(t+1)}}) + l(a_{n_1}, c_{n_2}) \\
&= 0 + \textstyle\sum_{t=1}^{i-1} \Big(x^*(T(1, a_{j_{(t+1)}}|c_{h_t})) - x^*(T(a_{j_{(t+1)}}|1, c_{h_t}))\Big) + \\
&\quad \textstyle\sum_{t=2}^{l-1} (x^*(T(a_{j_t}|1, c_{h_t})) - x^*(T(1, a_{j_t}|c_{h_t}))) + x^*(T(a_{n_1}|1, c_{n_2})) \\
&= \textstyle\sum_{t=1}^{l-1} x^*(T(1, a_{j_{(t+1)}}|c_{h_t})) - \sum_{t=1}^{l-1} x^*(T(a_{j_{(t+1)}}|1, c_{h_t})) + \\
&\quad \textstyle\sum_{t=2}^{l-1} x^*(T(a_{j_t}|1, c_{h_t})) - \sum_{t=2}^{l-1} x^*(T(1, a_{j_t}|c_{h_t})) + x^*(T(a_{n_1}|1, c_{n_2})) \\
&= x^*(T(1, a_{j_2}|c_{h_1})) + \textstyle\sum_{t=2}^{l-1} x^*(T(1, a_{j_{(t+1)}}|c_{h_t})) - \sum_{t=1}^{l-2} x^*(T(a_{j_{(t+1)}}|1, c_{h_t})) + \\
&\quad -x^*(T(a_{j_l}|1, c_{h_{(l-1)}})) + \textstyle\sum_{t=2}^{l-1} x^*(T(a_{j_t}|1, c_{h_t})) + \\
&\quad - \textstyle\sum_{t=2}^{l-1} x^*(T(1, a_{j_t}|c_{h_t})) + x^*(T(a_{n_1}|1, c_{n_2})) \\
&= x^*(T(1, a_{j_2}|c_{h_1})) + \textstyle\sum_{t=2}^{l-1} x^*(T(a_{j_t} + 1, a_{j_{(t+1)}}|c_{h_t})) + \\
&\quad \textstyle\sum_{t=2}^{l-1} x^*(T(a_{j_t}|1, c_{h_{(t-1)}} + 1, c_{h_t})) - x^*(T(a_{j_l}|1, c_{h_{(l-1)}})) + x^*(T(a_{n_1}|1, c_{n_2})) \\
&= x^*(T(1, a_{j_2}|c_{h_1})) + \textstyle\sum_{t=2}^{l-1} x^*(T(a_{j_t} + 1, a_{j_{(t+1)}}|c_{h_t})) + \\
&\quad \textstyle\sum_{t=2}^{l-1} x^*(T(a_{j_t}|1, c_{h_{(t-1)}} + 1, c_{h_t})) + x^*(T(a_{n_1}|c_{h_{(l-1)}} + 1, c_{n_2})) \\
&= x^*(T(P))
\end{aligned}$$

The same kind of computations can be carried for the other three possibilities for starting/ending arcs, and are omitted at this point. ◇

Theorem 3 *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 $V(i, j, \searrow)$ 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 $V(i, j, \nearrow)$ 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:

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

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

In figure 3 we draw a directed graph in which each node represents a cell of matrices $V(\cdot, \cdot, \searrow)$ and $V(\cdot, \cdot, \nearrow)$, and it has an arc incoming from each cell on which its value depends. From this drawing it's clear that the only boundary conditions we need are the values of $V(n_1, n_2, \searrow)$ and $V(n_1, n_2, \nearrow)$, which are easily set to $V(n_1, n_2, \searrow) =$

