BIOINFORMATICS

Vol. 18 Suppl. 2 2002
Pages S4–S16

Multiple sequence alignment with arbitrary gap costs: Computing an optimal solution using polyhedral combinatorics

Ernst Althaus[1], Alberto Caprara[2], Hans-Peter Lenhof[3] and Knut Reinert[4]

[1]International Computer Science Institute, 1947 Center Street, Berkeley, CA, 94704-1198, USA, [2]DEIS, Universitá di Bologna, Viale Risorgimento 2, Bologna, 40136, Italy, [3]Zentrum für Bioinformatik, Universität des Saarlandes, Im Stadtwald, Saarbrücken, 66123, Germany and [4]Informatics Research, Celera Genomics, 45 West Gude Drive, Rockville, 20850, USA

Received on April 8, 2002; accepted on June 15, 2002

ABSTRACT

Multiple sequence alignment is one of the dominant problems in computational molecular biology. Numerous scoring functions and methods have been proposed, most of which result in NP-hard problems. In this paper we propose for the first time a general formulation for multiple alignment with arbitrary gap-costs based on an integer linear program (ILP). In addition we describe a branch-and-cut algorithm to effectively solve the ILP to optimality. We evaluate the performances of our approach in terms of running time and quality of the alignments using the BAliBase database of reference alignments. The results show that our implementation ranks amongst the best programs developed so far.

Contact: althaus@icsi.berkeley.edu

INTRODUCTION

Aligning DNA or protein sequences is certainly one of the dominant problems in computational molecular biology. The spectrum of methods ranges from extremely fast, while less sensitive, hashing-based methods (e.g. Altschul *et al.* (1990), Delcher *et al.* (1999)) over moderately expensive pairwise comparisons based on dynamic programming (e.g. Gotoh (1982); Smith and Waterman (1981); Needleman and Wunsch (1970)), to costly and sensitive exact multiple alignment formulations (e.g. Gupta *et al.* (1995); Reinert *et al.* (1997, 2000); Lermen and Reinert (2000)), which are based either on the natural extension of the dynamic programming paradigm, or on the application of combinatorial optimization techniques.

The introduction of combinatorial optimization methods to the field of computational biology (Reinert *et al.*, 1997) produced useful tools for numerous applications such as physical mapping (Christof *et al.*, 1997), genome rearrangements (Caprara *et al.*, 1999), RNA secondary structure alignment (Lenhof *et al.*, 1998), optimization of flexible side chains in protein-protein docking (Althaus *et al.*, 2000), computing the fit of three-dimensional structures (Lancia *et al.*, 2001), and a general trace formulation (Kececioglu *et al.*, 2000; Lenhof *et al.*, 1999).

In this paper we extend the formulation of the *gapped trace problem* proposed by Reinert (1999) so that we can formulate a great variety of multiple sequence alignment problems, among which the weighted sum of pairs problem with arbitrary gap costs. To our knowledge this is the first algorithm that can deal with truly affine gap costs (Lipman *et al.* (1989) and Reinert *et al.* (2000) use what is called ‘quasi’-affine or natural gap costs). Indeed our method is independent of the choice of the gap cost function and can handle any function including convex and position-dependent gap costs which were proposed by several authors (Larmore and Schieber (1990); Eppstein (1990) to name a few).

To solve the problem to optimality, we propose a branch-and-cut algorithm and present an implementation of it. Our implementation was evaluated using BAliBase, a benchmark library of structural alignments (Thompson *et al.*, 1999). While our initial implementation occasionally failed to solve large examples in the specified time and memory constraints, it was often the best or among the tested programs. More precisely, for moderately sized instances we compute on average the best solutions. This positively answers a repeatedly asked question, whether it is worthwhile to compute optimal alignments using scoring functions which admittedly only approximate the true biological phenomena.

In the first section we present a graph-theoretic formulation for our problem, which is translated into an integer linear program (ILP). In the following section, we study the structure of the *gapped trace polytope*, namely the convex hull of the incidence vectors of the ILP solutions. The study of the polytope is a prerequisite for the algorithm we propose, which fits into the branch-and-cut framework. We present some classes of valid inequalities and describe how we reduce the number of variables in our ILP. Then we evaluate the quality of our alignment algorithm and finally discuss our results in the last section.

For the technically interested reader we also give the details of the branch-and-cut algorithm, specifically the separation routines in the Appendix. There we also give conditions under which the presented valid inequalities define facets of the problem polytope.

A GRAPH THEORETIC MODEL

The gapped trace problem can be formulated as follows. Let $S = \{s^1, s^2, \ldots, s^k\}$ be a set of k strings over an alphabet Σ and let $\bar{\Sigma} = \Sigma \cup \{-\}$, where '−' (dash) is a symbol to represent 'gaps' in strings. An *alignment* of S is a set $\bar{S} = \{\bar{s}^1, \bar{s}^2, \cdots, \bar{s}^k\}$ of strings over the alphabet $\bar{\Sigma}$ that satisfies the following two properties: (1) the strings in $\bar{S}$ have all the same length, and (2) ignoring dashes, string $\bar{s}^i$ is identical to string s^i. An alignment in which each string $\bar{s}^i$ has length l can be interpreted as an array of k rows and l columns where row i corresponds to string $\bar{s}^i$. Two characters of distinct strings in S are said to be *aligned* under $\bar{S}$ if they are placed into the same column of the alignment array.

In the following we will let $n := \sum_{i=1}^{k} \|s^i\|$. In order to express our problem on a graph we need the notion of a *mixed graph* which is a tuple $G = (V, E, A)$, where V is a set of vertices, E is a set of (undirected) edges and A is a set of (directed) arcs. A *path* in a mixed graph is an alternating sequence $v_1, e_1, v_2, e_2, \ldots, v_k$ of vertices and arcs or edges such that either $e_i = \{v_i, v_{i+1}\} \in E$ or $e_i = (v_i, v_{i+1}) \in A$, for all i, $1 \le i \le k$. A path is called a *mixed path* if it contains at least one arc in A and one edge in E. A mixed path is called a *mixed cycle* if the first and the last vertex on the path coincide. A mixed cycle represents an ordering conflict of the letters in the sequences. The most trivial mixed cycle would correspond to two alignment edges 'crossing', which is not allowed. Since a mixed path P (or a mixed cycle C) is determined by the set of arcs and edges in P (respectively in C), we often identify paths and cycles by their set of edges and arcs. We do allow multiple edges and arcs in between any two nodes, or to put it differently a mixed graph is a multigraph.

We can view the character positions of the k input strings in S as the vertex set V of a mixed graph $G = (V, E, A)$, called the *gapped alignment graph*, where each node v^i_j represents the character s^i_j. For convenience we denote by V^i the set of all nodes corresponding to characters in s^i, i.e. node set V is partitioned into $V^1, \ldots, V^k$. Additionally we need the notion of a *critical* cycle. A cycle C is called critical if for all i, $1 \le i \le k$, all vertices in $V^i \cap C$ occur consecutively in C.

The edges in E represent *alignments* of pairs of characters in different strings. Namely, we say that an edge $e = \{u, v\}$ is *realized* by an alignment if the endpoints of the edge are placed into the same column of the alignment array. Let $E^{i,j} \subseteq E$ denote the set of all edges with one endpoint in V^i and the other in V^j. Note that the graph obtained from G by removing all its arcs is k-partite with color classes $V^1, \ldots, V^k$.

