Simple and Fast Inverse Alignment*

John Kececioglu and Eagu Kim

Department of Computer Science,
The University of Arizona, Tucson, AZ 85721, USA
{kece, egkim}@cs.arizona.edu

Abstract. For as long as biologists have been computing alignments of sequences, the question of what values to use for scoring substitutions and gaps has persisted. While some choices for substitution scores are now common, largely due to convention, there is no standard for choosing gap penalties. An objective way to resolve this question is to learn the appropriate values by solving the *Inverse String Alignment Problem*: given examples of correct alignments, find parameter values that make the examples be optimal-scoring alignments of their strings.

We present a new polynomial-time algorithm for Inverse String Alignment that is *simple* to implement, *fast* in practice, and for the first time can learn hundreds of parameters simultaneously. The approach is also *flexible*: minor modifications allow us to solve inverse *unique* alignment (find parameter values that make the examples be the unique optimal alignments of their strings), and inverse *near-optimal* alignment (find parameter values that make the example alignments be as close to optimal as possible). Computational results with an implementation for global alignment show that, for the first time, we can find best-possible values for all 212 parameters of the standard protein-sequence scoring-model from hundreds of alignments in a few minutes of computation.

Keywords: Sequence analysis, parametric sequence alignment, substitution score matrices, affine gap penalties, supervised learning, linear programming, cutting plane algorithms.

1 Introduction

Perhaps the most studied problem in computational biology is the alignment of biological sequences with substitutions, insertions, and deletions. The standard formulations of string alignment optimize a sum of scores for each type of operation, often giving a penalty for a run of insertions or deletions, called a gap, that is linear in the length of the gap. When performing sequence alignment in practice, the question of what weights and penalties to use inevitably arises. An interesting attack on this question is *parametric sequence alignment*, where for a given pair of strings, the alignment problem is solved for effectively all possible choices of the scoring parameters, thereby eliminating the need to specify *any* weights and penalties. The problem with this approach is that it in effect defers

* Research supported by US National Science Foundation Grant DBI-0317498.

the question, since eventually a user must choose one of the solutions, and on what basis should this be done? Essentially one needs an example of a *correct alignment* to discriminate among the welter of parametric solutions. When one does solve the parametric problem and knows a biologically correct alignment of the sequences, this alignment is used to decide what region of the parameter space makes the correct alignment have optimal score.

This optimal parameter choice could be found much more directly, however, by solving *inverse parametric alignment*. In this problem, the input is an alignment of a pair of strings, and the output is a choice of parameters that makes the input alignment be an optimal-scoring alignment of its strings. We present a simple and fast algorithm for inverse parametric alignment that for the first time is capable of determining all substitution weights and gap penalties simultaneously. Such an algorithm, applied to a collection of benchmark protein-sequence alignments that are constructed by aligning their three-dimensional structures, could provide the first rigorous way of determining substitution scores and gap penalties for characterized classes of proteins.

Related Work. The first algorithms for *parametric* alignment of two sequences were discovered in the early 1990's by Waterman, Eggert and Lander [16] and Gusfield, Balasubramanian and Naor [7]. These algorithms handled *two* parameters, usually the gap open and extension penalties with fixed substitution scores. Zimmer and Lengauer [17] addressed numerical stability. Gusfield et al. [7] also bounded the number of regions in the decomposition of the parameter space, and constructed the decomposition with one optimal alignment computation per region. Fernández-Baca, Seppäläinen and Slutzki [4] showed these bounds are asymptotically tight, and initiated the study of parametric multiple-sequence alignment. Gusfield and Stelling [8] released a software implementation called `XPARAL`, and were the first to consider *inverse* parametric alignment, for which they gave a heuristic that attempted to avoid computing a decomposition of the entire parameter space. Pachter and Sturmfels [13] explored the relation of algebraic statistics to parametric sequence alignment.

Recently, Sun, Fernández-Baca and Yu [15] gave the first direct algorithm for inverse parametric alignment. While they consider three parameters, their solution effectively fixes one parameter value at zero. For two strings of length n, their algorithm runs in $O(n^2 \log n)$ time. Their approach is involved, and does not appear to have been implemented.

In contrast to prior work, our algorithm for inverse parametric alignment is simple to implement, does not compute a decomposition of the entire parameter space, solves both the near-optimal and unique-optimal inverse alignment problems, handles a set of input alignments, and for the first time can quickly solve problems with hundreds of free parameters.