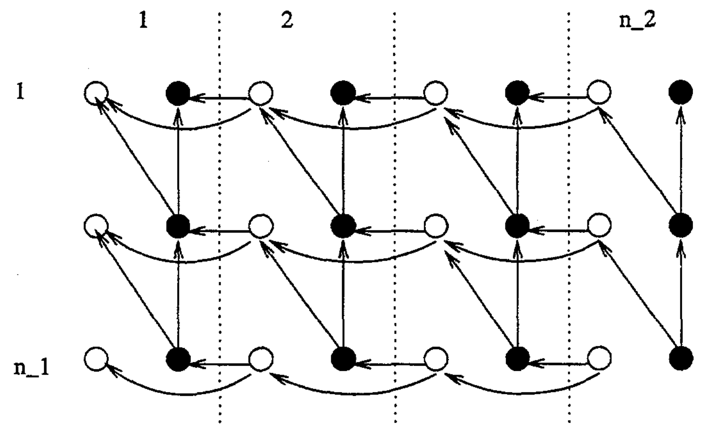

Figure 3: $V(\cdot,\cdot,\searrow)$ white, and $V(\cdot,\cdot,\nearrow)$ black

$l(a_{n_1}, c_{n_2})$ and $V(n_1, n_2, \nearrow) = l(c_{n_2}, a_{n_1})$. The recurrence can be then solved backwards from (n_1, n_2), in time $O(n^2)$. At the end, $V(1, 2, \searrow)$ is the length of the longest leftright path starting with arc (c_1, a_1) and $V(2, 1, \nearrow)$ 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. $\diamond$

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

Proof From theorem 2, 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 3, this concludes the proof. $\diamond$

6 Cliques in the Y variables

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 and R one of A1, A2, A3, B1, B2, C, call $R(e)$ the set of edges which are in the relation R with e (actually, for a fixed edge e, each case leads to two relations, depending on which of the two edges in the drawing is e. To keep notation simple, we will omit this distinction).

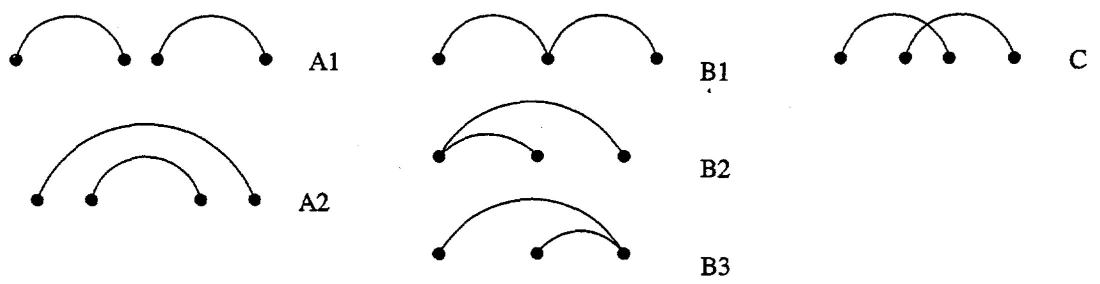

Figure 4: Relationships between edges

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. Hence the following inequality is valid:

$$\sum_{g \notin R(e')} y_{e'g} + y_{ef} \leq 1 \qquad \forall R = A1, A2, A3, B1, B2, C, \qquad e, f \in E_1, f \in R(e)$$

It can be easily proved that the clique inequalities relative to the cases B1, B2, B3 and C are implied by the activation constraints together with the matching constraints for the nodes, and hence cannot be used as cuts in our formulation. The remaining inequalities however are not implied. Consider for instance the case $e = (2,3) \in E_1$, matched with $e' = (2,5) \in E_2$. The edge $f = (1,4) \in E_1$ is in relation A2 with e. However, $f_1 = (1,4) \in E_2$ and $f_2 = (3,6) \in E_2$, are not in the relation A2 with e'. The solution $y_{ee'} = y_{ff_1} = y_{ff_2} = 1/2$ and $x_{11} = x_{13} = x_{22} = x_{35} = x_{44} = x_{46} = 1/2$ is feasible for all the activation and x–clique constraints, but violates the y–clique constraint. It is possible to describe a similar example for the case R=A1. 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. 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 $y_{ef_1} = y_{ef_2} = y_{ee'} = y_{ff_1} = y_{ff_2} = y_{fe'} = 1/4$. This solution is now feasible for all cliques in both x and y variables.

7 On the Strength of the 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 ([14]) states that weakly triangulated graphs are perfect. We show that G_x is perfect by proving the following theorem.

Theorem 4 *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 rigth of l_3.