The arcs in $A = A_g \cup A_p$ represent positional constraints. Arcs in A_p represent *consecutivity* of characters within a same string and run from each node to its 'right' neighbor, i.e. $A_p = \{(v^i_j, v^i_{j+1}) : 1 \le i \le k, 1 \le j < \|s^i\|\}$. The arcs in A_g represent *gaps* in the alignment. Each substring of a string s^i can be aligned with gap characters in any other string s^j, or to put it differently, it may be the case that no character in this substring of s^i is aligned with any character in s^j. Hence we introduce for each substring of s^i from s^i_l to s^i_m and for each $1 \le j \le k$, $j \ne i$, an arc from v^i_l to v^i_m, denoted by $(v^i_l, v^i_m)^j$. In other words, there are $k - 1$ arcs in A_g from v^i_l to v^i_m. We again say that a gap arc $(v^i_l, v^i_m)^j$ is *realized* by an alignment if the substring in s^i from position l to position m is not aligned to any character in s^j, whereas both s^i_{l-1} (if $l > 1$) and s^i_{m+1} (if $m < \|s^i\|$) are aligned with some letter in s^j. We also say that the nodes corresponding to the substring of s^j are *spanned* by the gap arc. Let $A^{i,j} \subseteq A_g$ denote the set of all gap arcs for substrings in s^i aligned with gap characters in s^j. Also, given two gap arcs $(v^i_l, v^i_m)^j$, $(v^i_p, v^i_q)^j \in A^{i,j}$, we say that they *conflict* if the substrings spanned by the arcs overlap or even touch, that is if $[l, m+1] \cap [p, q] \ne \emptyset$. The fact that the two arcs conflict even if $p = m+1$ is due to the above definition of realization – informally, there must be at least one aligned character between consecutive gap arcs. Finally, we let

$$A^{i,j}(l \leftrightarrow m) := \{(v^i_p, v^i_q)^j : p \le l, q \ge m\}$$

denote the set of arcs in $A^{i,j}$ spanning $v^i_l, \ldots, v^i_m$. We also allow $l = m+1$, i.e. use $A^{i,j}(p+1 \leftrightarrow p)$ to denote the set of arcs that span either v^i_p or v^i_{p+1}. The latter is motivated by the necessity of representing sets of conflicting arcs.

In order to score the alignment, each of the edges in E and gap arcs in A_g is assigned a *weight* that corresponds to the benefit (or cost) of realizing the edge or arc. We let w_e and w_a denote respectively the weight of edge $e \in E$ and arc $a \in A_g$. Note that arcs A_p are independent of the alignment, which specifies which edges among E and arcs among A_g are realized.

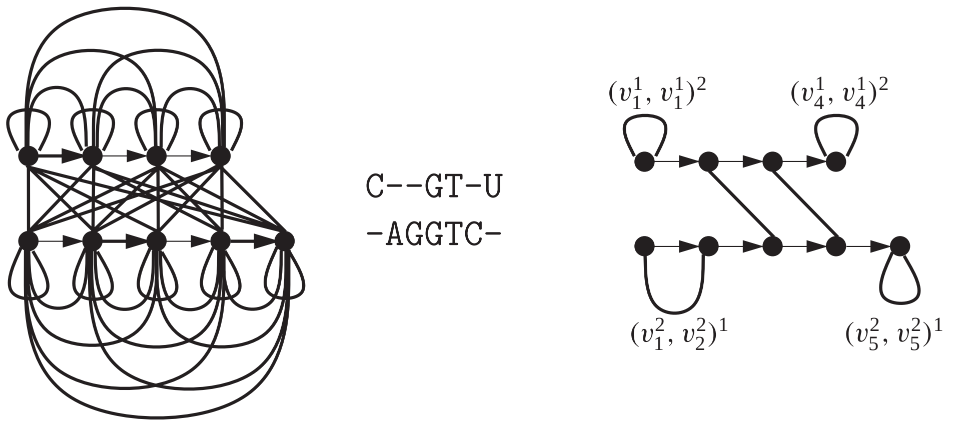

Fig. 1. Gapped alignment graph for two sequences. In the middle is an alignment that realizes the gapped trace on the right.

A subgraph of the gapped alignment graph is called *gapped trace* if it corresponds to a gapped alignment. A gapped trace has to fulfill the following conditions which we will formalize in the mathematical model of the next section:

1. For each pair of strings, each node is either incident to exactly one alignment edge or spanned by exactly one gap arc.

2. There must not be a critical mixed cycle in the subgraph. (see Reinert *et al.* (1997) for a proof).

3. There cannot be a pair of conflicting gap arcs for a given pair of strings.

4. Whenever we realize two edges incident with the same node, say $\{v_{l_1}^{i_1}, v_{l_2}^{i_2}\}$ and $\{v_{l_1}^{i_1}, v_{l_3}^{i_3}\}$, by transitivity we must also realize edge $\{v_{l_2}^{i_2}, v_{l_3}^{i_3}\}$.

The goal is to identify the gapped trace which has the highest weight given a suitable scoring scheme (see Figure 1 for an example of a gapped alignment graph and an associated gapped trace).

Usually one assigns to each edge $e \in E$ the corresponding cost derived from an amino acid substitution table (e.g. Henikoff and Henikoff (1992); Dayhoff *et al.* (1979)) and to each arc $a \in A_g$ an affine or convex cost function. However our method is capable of handling arbitrary, even position-dependent, costs for both the edges in E and the arcs in A_g. This is certainly in contrast to algorithms presented in (Lipman *et al.*, 1989; Reinert *et al.*, 2000). They cannot compute alignments with truly affine gap costs, which are currently the method of choice in most pairwise alignment methods.

AN ILP FORMULATION

We assume in the following that $\|s^i\| \geq 3$ for all strings s^i and $k \geq 2$.

An initial model

We have two types of variables:

- For every edge $e = \{v_l^i, v_m^j\} \in E^{i,j}$, we define a binary variable x_e (we also write $x_{\{v_l^i, v_m^j\}}$), which indicates whether s_l^i is aligned with s_m^j or not. We call these variables the *alignment variables*.
- For every arc $a = (v_l^i, v_m^i)^j \in A^{i,j}$, representing a gap in string s^j aligned to the substring $s_{l \leftrightarrow m}^i$ of s^i, we define a binary variable y_a (we also write $y_{(v_l^i, v_m^i)^j}$). We call these variables the *gap variables*.

For a cycle C, we denote the set of edges with C_E and the set of arcs with C_A. Let $\mathcal{C}$ denote the (exponentially large) collection of all critical mixed cycles in G and $\mathcal{I}$ the collection of all maximal sets of conflicting gap arcs. It is not difficult to show (see also the next section) that

$$\mathcal{I} = \{A^{i,j}(l+1 \leftrightarrow l) : 1 \leq i, j \leq k, i \neq j, 1 \leq l < |s^i|\}.$$

Our initial ILP formulation is given in Figure 2.

Constraints (2), (3), (4) and (5) correspond to requirements 1, 2, 3 and 4, respectively. Note that inequalities (4), which prevent the realization of ‘touching’ gap arcs, are not necessary in the case of *convex* gap costs. Note also that transitivity inequalities involving four or more strings, e.g.

$$x_{\{v_{l_1}^{i_1}, v_{l_2}^{i_2}\}} + x_{\{v_{l_2}^{i_2}, v_{l_3}^{i_3}\}} + x_{\{v_{l_3}^{i_3}, v_{l_4}^{i_4}\}} - x_{\{v_{l_1}^{i_1}, v_{l_4}^{i_4}\}} \leq 2,$$

are implied by (5).

By the above discussion we have

$$\max \sum_{e\in E} w_e \cdot x_e + \sum_{a \in A_g} w_a \cdot y_a \tag{1}$$

subject to

$$\sum_{1\le m\le |s^j|} x_{\{v_l^i, v_m^j\}} + \sum_{a\in A^{i,j}(l\leftrightarrow l)} y_a = 1,\ \forall\, 1 \le i, j \le k, i \ne j, 1 \le l \le |s^i| \tag{2}$$

$$\sum_{e\in C_E} x_e \le |C_E| - 1,\ \forall\, C \in \mathcal{C} \tag{3}$$

$$\sum_{a\in I} y_a \le 1,\ \forall\, I \in \mathcal{I} \tag{4}$$

$$x_{\{v_{l_1}^{i_1}, v_{l_2}^{i_2}\}} + x_{\{v_{l_1}^{i_1}, v_{l_3}^{i_3}\}} - x_{\{v_{l_2}^{i_2}, v_{l_3}^{i_3}\}} \le 1,\ \forall\, 1 \le i_r \le k, 1 \le l_r \le \|s^{i_r}\| (r = 1, 2, 3) \tag{5}$$

$$x_e, y_a \in \{0, 1\},\ \forall\, e \in E, a \in A_g \tag{6}$$

Fig. 2. Integer linear program describing the gapped trace polytope.