Overview. In the next section we give a precise statement of the inverse parametric alignment problem and two variations: inverse near-optimal and unique-optimal alignment. Section 3 reduces all three variations to linear programming. Section 4 explains how the resulting linear programs, even though they have an exponential number of inequalities, can be solved in polynomial time. Section 5 then discusses a version of this approach, called a cutting-plane algorithm, that is highly effective in practice. Finally Section 6 presents experimental results with an implementation for inverse optimal and near-optimal global alignment.

2 Inverse Alignment and Its Variations

The standard string alignment problem is: given a pair of strings A and B and a function f that scores alignments, where f usually has several parameters that weight substitutions, insertions, deletions, and identities, find an alignment $\mathcal{A}$ of A and B that has optimal score under f. The *inverse alignment problem* turns this around: given an alignment $\mathcal{A}$, find values for the parameters of f that make $\mathcal{A}$ be an optimal alignment. (From the view of machine learning, this learns the parameters for optimal alignment from training examples of correct alignments.) Of course to find reliable values when f has many parameters may require several alignments. Formally, we define inverse alignment as follows.

Definition 1 (Inverse Optimal Alignment). The *Inverse String Alignment Problem* is the following. The input is a collection of alignments $\mathcal{A}_1, \ldots, \mathcal{A}_k$ of strings, and an alignment scoring function f_w with parameters $w = (w_1, \ldots, w_p)$. The output is values $x = (x_1, \ldots, x_p)$ for the parameters such that each $\mathcal{A}_i$ is an optimal alignment of its strings under f_x. □

For example, the $\mathcal{A}_i$ might be structural global alignments of pairs of protein sequences, and f might score alignments using substitution scores σ_{ab} for all pairs of amino acids a, b together with a gap-open penalty γ and a gap-extension penalty λ. In this case, scoring function f has 212 parameters. For another example, the $\mathcal{A}_i$ might be local alignments of pairs of strings, also scored using substitutions and gap penalties, or the $\mathcal{A}_i$ might even be alignments of alignments [11] scored using the weighted sum-of-pairs measure.

Note that Inverse Optimal Alignment may have no solution: it may be that no choice x for the parameter values makes the $\mathcal{A}_i$ all be optimal alignments. An algorithm for inverse alignment must detect this situation, and report that no solution exists.

Given that it may be impossible to find parameters that make the alignments in a collection all be optimal, we might instead seek the next-best thing: parameters that make the alignments all be *near-optimal.* When the objective is to minimize scoring function f, we say an alignment $\mathcal{A}$ of a set of strings $\mathcal{S}$ is ϵ*-optimal* for some $\epsilon \geq 0$ if

$$f(\mathcal{A}) \;\; \leq \;\; (1+\epsilon)\, f(\mathcal{A}^*), \qquad (1)$$

where $\mathcal{A}^*$ is an optimal alignment of $\mathcal{S}$ under f. Note that when $\epsilon = 0$, an ϵ-optimal alignment is optimal.

Definition 2 (Inverse Near-Optimal Alignment). The *Inverse Near-Optimal Alignment Problem* is: given a collection of alignments $\mathcal{A}_i$, scoring function f, and a real number $\epsilon \geq 0$, find parameter values x such that each alignment $\mathcal{A}_i$ is ϵ-optimal under f_x. □

For large enough ϵ, Inverse Near-Optimal Alignment always has a solution. In practice though we might not know an appropriate value of ϵ in advance. Nevertheless an algorithm for Inverse Near-Optimal Alignment can be used to efficiently find the *smallest* value ϵ^* for which there is a solution, to any desired accuracy $\xi > 0$. First, by repeated doubling, find an upper bound b on ϵ^* by iteratively solving Inverse Near-Optimal Alignment with $\epsilon = 2^i \xi$ for an i of $1, 2, \ldots$ until a solution is found. Then, given upper bound b and lower bound $a = b/2$ on ϵ^*, perform binary search on the real values in interval $[a, b]$ that are spaced distance ξ apart. This finds the best possible ϵ to within accuracy ξ using $O(\log(\epsilon/\xi))$ calls to an algorithm for Inverse Near-Optimal Alignment. As we show in Section 6, such an approach is very fast in practice.