(*ii*) We now show that G_x has no antiholes of size 5 or more. First make precise how G_x is created. 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 the bipartite graph G_{n_1,n_2}, as discussed before, 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 [15] and Raghunathan [21]. 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. Take for instance $E_1 = \{e_1, \dots, e_5\} = \{(1,8), (2,5), (3,7), (1,4), (6,7)\}$ and $E_2 = \{f_1, \dots, f_5\} = \{(1,7), (2,5), (3,8), (2,4), (5,6)\}$. Consider the sharings $s_i = (e_i, f_i)$ for $i = 1, \dots, 5$. Then it can be checked that $(s_1, s_2, s_3, s_4, s_5, s_1)$ is a chordless cycle in G_y, i.e. each sharing is not compatible with the two adjacent ones but is compatible with anyone else. There is a known polynomial time algorithm for separating odd-holes ([19]), which we used in our code.

8 Computational results

We have implemented our branch and cut in C, and run it on a Pentium PC, with Linux RedHat 6.0, using the branch–and–cut framework ABACUS 2.3. Feasible solutions

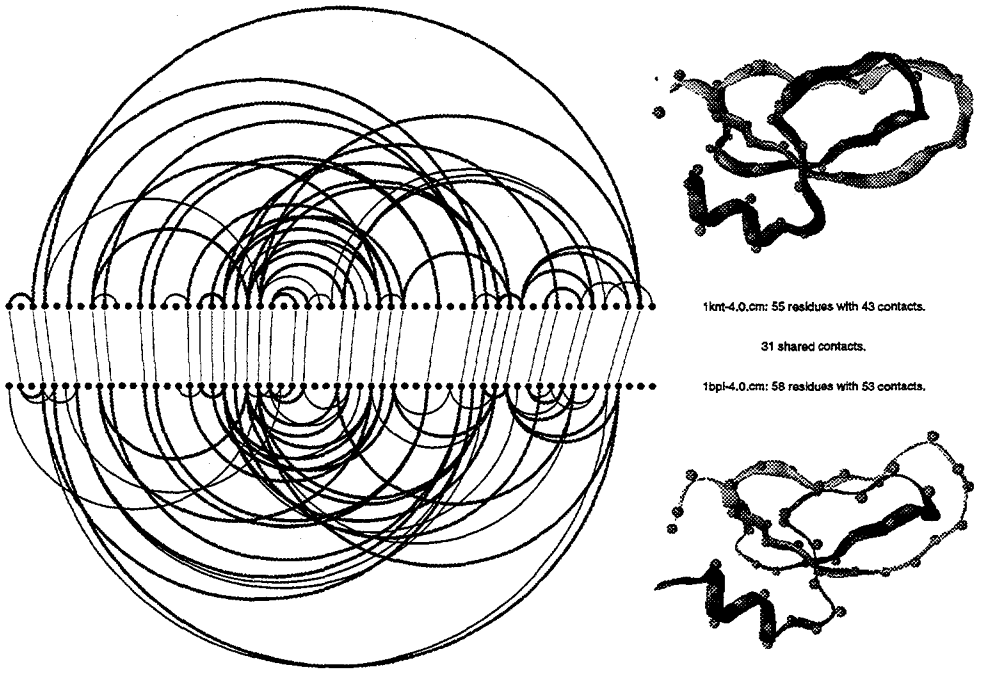
1knt-4.0.cm: 55 residues with 43 contacts.
31 shared contacts.
1bpi-4.0.cm: 58 residues with 53 contacts.

Figure 5: Alignment of the 4Å contact maps of proteins `1bpi` and `1knt`.

were obtained by two greedy local search algorithms (which we do not have space to describe here), run for many iterations. The real data was obtained from the Protein Data Bank (PDB, [4]). The contact maps were computed by using the software `tortilla` which is being developed at Sandia National Labs. Here we report only some preliminary results, which are already very promising. A more complete version of this paper will follow in which we compare all proteins of less than 100 residues versus each other. These comparisons will be used to cluster the proteins in families of similar structure. In Table 1 we report the optimal alignment for ten pairs of proteins from PDB. The contact maps are relative to a 4 Å threshold. Columns NR and NC report the (joint) number of residues and contacts respectively. For each instance we report the size of the starting LP (rows $\times$ columns), the number of cuts generated and an estimate of the –implicit– number of cuts considered, the total number of LPs and the running times in seconds, for the whole algorithm and its LP solver part (which is more than 90% of the total). All these problems were solved to optimality in less than 15 minutes. This is the first time that provably optimal solutions are found for contact map alignment of real proteins. Finally, a nice feature of branch and cut is that, for instances too large to be solved exactly, the procedure can return approximate feasible solutions and a bound on the maximum error. We expect solutions provably close to