PROPOSITION 1. *Every gapped alignment corresponds to a feasible solution of ILP (1)–(6) and vice versa.*

We will use branch-and-cut techniques to solve this ILP which can be shortly explained as follows: First define the *gapped trace polytope* $\mathcal{P}$ of a gapped alignment graph G as the convex hull of all incidence vectors of gapped traces, i.e.

$$\mathcal{P} := conv\{\chi^X \in \{0,1\}^{|E\cup A|} \mid X \subseteq E \cup A \\ \text{is a gapped trace of } G\}.$$

It is well known that the optimal ILP solution corresponds to a vertex of this polytope. Unfortunately solving an ILP directly is NP-hard. Hence we relax the given integer program by dropping the integer conditions, that means we replace (6) by

$$x_e, y_a \ge 0,\ \forall\, e \in E, a \in A_g \tag{7}$$

(Note that $x_e, y_a \le 1$ is implied by (in)equalities (2) and (4)). Then we solve the resulting linear program.

If the solution $\bar{x}$ of the linear program is integral we have the optimal solution. Otherwise we search for a valid inequality $fx \le f_0$ that 'cuts off' the solution $\bar{x}$, i.e. $fy \le f_0$ for all $y \in \mathcal{P}$ and $f\bar{x} > f_0$; the set $\{x \mid fx = f_0\}$ is called a *cutting plane*. The search for a cutting plane is called the *separation* problem. Any cutting plane found is added to the linear program and the linear program is resolved. The generation of cutting planes is repeated until either an optimal solution is found or the search for a cutting plane fails. In the second case a branch step follows: We generate two subproblems by setting one fractional variable x_e to 0 in the first subproblem and to 1 in the second subproblem and solve these subproblems recursively. This gives rise to an enumeration tree of subproblems.

It would be desirable that the LP relaxation be 'close' to the original problem. Geometrically, one would like the polytope defined by (2)–(5) and (7) to be 'close' to $\mathcal{P}$. In fact, one may be interested in determining better LP relaxations. The study of the structure of $\mathcal{P}$, in particular the determination of linear inequalities which are satisfied by all points in $\mathcal{P}$, yields useful information in this direction. In the next section, we illustrate valid inequalities for $\mathcal{P}$ that are extensions of those in the initial ILP formulation. In the Appendix we prove that they define *facets* of $\mathcal{P}$, i.e. they are as strong as possible, and we describe efficient separation procedures for each class.

Valid inequalities

In order to define valid inequalities for $\mathcal{P}$, we characterize pairs of edges and/or arcs that are *incompatible*, i.e. the associated variables cannot both take the value 1 in a feasible solution. All these incompatibilities follow immediately from requirements 1 and 3 and the corresponding constraints (2) and (4) in the ILP formulation.

Two alignment edges are incompatible if and only if they are *crossing*, i.e. each edge $\{v_l^i, v_m^j\}$ is incompatible with all edges $\{v_p^i, v_q^j\}$ such that either $p \le l$ and $q \ge m$ or $p \ge l$ and $q \le m$.

Two gap arcs are incompatible if and only if they *overlap* or *touch*, i.e. each arc $(v_l^i, v_m^i)^j$ is incompatible with all arcs $(v_p^i, v_q^i)^j$ such that $p \le m+1$ and $l \le q+1$.

An alignment edge and a gap arc are incompatible if and only if one endpoint of the former is spanned by the latter, i.e. each edge $\{v_l^i, v_m^j\}$ is incompatible with all arcs $(v_p^i, v_q^i)^j$ such that $p \leq l \leq q$, as well as all arcs $(v_p^j, v_q^j)^i$ such that $p \leq m \leq q$.

In particular, note that two variables may be incompatible only if they are associated with the same (ordered) pair of strings.

Clique inequalities We call *clique* a maximal set K of pairwise incompatible edges and arcs. Moreover, we denote by K_E and K_A the edges and arcs in K, respectively. The corresponding *clique inequality* has the form

$$\sum_{e \in K_E} x_e + \sum_{a \in K_A} y_a \leq 1, \tag{8}$$

and is clearly valid for $\mathcal{P}$. Note that (4), as well as (2) if '=' is replaced by '≤', are examples of clique inequalities.

Lifted mixed cycle inequalities In this section, we illustrate a class of inequalities which are stronger than the original (3).

As their 'weak' version (3), these inequalities involve only alignment variables. Consider a sequence of strings $s^{i_1}, \ldots, s^{i_t}$, along with edge set $C \subseteq E$, which is partitioned into edge sets $C^{i_r, i_{r+1}} \subseteq E^{i_r, i_{r+1}}$, $r = 1, \ldots, t$ (letting $i_{t+1} := i_1$). If C meets the following requirements

(a) for $r = 1, \ldots, t$, all edges in $C^{i_r, i_{r+1}}$ are pairwise incompatible;

(b) every set $\{e_1, \ldots, e_r\}$, where each e_r is chosen arbitrarily from $C^{i_r, i_{r+1}}$, is the set of edges in a critical mixed cycle;

then the following inequality

$$\sum_{e \in C} x_e \leq t - 1, \tag{9}$$

is valid for $\mathcal{P}$. Note that (3) are a special case of (9) in which each set $C^{i_r, i_{r+1}}$ contains only one edge. If in addition

(c) C is *maximal* with respect to properties (a) and (b);

we call (9) a *lifted mixed cycle inequality*.

Generalized transitivity inequalities The last class of inequalities that we present is a generalization of the transitivity inequalities (5).

Consider three different strings $s^{i_1}, s^{i_2}, s^{i_3}$, along with a letter $s_{l_1}^{i_1}$ of string s^{i_1} and subsets S^2 and S^3 of letters in s^{i_2} and s^{i_3}, respectively. Clearly, if we align $s_{l_1}^{i_1}$ with a letter in S^2 as well as with a letter in S^3, then we must align a letter in S^2 with a letter in S^3. This yields the valid inequality

$$\sum_{s_{l_2}^{i_2} \in S^2} x_{\{v_{l_1}^{i_1}, v_{l_2}^{i_2}\}} + \sum_{s_{l_3}^{i_3} \in S^3} x_{\{v_{l_1}^{i_1}, v_{l_3}^{i_3}\}} - \sum_{s_{l_2}^{i_2} \in S^2} \sum_{s_{l_3}^{i_3} \in S^3} x_{\{v_{l_2}^{i_2}, v_{l_3}^{i_3}\}} \leq 1, \tag{10}$$

which coincides with one of (5) if $|S^2| = |S^3| = 1$.

Variable reduction

Even for rather small instances, the number of variables in our ILP formulation is large. For example, an instance with 5 strings of length 100 has 201 000 variables. Most of these variables are very unlikely to take the value 1 in an optimal solution. In this section, we describe a successful method to reduce the number of variables by eliminating those variables that can not appear in an optimal solution.

Assume we know a good lower bound L on the optimum, e.g. found by a heuristic. For every variable v, we compute an upper bound U_v on the value of the optimal alignment in which variable v takes the value 1. If $U_v \leq L$, we know that this variable is not used by any alignment which is better than the one we already have, so we can permanently fix the variable to 0, removing it.

A simple upper bound U_v for an alignment variable v can be computed as follows: Compute all optimal alignments between two strings s^i, s^j, $1 \leq i < j \leq k$. The sum of the values of all these alignments is clearly an upper bound on the value of the optimal alignment, called the *pairwise upper bound*. Moreover, for any alignment variable corresponding to an edge between s^i and s^j we can compute an upper bound on the value of the optimal alignment that realizes the associated edge, by computing the optimal alignment between s^i and s^j that realizes this edge and adding to the resulting value the values of the optimal alignments between the other string pairs. We can compute this upper bound for all alignment variables by finding only two pairwise optimal alignments (see Gusfield (1997)), with a total running time of $O(n^2)$.

For a gap variable corresponding to arc $(v_l^i, v_k^i)^j$, we can compute an upper bound on the value of the optimal alignment that realizes this gap by iterating over all nodes v_h^j of string j and assuming that v_{l-1}^i is aligned with v_h^j and that v_{k+1}^i is aligned with some letter after v_h^j. The total running time to compute all these upper bounds is $O(n^3)$.

Finally, we can compute a better upper bound for alignment variables with the following idea. Assume we fix edge $\{v_l^i, v_m^j\}$ in the solution. For every string h there must be a node v_k^h so that every letter before s_k^h must be aligned with a letter before s_l^i in string i as well as with a letter before s_m^j in string j, and every letter after s_{k+1}^h must be aligned with a letter after s^i_{l+1} in string i as well as with a letter after s^j_{m+1} in string j. The total running time to compute all these upper bounds is $O(n^4)$.

COMPUTATIONAL EXPERIMENTS

Goal

The goal of our experiments is twofold. First, we want to assess the quality of *optimal* alignments computed with our algorithm. Second, we want to explore how practical the current implementation is in terms of number of sequences and sequence length, where each of these factors will increase the size of the ILP and hence makes it potentially harder to solve. Our comparison with other programs serves these two goals. It reveals that computing optimal alignments does pay off in terms of quality and it explores the current limits for feasibility of our approach. It should not be understood as a general ranking of these programs, since, although we perform very well on instances we can solve, we are still not able to solve larger problems.

Test data

We tested our implementation, which we will call COSA (COmbinatorial Sequence Alignment) using a database of different benchmark alignments (Thompson *et al.*, 1999) containing groups of sequences of different lengths, called Reference 1 to 5, or (for short) R1 to R5. R1, R2 and R3 are subdivided into three groups of different lengths. R1 is further subdivided into three subgroups according to different identity levels (V1: identity $<$ 25%, V2: identity 20–40%, and V3: identity $>$ 35%). (R4 and R5 are not subdivided.) The database also offers an evaluation program which computes a score between 0 and 1 indicating the percentage of correctly aligned residues in the core regions of the alignments, where the reference alignments are hand-created using structural information. Note that we use the newest version of the evaluation program (from June 14, 2002) for which no bug is known. In addition to the database Thompson *et al.* (1999) also published a survey of the qualitative performance of different alignment programs available at that time.

Recently BAliBase 2 was made available which contains updated reference alignments and more test sets. Since we wanted to use the most recent structural alignments as standard of truth we decided to use the overall best performing programs from Thompson's survey, PRRP, ClustalX, and Dialign together with a recently published program T-Coffee (Notredame *et al.*, 2000) which generally outperforms the other programs. The source code of all these programs was downloaded and they were run with default parameters. We disregard all other programs, since they perform significantly worse (compare `http://www-igbmc.u-strasbg.fr/BioInfo/BAliBASE/prog_scores.html`).

Table 1. Comparison of different scoring functions. The table shows the average scores of the overall best general and affine function. For both functions, we show the score with and without a penalty on end gaps

Group	$8 + 2l + 2\sqrt{l}$		$6 + 4l$		$8 + 6\sqrt{l}$	
R1 V1	0.552	0.556	0.513	0.524	0.540	0.612
R1 V2	0.848	0.879	0.846	0.851	0.766	0.843
R1 V3	0.974	0.978	0.978	0.981	0.957	0.981

Setup for COSA

We implemented the above algorithms using the branch-and-cut framework SCIL (SCIL, 2002). The input variable set is constructed as follows: We used the variable reduction as described earlier with the pairwise upper bound minus 20 as lower bound. This was necessary to reduce the run time and memory consumptions. Note that this value is not a valid lower bound in most cases. That means that we might lose variables in the optimal solution and with that the optimal solution itself (For reference 1 we checked our computed solutions and can prove that in most cases they are optimal, for other references we find suboptimal or no solutions, i.e. they are in practice infeasible).

To evaluate our implementation we proceeded as follows: In order to find a good scoring function we computed the optimal alignment for all short instances of the group R1 for different affine and convex gap cost functions with and without end gap penalties. As a amino acid substitution matrix we used `blosum62`. We also experimented with `blosum30`, `blosum90` and `pam250`, which gave worse results. For each group of instances in R1 we computed the average score for different objective functions each with and without end-gap penalty (see Table 1). It is interesting to note that $8 + 2l + 2\sqrt{l}$, a non linear function, was the best objective function. Hence we used it to compute the optimal alignments in the other groups. We ran the experiments on a Sparc Ultra-Enterprise-10000 (333 MHz) with 15 GB main memory. We allowed the programs a running time of at most 10 hours and memory of at most 4 GB. Any run that exceeded these limits was counted as unsuccessful.

Results

Tables 2, 3, and 4 contain the results for some of the reference sets. The first column contains the name of the data set and the subsequent columns the results for the 5 alignment programs. Each entry in the tables is the score computed by the evaluation program provided with the

Table 2. Comparison of the Core scores for COSA, TCOFFE and other programs. In the first column we give the name of the instance and its size (in number of strings/total number of characters)

Data	COSA	TCOFFEE	PRRP	CLUSTALX	DIALIGN
Reference 1 short V1					
1aboA (5/297)	0.558 (89)	0.509 (81)	0.212 (34)	0.63 (100)	0.624 (99)
1idy (5/269)	0.646 (95)	0.065 (10)	0.681 (100)	0.415 (61)	0.031 (5)
1r69 (4/277)	0.493 (100)	0.28 (57)	0.427 (87)	0.48 (97)	0.107 (22)
1tvxA (4/242)	0.158 (72)	0.219 (100)	0.132 (60)	0.105 (48)	0 (0)
1ubi (4/327)	0.46 (61)	0.447 (59)	0.46 (61)	0.76 (100)	0.06 (8)
1wit (5/484)	0.854 (100)	0.792 (93)	0.833 (98)	0.721 (84)	0.417 (49)
2trx (4/362)	0.723 (98)	0.587 (80)	0.364 (50)	0.705 (96)	0.735 (100)
avg.	0.556 (100)	0.414 (74)	0.444 (80)	0.545 (98)	0.282 (51)
Reference 1 short V2					
1aab (4/291)	0.929 (100)	0.929 (100)	0.929 (100)	0.869 (94)	0.929 (100)
1fjlA (6/398)	1 (100)	0.993 (99)	1 (100)	1 (100)	1 (100)
1hfh (5/606)	0.958 (100)	0.922 (96)	0.869 (91)	0.694 (72)	0.259 (27)
1hpi (4/293)	0.841 (99)	0.727 (85)	0.852 (100)	0.841 (99)	0.545 (64)
1csy (5/510)	0.975 (100)	0.953 (98)	0.946 (97)	0.907 (93)	0.865 (89)
1pfc (5/560)	0.892 (94)	0.944 (100)	0.861 (91)	0.826 (88)	0.438 (46)
1tgxA (4/239)	0.641 (69)	0.7 (75)	0.865 (93)	0.929 (100)	0.459 (49)
1ycc (4/426)	0.979 (99)	0.921 (93)	0.958 (97)	0.991 (100)	0.636 (64)
3cyr (4/414)	0.77 (100)	0.713 (93)	0.713 (93)	0.751 (98)	0.355 (46)
451c (5/400)	0.801 (100)	0.713 (89)	0.618 (77)	0.646 (81)	0.571 (71)
avg.	0.879 (100)	0.852 (97)	0.861 (98)	0.845 (96)	0.606 (69)
Reference 1 short V3					
1aho (5/320)	1 (100)	1 (100)	0.937 (94)	0.857 (86)	0.943 (94)
1csp (5/339)	0.993 (100)	0.993 (100)	0.967 (97)	0.993 (100)	0.967 (97)
1dox (4/374)	0.918 (100)	0.918 (100)	0.922 (100)	0.911 (99)	0.848 (92)
1fkj (5/517)	0.987 (100)	0.987 (100)	0.92 (93)	0.917 (93)	0.895 (91)
1fmb (4/400)	0.978 (100)	0.978 (100)	0.967 (99)	0.978 (100)	0.961 (98)
1krn (5/390)	1 (100)	1 (100)	1 (100)	0.992 (99)	0.84 (84)
1plc (5/470)	0.947 (97)	0.947 (97)	0.976 (100)	0.935 (96)	0.838 (86)
2fxb (5/287)	0.963 (98)	0.963 (98)	0.963 (98)	0.981 (100)	0.963 (98)
2mhr (5/572)	1 (100)	0.996 (100)	1 (100)	1 (100)	0.922 (92)
9rnt (5/499)	0.99 (99)	0.995 (100)	0.977 (98)	0.995 (100)	0.854 (86)
avg.	0.978 (100)	0.978 (100)	0.963 (98)	0.956 (98)	0.903 (92)