proteins	value	NR	NC	start size	cuts	cut space	tot LPs	time
5pti-1bpi	49	116	126	8650×7233	72	781×10^{30}	4	320 (293)
5pti-2knt	36	116	118	7888×6649	554	781×10^{30}	14	760 (743)
5pti-1knt	34	113	116	7585×6329	682	95×10^{30}	22	934 (918)
1bpi-2knt	32	116	98	5988×5749	440	781×10^{30}	12	423 (410)
1bpi-1knt	31	113	96	5757×5469	300	95×10^{30}	7	331 (319)
2knt-1knt	43	113	88	5215×5125	0	95×10^{30}	1	52 (36)
3ebx-6ebx	46	124	122	7490×7556	99	193×10^{33}	4	388 (358)
3ebx-1era	37	124	108	5924×6744	282	193×10^{33}	7	487 (463)
6ebx-1era	39	124	114	6474×7044	230	193×10^{33}	5	467 (445)
1vii-1cph	5	57	28	512×903	437	238×10^{12}	103	12 (11)

Table 1: Optimal alignments for some PDB proteins.

optimum to be already effective in classifying proteins according to their 3D structure.

9 Acknowledgments

Thanks to Alberto Caprara for useful discussions, and to Brian Walenz for his precious help throughout the project. This work was performed while the second author was visiting Sandia National Labs, on a leave from the University of Padua, Department of Electronics and Informatics.

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science,* pages 14-23, 1992.

[2] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 2-13, 1992.

[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, P.E. Bourne, The Protein Data Bank, *Nucleic Acids Research*, 28 pp. 235-242, 2000.

[5] R. B. Boppana and M. M. Halldorsson. Approximating maximum independent sets by excluding subgraphs. *BIT, 32(2):180-196*, 1992.

[6] R. Carraghan and P. M. Pardalos. An exact algorithm for the maximum clique problem, *Operations Research Letters*, 9:375–382, 1990.

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

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

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

[10] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science,* pages 2-13, 1991.

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

[12] 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.

[13] J. Hastad. Fast and efficient testing of the long code. *Proceedings ACM STOC*, 1996.

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

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

[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] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnoy Kan and D. B. Shmoys eds., *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, 1985.

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

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

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

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

Distribution

1. **External Distribution**

   1 Prof. Egon Balas
   GSIA, Carnegie Mellon University
   Pittsburgh, PA 15213-3890

   1 Dr. Alberto Caprara
   DEIS, Università di Bologna, Viale Risorgimento 2
   40136 Bologna, Italy

   1 Prof. Matteo Fischetti
   Dipartimento di Elettronica e Informatica, Via Gradenigo 6/a
   35131 Padova, Italy,

   1 Prof. R. Ravi
   GSIA, Carnegie Mellon University
   Pittsburgh, PA 15213-3890

   1 Dr. Franca Rinaldi
   Dipartimento di Matematica e Informatica, Viale delle Scienze 206
   33100 Udine, Italy

   1 Prof. Giorgio Romanin Jacur
   Dipartimento di Elettronica e Informatica, Via Gradenigo 6/a
   35131 Padova, Italy,

   1 Prof. Paolo Serafini
   Dipartimento di Matematica e Informatica, Viale delle Scienze 206
   33100 Udine, Italy

2. **Internal Distribution**

   10 MS1110 R. Carr, 9211
   10 MS1110 G. Lancia, 9214
   1 MS1110 D. Womble, 9214
   1 MS1110 B. Hart, 9211
   1 MS1110 C. Phillips, 9211
   1 9018 Central Technical Files, 8945-1
   2 0899 Technical Library, 9616
   1 0612 Review and Approval Desk, 9612 for DOE/OSTI