Table 3. Comparison of the Core scores for COSA, TCOFFE and other programs. In the first column we give the name of the instance and its size (in number of strings/total number of characters)

Data	COSA	TCOFFEE	PRRP	CLUSTALX	DIALIGN
Reference 2 (5 of 9 instances)					
1aboA (16/945)	0.634 (74)	0.837 (98)	0.809 (95)	0.853 (100)	0.77 (90)
1csy (19/1581)	0.555 (62)	0.882 (99)	0.893 (100)	0.877 (98)	0.859 (96)
1tgxA (20/1246)	0.231 (24)	0.934 (99)	0.915 (97)	0.944 (100)	0.831 (88)
1tvxA (19/1056)	0.711 (73)	0.931 (96)	0.94 (97)	0.97 (100)	0.855 (88)
1ubi (17/1546)	0.484 (49)	0.88 (89)	0.925 (94)	0.984 (100)	0.842 (86)
avg.	0.523 (56)	0.893 (96)	0.896 (97)	0.926 (100)	0.831 (90)

Table 4. Comparison of the Core scores for COSA, TCOFFE and other programs. In the first column we give the name of the instance and its size (in number of strings/total number of characters)

Data	COSA	TCOFFEE	PRRP	CLUSTALX	DIALIGN
		Reference 4 (10 of 12 instances)			
1dynA (6/1468)	0 (0)	0.1 (50)	0 (0)	0 (0)	0.2 (100)
1ckaA (10/1758)	0 (0)	1 (100)	0.75 (75)	0.75 (75)	0 (0)
1csp (6/1405)	0 (0)	0 (0)	0 (0)	0 (0)	0.11 (100)
1lkl (8/2007)	1 (100)	0.9 (90)	1 (100)	0.9 (90)	0.9 (90)
1mfa (7/1735)	0 (0)	0.38 (83)	0.46 (100)	0 (0)	0.38 (83)
1pfc (10/2315)	1 (100)	0.25 (25)	0.15 (15)	0.34 (34)	0 (0)
1vln (14/1195)	0 (0)	0.57 (79)	0 (0)	0.72 (100)	0.48 (67)
1ycc (9/1095)	0.485 (77)	0.48 (76)	0.48 (76)	0.63 (100)	0.45 (71)
2abk (6/1547)	0 (0)	0.58 (100)	0 (0)	0 (0)	0.47 (81)
kinase1 (6/1547)	0 (0)	0.89 (100)	0 (0)	0.06 (7)	0 (0)
avg.	0.249 (48)	0.515 (100)	0.284 (55)	0.34 (66)	0.299 (58)

database. In brackets we give the percentage relative to the best performing program for this example. In addition we provide an average for each group.

In Table 2 we give the values for all short sequences of reference set 1. Our implementation is on average superior to all other programs in each of these data sets. This indicates that it pays to compute the optimal solution, because our approach produced on average the biologically most meaningful alignments.

Tables 3 and 4 exhibit the current practical limitations of our approach. For reference 2 we are able to solve only 5 of 9 instances where the average quality is at least acceptable, while for reference 4 we can solve 10 of 12 instances with a miserable performance. This can be attributed to the fact that we have not enough variables in our initial ILP, but of course that would not be a guarantee for a good performance (TCoffee and Dialign score in this set extremely well, since they accommodate local alignment information in contrast to PRRP, ClustalX and COSA).

In summary, these experiments back out the claim that optimal sequence alignment is able to find subtle biological signals and that the current implementation is capable of solving real world problems. However some problem sizes are certainly still out of reach and better addressed by other programs.

CONCLUSION

We have presented for the first time a general multiple sequence alignment formulation with arbitrary gap costs. The graph-theoretic formulation allows us to define an ILP model which is valid for any choice of the gap penalty function. The ILP model is solved by branch-and-cut. Our implementation is competitive with or better than the currently best programs for moderately sized problems. We anticipate that we can improve the performance by (a) a closer evaluation of different scoring and gap functions (which is trivial in our approach), (b) a speed up in the solution of the LP relaxation of the ILP, (c) an improvement of the variable reduction procedure.

ACKNOWLEDGMENTS

This work was supported by a grant from DAAD within the ‘Higher Education Special Program III’ (Hochschulsonderprogramm III) of the German Federal Government and the Federal States.

REFERENCES

Althaus,E., Kohlbacher,O., Lenhof,H.-P. and Müller,P. (2000) A combinatorial approach to protein docking with flexible side-chains. In *Proceedings of the 4th Annual International Conference on Computational Molecular Biology (RECOMB-00)*. ACM Press, pp. 15–24.

Altschul,S.F., Gish,W., Miller,W., Myers,E.W. and Lipman,D.J. (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.

Caprara,A., Lancia,G. and Ng,S. (1999) A column-generation based branch-and-bound algorithm for sorting by reversals. In Farach-Colton,M., Roberts,F., Vingron,M. and Waterman,M. (eds), *Mathematical Support for Molecular Biology*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 47, pp. 213–226.

Christof,T., Jünger,M., Kececioglu,J., Mutzel,P. and Reinelt,G. (1997) A branch-and-cut approach to physical mapping with end-probes. In *Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB97)*. pp. 84–93.

Dayhoff,M., Schwartz,R. and Orcut,B. (1979) A model of evolutionary change in proteins. In Dayhoff,M. (ed.), *Atlas of Protein Sequence and Structure*, Vol. 5, National Biomedical Research Foundation, Washington, DC, pp. 345–352.

Delcher,A., Kasif,S., Fleischmann,R., Peterson,J.O.W. and Salzberg,S. (1999) Alignment of whole genomes. *Nucleic Acids Res.*, **27**, 2369–2376.

Eppstein,D. (1990) Sequence comparison with mixed convex and concave costs. *J. Algorithms*, 85–101.

Gotoh,O. (1982) An improved algorithm for matching biological sequences. *J. Mol. Biol.*, **162**, 705–708.

Gupta,S., Kececioglu,J. and Schaeffer,A. (1995) Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. *J. Comput. Biol.*, **2**, 459–472.

Gusfield,D. (1997) *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge.

Henikoff,S. and Henikoff,J. (1992) Amino acid substitution matrices from protein blocks. *Proc. Natl Acad. Sci. USA*, **89**, 10915–10919.

Kececioglu,J.D., Lenhof,H.-P., Mehlhorn,K., Mutzel,P., Reinert,K. and Vingron,M. (2000) A polyhedral approach to sequence alignment problems. *Discrete Appl. Math.*, **104**, 143–186.

Lancia,G., Carr,R., Walenz,B. and Istrail,S. (2001) 101 optimal pdb structure alignments: a branch-and-cut algorithm for the maximum contact overlap problem. In *Proceedings of the 5th Annual International Conference on Computational Molecular Biology (RECOMB-01)*. ACM Press, pp. 193–202.

Larmore,L. and Schieber,B. (1990) Online dynamic programming with applications to the prediction of rna secondary structure. In *Proceedings of the First Symposium on Discrete Algorithms*. pp. 503–512.

Lenhof,H.-P., Morgenstern,B. and Reinert,K. (1999) An exact solution for the segment-to-segment multiple sequence alignment problem. *Bioinformatics*, **15**, 203–210.

Lenhof,H.-P., Reinert,K. and Vingron,M. (1998) A polyhedral approach to RNA sequence structure alignment. *J. Comput. Biol.*, **5**, 517–530.

Lermen,M. and Reinert,K. (2000) The practical use of the $\mathcal{A}^*$ algorithm for exact multiple sequence alignment. *J. Comput. Biol.*, **7**, 655–673.

Lipman,D., Altschul,S. and Kececioglu,J. (1989) A tool for multiple sequence alignment. *Proc. Natl Acad. Sci. USA*, **86**, 4412–4415.

Needleman,S. and Wunsch,C. (1970) A general method applicable to the search for similarities in the amino-acid sequence of two proteins. *J. Mol. Biol.*, **48**, 443–453.

Notredame,C., Higgins,D.G. and Heringa,J. (2000) T-coffee: a novel method for fast and accurate multiple sequence alignment. *J. Mol. Biol.*, **302**, 205–217.

Reinert,K. (1999) *A Polyhedral Approach to Sequence Alignment Problems*, Ph.D. thesis, Universität des Saarlandes.

Reinert,K., Lenhof,H.-P., Mutzel,P., Mehlhorn,K. and Kececioglu,J. (1997) A branch-and-cut algorithm for multiple sequence alignment. In *Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB-97)*. pp. 241–249.

Reinert,K., Stoye,J. and Will,T. (2000) An iterative methods for faster sum-of-pairs multiple sequence alignment. *Bioinformatics*, **16**, 808–814.

SCIL, (2002) SCIL–Symbolic Constraints for Integer Linear programming. `http://www.mpi-sb.mpg.de/SCIL`.

Smith,T.F. and Waterman,M.S. (1981) Identification of common molecular subsequences. *J. Mol. Biol.*, **147**, 195–197.

Thompson,J.D., Plewniak,F. and Poch,O. (1999) BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, **15**, 87–88.

APPENDIX

In the appendix we elaborate more details for two different classes of valid inequalities that we proposed in the paper, the *clique inequalities* and the *lifted mixed cycle inequalities*. In addition we describe the separation routines and and their time complexity.

The following general structure of maximal sets of pairwise incompatible edges will be used next. For $1 \le i < j \le k$, $1 \le l_b \le l_e \le |s^i|$, $1 \le m_b \le m_e \le |s^j|$, we let

$$\mathcal{E}^{i,j}(l_b \leftrightarrow l_e, m_b \leftrightarrow m_e)$$

denote the collection of all set of edges in $S \subseteq E^{i,j}$ such that

1. all edges in S are pairwise incompatible;
2. for each edge $\{v_l^i, v_m^j\} \in S$, $l_b \le l \le l_e$ and $m_b \le m \le m_e$;
3. S is maximal with respect to properties 1 and 2.

LEMMA 1. *Every set* $S \in \mathcal{E}^{i,j}(l_b \leftrightarrow l_e, m_b \leftrightarrow m_e)$ *has the form*

$$S = \{\{v_{l_1}^i, v_{m_1}^j\}, \dots, \{v_{l_p}^i, v_{m_p}^j\}\},$$

where

1. $l_1 = l_b, l_p = l_e, m_1 = m_e, m_p = m_b;$
2. *for* $q = 1, \dots, p-1$, *either* $l_{q+1} = l_q$ *and* $m_{q+1} = m_q - 1$ *or* $l_{q+1} = l_q + 1$ *and* $m_{q+1} = m_q$.

PROOF. By requirement (a), all edges in S are crossing, meaning that, if we order them by increasing letter of s^i to which they are incident (with appropriate tie breaking), they will also be ordered by decreasing letter of s^j to which they are incident. Therefore, we assume that the edges in S according to this order are $e_1, \dots, e_p$. Requirement (b) imposes $l_1 \ge l_b, l_p \le l_e, m_1 \le m_e, m_p \ge m_b$. It is easy to verify that the maximality requirement (c) implies that all these inequalities hold at equality, otherwise at least one edge could be added to S preserving (a) and (b). For instance, if $l_1 > l_b$, edge $\{v_{l_b}^i, v_{m_e}^j\}$ could be added to S. This shows (i). It is easy to verify that, if (ii) were violated, again at least one edge could be added to S. For instance, if $l_{q+1} = l_q + 1$ and $m_{q+1} = m_q - 1$, edge $\{v_{l_{q+1}}^i, v_{m_q}^j\}$ could be added to S. This shows (ii) and concludes the proof. □

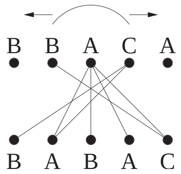

Fig. 3. Illustration of possible edges and arcs in a clique inequality.

Clique inequalities Recall that the *clique inequality* has the form

$$\sum_{e \in K_E} x_e + \sum_{a \in K_A} y_a \leq 1, \tag{11}$$

and is clearly valid for $\mathcal{P}$.

The following characterization of all (exponentially many) clique inequalities for our problem, which is an extension of the one in Reinert *et al.* (1997) (where only alignment variables are considered), will lead to an efficient algorithm for their separation.

PROPOSITION 2. *For every clique inequality (11), there exist two strings s^i, s^j such that either*

$$K_E = \emptyset, K_A = A^{i,j}(l+1 \leftrightarrow l)$$

for some $1 \leq l \leq \|s^i\|$, *or*

$$K_E \in \mathcal{E}^{i,j}(l_b \leftrightarrow l_e, 1 \leftrightarrow |s^j|), K_A = A^{i,j}(l_b \leftrightarrow l_e)$$

for some $1 \leq l_b \leq l_e \leq \|s^i\|$

PROOF. The requirement that all edges and arcs be incompatible implies $K_E \subseteq E^{i,j}$ and $K_A \subseteq A^{i,j}$ for some $1 \leq i, j \leq k, i \neq j$. If $K_E = \emptyset$, K is a maximal set of incompatible arcs in $A^{i,j}$, which is easily seen to have the form given in the statement, the resulting clique inequality being one of (4). Otherwise, note that the maximal set of arcs in $A^{i,j}$ pairwise incompatible with all edges in a set $S \subseteq E^{i,j}$ is uniquely determined by the first and last letter $s^i_{l_b}$ and $s^i_{l_e}$ of s^i to which the edges in S are adjacent. Specifically, this set of arcs is $A^{i,j}(l_b \leftrightarrow l_e)$. Along with the maximality requirement, this implies that $K_E \in \mathcal{E}^{i,j}(l_b \leftrightarrow l_e, m_b \leftrightarrow m_e)$ with $m_b = 1$ and $m_e = |s^j|$. In particular, if $m_b > 1$ (resp. $m_e < |s^j|$) then edge $\{v^i_{l_e}, v^j_1\}$ (resp. $\{v^i_{l_b}, v^j_{|s^1|}\}$) would be incompatible with all members of K.

An illustration of possible sets K_E and K_A in a clique inequality is given in Figure 3.

Lifted mixed cycle inequalities Recall that the lifted mixed cycle inequalities have the following form:

$$\sum_{e \in C} x_e \leq t - 1, \tag{12}$$

and are valid for $\mathcal{P}$.

PROPOSITION 3. *For every lifted mixed cycle inequality (12) there exists a path*

$$P = v^{i_1}_{l_1}, \{v^{i_1}_{l_1}, v^{i_2}_{l_2}\}, v^{i_2}_{l_2}, \{v^{i_2}_{l_2}, v^{i_3}_{l_3}\}, \ldots, \{v^{i_t}_{l_t}, v^{i_1}_{l_1-1}\}, v^{i_1}_{l_1-1}$$

containing t edges (and no arc) such that

1. *for* $r = 1, \ldots, t$, $\{v^{i_r}_{l_r}, v^{i_{r+1}}_{l_{r+1}}\} \in C^{i_r, i_{r+1}}$*;*
2. *for* $r = 1, \ldots, t$, $C^{i_r, i_{r+1}} \in \mathcal{E}^{i_r, i_{r+1}}(l_r \leftrightarrow |s^{i_r}|, 1 \leftrightarrow l_{r+1})$*;*

where $i_{t+1} := i_1$ *and* $l_{t+1} := l_1 - 1$.

PROOF. First of all, note that P together with arc $(v^{i_1}_{l_1-1}, v^{i_1}_{l_1}) \in A_p$ defines a mixed cycle. We shall call this the *reference* (mixed) cycle. In order to prove the statement, we first observe that any set C that satisfies conditions (i) and (ii) in the statement also satisfies (a) (by definition of $\mathcal{E}^{i_r, i_{r+1}}(\cdot)$) and (b), since any set of edges obtained by selecting one edge from each $C^{i_r, i_{r+1}}$ defines a mixed cycle, which is obtained from the reference cycle by replacing two consecutive edges by two edges with a path in A_p in between (formal details are skipped). As to the maximality requirement (c), note that the addition to $C^{i_r, i_{r+1}}$ of an edge in $E^{i_r, i_{r+1}}$ compatible with some edges already present would violate requirement (a) (and the resulting inequality would not be valid), whereas the addition of an edge $e = \{v^{i_r}_l, v^{i_{r+1}}_m\}$ with $l < l_r$ or $m > l_{r+1}$ would violate (b), since it is easy to verify that $P \setminus \{v^{i_r}_{l_r}, v^{i_{r+1}}_{l_{r+1}}\} \cup e$ is not the set of edges in a mixed cycle.

The proof is concluded by showing that every set C that satisfies (a), (b) and (c) has the form given in the statement. For $r = 1, \ldots, t$, order the edges in $C^{i_r, i_{r+1}}$ according to increasing letter of s^{i_r} to which they are incident, breaking ties by decreasing letter of $s^{i_{r+1}}$ to which they are incident, and let e_r be the last edge according to this ordering. Consider the mixed cycle C with edges $e_1, \ldots, e_t$ and note that, after possible shifting of the indices $i_1, \ldots, i_k$, we can assume without loss of generality that C contains at least one arc in A_p associated with s^{i_1}. We show that C contains only one arc. Indeed, if C contains also an arc in A_p associated with string s^{i_r} for some $1 < r \leq t$, we have a contradiction to the maximality of C as we may add to C^{i_{r-1}, i_r} the edge in E^{i_{r-1}, i_r} with the same endpoint in $s^{i_r - 1}$ as e_{r-1} and the same endpoint in s^{i_r} as e_r. An analogous reasoning shows that C contains exactly one arc in A_p associated with string s^{i_1}. Finally, (i) is verified by definition, and (ii) follows immediately by the maximality requirement on C.

For an illustration see Figure 4. Note that P together with arc $(v^{i_1}_{l_1-1}, v^{i_1}_{l_1}) \in A_p$ defines a mixed cycle. We shall call this the *reference* (mixed) cycle.

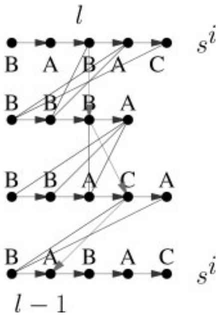

Fig. 4. Illustration of possible edges in a lifted mixed cycle inequality (edges represented by an arrow are those in the reference mixed cycle).

Polyhedral results

We summarize our polyhedral results in the following theorem, whose proof will be given in the full paper.

THEOREM 1.

- •*Clique inequalities (11) define facets if* $l_e - l_b \geq 1$, $l_b > 1$, $l_e < \|s^i\|$, $(v^i_{l_b}, v^j_{\|s^j\|-1}) \in K_E$ *and* $(v^i_{l_e}, v^j_2) \in K_E$.
- •*Lifted mixed cycle inequalities (12) define facets if* $2 \leq l_r \leq \|s^{i_r}\| - 2$ *for* $r = 1, \ldots, t$.
- •*Generalized transitivity inequalities (10) define facets if there are* l *and* m *such that* $s^{i_2}_l \notin S^2$, $s^{i_2}_{l+1} \notin S^2$, $s^{i_3}_m \notin S^3$ *and* $s^{i_3}_{m+1} \notin S^3$.

The Branch-and-cut algorithm

Branch-and-cut is one of the most successful frameworks for the solution of ILPs and works as follows. We relax the integrality condition and solve the corresponding (exponentially large) LP. If all variables in the optimal solution of the LP are integral, we are done. Otherwise we partition the problem into two subproblems by choosing a variable with a fractional value and fixing it to 0 and 1 in the two subproblems. The procedure is iterated on the subproblems.

Our integer program contains an exponential number of clique, lifted mixed cycle and generalized transitivity inequalities. Obviously, we cannot afford to simply give all of them to a solver. Therefore we use cutting plane approach. We start with a initial, small set of constraints and solve the LP. We call the constraints that are given to the LP solver the *active* constraints. We test, if the solution of the LP over the active constraints violates any non-active constraint. If so, we add one or more violated constraints to the set of active constraints and iterate. Thus we have to solve the following problem: given a solution of an LP, test if a violated constraint exists and if so, find one. This problem is called the *separation problem* and the violated constraint is called a *cutting plane*. We describe efficient algorithms that solve the separation problem for the different classes of inequalities below. The fact that polynomial-time separation algorithms exist for the facet-defining inequalities that we present is somehow unusual (in the positive sense), in that often efficient separation procedures are known only for *weaker* versions of the facet-defining (i.e. strongest possible) inequalities that one can derive for the problem at hand. In this section we also describe a simple but efficient procedure to reduce the set of variables. The other details of the branch-and-cut algorithm are standard and deferred to the full paper.

Separation procedures

Note first that there is no need for separation algorithms for the constraints in our initial ILP model, since there are only $n(k-1)$ equations of type (2) and inequalities of type (4), transitivity inequalities (5) are a special case of generalized transitivity inequalities, and mixed cycle inequalities (3) are dominated by lifted mixed cycle inequalities. In the following, let (x^*, y^*) be the solution that should be separated.

Pairgraphs Given a clique K and edge weights w_e for all edges $e \in E$, it is useful to compute $K_E \in \mathcal{E}^{i,j}(l_b \leftrightarrow l_e, m_b \leftrightarrow m_e)$, which maximizes $\sum_{e \in K_E} w_e$. According to Lemma 1 one may use dynamic programming. We represent the dynamic programming procedure as a path computation in a directed acyclic graph. Specifically, we need the notion of *pairgraph*, introduced by Reinert et al. Reinert *et al.* (1997). They used them in the special case $l_b = 1$, $l_e = \|s^1\|$, $m_b = 1$, and $m_e = \|s^1\|$, which generalizes directly to our case. The pairgraph $PG^{i,j}$ for an ordered pair of strings s^i, s^j is a directed acyclic graph with a node n_e for every edge $e \in E^{i,j}$ of the alignment graph. Every node $n_{\{v^i_l, v^j_m\}}$ has up to two outgoing arcs, namely $(n_{\{v^i_l, v^j_m\}}, n_{\{v^i_l, v^j_{m-1}\}})$, if $m > 1$, and $(n_{\{v^i_l, v^j_m\}}, n_{\{v^i_{l+1}, v^j_m\}})$, if $l < \|s^i\|$.

LEMMA 2. *Every path* $n_{e_1}, \ldots, n_{e_l}$ *in* $PG^{i,j}$ *such that* $e_1 = \{v^i_{l_b}, v^j_{m_e}\}$ *and* $e_l = \{v^i_{l_e}, v^j_{m_b}\}$, *corresponds to a set* $\{e_1, \ldots, e_l\} \in \mathcal{E}^{i,j}(l_b \leftrightarrow l_e, m_b \leftrightarrow m_e)$ *and vice versa.*

PROOF. Follows directly from Lemma 1.

For further details on pairgraphs, see Reinert *et al.* (1997). According to Lemma 2, the set $K_E \in \mathcal{E}^{i,j}(l_b \leftrightarrow l_e, m_b \leftrightarrow m_e)$ which maximizes $\sum_{e \in K_E} w_e$ corresponds to the longest node-weighed path in $PG^{i,j}$ between $n_{\{v^i_{l_b}, v^j_{m_e}\}}$ and $n_{\{v^i_{l_e}, v^j_{m_b}\}}$ with respect to node weights

$w_{n_e} := w_e$ for $e \in E$. Since $PG^{i,j}$ is acyclic, we can compute such a path in linear time. In order to speed up the computation, we use a sparse version of the pairgraph, as described in Reinert *et al.* (1997).

Clique inequalities In the separation of clique inequalities (11), we fix the strings s^i and s^j, $i \neq j$, considering all $k(k-1)$ pairs. For every $1 \leq l_b < l_e \leq \|s^i\|$, we (a) compute $K_E \in \mathcal{E}^{i,j}(l_b \leftrightarrow l_e, 1 \leftrightarrow \|s^j\|)$ which maximizes $\sum_{e \in K_E} x_e^*$ by finding a longest path in the pairgraph from $n_{\{v_{l_b}^i, v_{\|s^j\|}^j\}}$ to $n_{\{v_{l_e}^i, v_1^j\}}$; (b) compute $\sum_{a \in A^{i,j}(l_b \leftrightarrow l_e)} y_a^*$; and (c) test if the corresponding clique inequality is violated, that is if $\sum_{e \in K_E} x_e^* + \sum_{a \in A^{i,j}(l_b \leftrightarrow l_e)} y_a^* > 1$.

Given i and j, Step (a) takes, overall, $O(\|s^i\|^2 \|s^j\|)$ time, since for each of the $\|s^i\| - 1$ values of l_b we find the shortest path tree from $n_{\{v_{l_b}^i, v_{\|s^j\|}^j\}}$ in $PG^{i,j}$, whose size is $O(\|s^i\| \|s^j\|)$. Step (b) requires overall $O(\|s^i\|^2)$ time, noting that there are $\|s^i\|^2$ gap variables of length at most $\|s^i\|$. The running time for Step (c) is overall $O(\|s^i\|^2)$. Thus the total running time is $O(\sum_{i=1}^k \sum_{j=1}^k \|s^i\|^2 \|s^j\| + \sum_{i=1}^k \|s^i\|^2) = O(n^3)$. However, if all strings have approximately the same length n/k, the actual complexity is roughly n^3/k.

The discussion above shows

THEOREM 2. *Clique inequalities (11) can be separated in $O(n^3)$ time.*

To accelerate the separation (mainly in the first iterations), we use some heuristics described in the full paper.

Lifted mixed cycle inequalities For the separation of lifted mixed cycle inequalities (12), it is easy to observe that, if $v_{l_r}^{i_r} v_{l_{r+1}}^{i_{r+1}}$ is an edge of the reference mixed cycle corresponding to a violated inequality, then $C^{i_r, i_{r+1}}$ can be defined as the element of $\mathcal{E}^{i_r, i_{r+1}}(l_r \leftrightarrow \|s^{i_r}\|, 1 \leftrightarrow l_{r+1})$ which maximizes $\sum_{e \in C^{i_r, i_{r+1}}} x_e^*$. Accordingly, let $H = (V, E')$ be the directed graph with the same nodes as G and two arcs $(u, v), (v, u) \in E'$ for every edge $\{u, v\}$ of G. The weight w_a of an arc $a = (v_l^i, v_m^j) \in E'$ is defined as

$$w_a := 1 - \max_{K_E \in \mathcal{E}^{i,j}(l \leftrightarrow \|s^i\|, 1 \leftrightarrow m)} \sum_{e \in K_E} x_e^*.$$

The most violated lifted mixed cycle inequality whose reference cycle contains the arc $(v_{l_1-1}^{i_1}, v_{l_1}^{i_1}) \in A_p$ (if any) corresponds to a minimum-weight path in H from $v_{l_1}^{i_1}$ to $v_{l_1-1}^{i_1}$. Indeed, if this path, say P, is given by arcs $(v_{l_1}^{i_1}, v_{l_2}^{i_2}), \ldots, (v_{l_t}^{i_t}, v_{l_1-1}^{i_1})$, the violation is given by

$$\sum_{r=1}^{t} \sum_{e \in C^{i_r, i_{r+1}}} x_e^* - t + 1 = 1 - \sum_{r=1}^{t} (1 - \sum_{e \in C^{i_r, i_{r+1}}} x_e^*)$$
$$= 1 - \sum_{a \in P} w_a,$$

i.e. the inequality is violated if and only if the weight of P is smaller than 1. If all clique inequalities are satisfied, which can be assumed if we separate these inequalities before lifted mixed cycle inequalities, we know that $w_a \geq 0$ for $a \in E'$ and thus we can find the path P with Dijkstra's algorithm.

Computing the weights for the arcs in E' can be done in $O(n^3)$ by finding the all pairs shortest paths in each of the $\binom{k}{2}$ pairgraphs (and is already done for the separation of clique inequalities). The running time for each call of Dijkstra's algorithm is $O(n^2)$, since graph H has n nodes and $O(n^2)$ edges. Hence, the total running time is $O(n^3)$, since Dijkstra's algorithm is called $n - k$ times, once for each candidate pair $v_{l_1-1}^{i_1}, v_{l_1}^{i_1}$. This shows

THEOREM 3. *Lifted mixed cycle inequalities (12) can be separated in $O(n^3)$ time.*

To accelerate the separation, we sparsify graph H by the same idea as in Reinert *et al.* (1997) (details are given in the full paper).

Generalized transitivity inequalities For the separation of generalized transitivity inequalities (10), we fix the three strings $s^{i_1}, s^{i_2}, s^{i_3}$ along with the letter $s_{l_1}^{i_1}$ of string s^{i_1}. Then we need to check the existence of two sets $A \subseteq \{1, \ldots, \|s^{i_2}\|\}$ and $B \subseteq \{1, \ldots, \|s^{i_3}\|\}$ such that

$$\sum_{l \in A} x^*_{\{v_{l_1}^{i_1}, v_l^{i_2}\}} + \sum_{m \in B} x^*_{\{v_{l_1}^{i_1}, v_m^{i_3}\}} - \sum_{l \in A} \sum_{m \in B} x^*_{\{v_l^{i_2}, v_m^{i_3}\}} > 1.$$

Consider the complete bipartite graph $B = (U \cup V, F)$ in which $U = \{1, \ldots, \|s^{i_2}\|\}$ and $V = \{1, \ldots, \|s^{i_3}\|\}$. Assign to each vertex $l \in U$ profit $p_l := x^*_{\{v_{l_1}^{i_1}, v_l^{i_2}\}}$, to each vertex $m \in V$ profit $p_m := x^*_{\{v_{l_1}^{i_1}, v_m^{i_3}\}}$, and to each edge $\{l, m\} \in F$ cost $c_{\{l,m\}} := x^*_{\{v_l^{i_2}, v_m^{i_3}\}}$. Clearly, finding the inequality (10) which is most violated is equivalent to finding subsets $A \subseteq U$ and $B \subseteq V$ such that

$$\sum_{l \in A} p_l + \sum_{m \in B} p_m - \sum_{l \in A} \sum_{m \in B} c_{\{l,m\}}$$

is maximized, or equivalently finding subsets $\overline{A} \subseteq U$ and $\overline{B} \subseteq V$ such that

$$\sum_{l \in \overline{A}} p_l + \sum_{m \in \overline{B}} p_m + \sum_{l \in U \setminus \overline{A}} \sum_{m \in V \setminus \overline{B}} c_{\{l,m\}} \qquad (13)$$

is minimized. We show that the second problem is equivalent to finding an $\{s, t\}$-cut of minimum value in a suitable directed network. This network has node set $\{s, t\} \cup U \cup V$, where s plays the role of the *source* and t of the *sink*, an arc (s, l) of capacity p_l for each $l \in U$, an arc (m, t) of capacity p_m for each $m \in V$, and an arc (l, m) of capacity $c_{\{l,m\}}$ for each $l \in U, m \in V$. Recall that the value of an $\{s, t\}$-cut defined by node set S, with $s \in S$, $t \in \overline{S}$, is given by the sum of the capacities of all arcs with tail in S and head in $\overline{S}$, where $\overline{S}$ is the set of nodes outside S. Letting $\overline{A} := U \cap \overline{S}$ and $\overline{B} := V \cap S$, it is simple to verify that the value of this cut is given by (13). Note that it may well be the case that either $\overline{A} = U$ or $\overline{B} = V$ in the above problem (i.e. either $A = \emptyset$ or $B = \emptyset$), which means that no inequality (10) is violated since, by equations (2) both $\sum_{l \in U} x^*_{\{v^{i_1}_{l_1}, v^{i_2}_{l}\}} \leq 1$ and $\sum_{m \in V} x^*_{\{v^{i_1}_{l_1}, v^{i_3}_{m}\}} \leq 1$. Note also that the two problems above are generalizations of Stable Set and Vertex Cover in a bipartite graph (arising when the edge costs are are either 0 or ∞), and in fact the solution method above is a simple extension of the method for these problems.

In the separation procedure above, for each triple $s^{i_1}, s^{i_2}, s^{i_3}$ and $l_1 \in \{1, \ldots, \|s^{i_1}\|\}$, we have to compute a maximum flow in a network with $O(\|s^{i_2}\| + \|s^{i_3}\|)$ nodes, which can be done in $O(\|s^{i_2}\|^3 + \|s^{i_3}\|^3)$ time. The overall complexity is $O(\sum_{i_1=1}^{k} \sum_{i_2=1}^{k} \sum_{i_3=1}^{k} \|s^{i_1}\|(\|s^{i_2}\|^3 + \|s^{i_3}\|^3)) = O(n^4)$. As before, if each string has approximately the same length n/k, the complexity is roughly n^4/k. This shows

THEOREM 4. *Generalized transitivity inequalities (10) can be separated in $O(n^4)$ time.*