Finally, in some applications of inverse alignment we may need to find parameter values that make a given alignment be the *unique* optimal alignment of its strings. (For example, suppose we have alignment software that attempts to optimize a scoring function; when testing how well the software performs at recovering a benchmark alignment, the best parameter values to use would be those that make the benchmark be the unique optimal alignment.) To be the unique optimal alignment, every other alignment must score worse. We quantify how much worse as follows. When the objective is to minimize scoring function f, we say an alignment $\mathcal{A}$ of a set of strings $\mathcal{S}$ is *δ-unique* for some $\delta > 0$ if

$$f(\mathcal{B}) \;\geq\; f(\mathcal{A}) + \delta,$$

for every alignment $\mathcal{B}$ of $\mathcal{S}$ other than $\mathcal{A}$.

Definition 3 (Inverse Unique-Optimal Alignment). The *Inverse Unique-Optimal Alignment Problem* is: given a collection of alignments $\mathcal{A}_i$, scoring function f, and a real number $\delta > 0$, find parameter values x such that each alignment $\mathcal{A}_i$ is a δ-unique alignment of its strings under f_x. □

Note that using the same doubling and binary-search idea described above to find the smallest ϵ for Inverse Near-Optimal Alignment, we can find the *largest* $\delta > 0$ for which the $\mathcal{A}_i$ are all δ-unique, to within accuracy $\xi > 0$, using $O(\log(\delta/\xi))$ calls to an algorithm for Inverse Unique-Optimal Alignment.

When the alignment scoring function f is *linear* in its parameters—as is the case for most forms of alignment used in practice (including the standard formulations of global and local alignment)—all three variations of inverse alignment can be solved using linear programming, as we show next.

3 Reduction to Linear Programming

For most standard forms of alignment, the alignment scoring function f is a linear function of its parameters. We make this precise as follows. In general suppose that f scores an alignment $\mathcal{A}$ by measuring $p+1$ features of $\mathcal{A}$ through functions $f_0, f_1, \ldots, f_p$, and combines these measures into one score through a weighted sum involving p parameters $w_1, \ldots, w_p$, by

$$f(\mathcal{A}) \;:=\; f_0(\mathcal{A}) + f_1(\mathcal{A})\, w_1 + \cdots + f_p(\mathcal{A})\, w_p. \tag{2}$$

Then we say f is *linear* in parameters $w_1, \ldots w_p$. (Note that f_0 is not weighted by a parameter.) When we want to indicate the dependence of function f on all its parameters $w = (w_1, \ldots, w_p)$, we write f_w.

For a concrete example, consider *standard global alignment* of two protein sequences with linear gap penalties, where a substitution of letter a by b has similarity value σ_{ab}, and a gap of length ℓ incurs a penalty of $\gamma + \lambda\ell$. (A *gap* in an alignment is a maximal run of either insertions or deletions; the *length* of the gap is the number of letters in the run. Here γ is the gap-open penalty and λ is the gap-extension penalty.) Suppose we have fixed all the similarity values σ_{ab} by choosing one of the standard substitution-score matrices (such as a `PAM` [2] or `BLOSUM` [9] matrix). If the only parameter values we want to find through inverse alignment are the gap open and extension penalties, we have $p = 2$ parameters: γ and λ. For an alignment $\mathcal{A}$, let

- $g(\mathcal{A})$ be the number of gaps in $\mathcal{A}$,
- $\ell(\mathcal{A})$ be the total length of all gaps in $\mathcal{A}$, and
- $s(\mathcal{A})$ be the total score of all substitutions (including identities) in $\mathcal{A}$.

Then the similarity score of alignment $\mathcal{A}$ is

$$f(\mathcal{A}) \;:=\; s(\mathcal{A}) \;-\; g(\mathcal{A})\,\gamma \;-\; \ell(\mathcal{A})\,\lambda. \tag{3}$$

Here $(f_0, f_1, f_2) = (s, g, \ell)$ and $(w_1, w_2) = (-\gamma, -\lambda)$ in the notation of (2).

On the other hand, if no parameters are fixed and we want to find values for all the substitution scores σ_{ab} and gap penalties simultaneously, then the scoring function becomes

$$f(\mathcal{A}) \;:=\; \left(\sum_{a,b} h_{ab}(\mathcal{A})\, \sigma_{ab} \right) \;-\; g(\mathcal{A})\,\gamma \;-\; \ell(\mathcal{A})\,\lambda, \tag{4}$$

where a and b range over all letters in the alphabet, and the functions $h_{ab}(\mathcal{A})$ count the number of substitutions in $\mathcal{A}$ that replace a by b. For the protein alphabet of 20 amino acids, there are 210 substitution parameters σ_{ab}. These plus the two gap parameters gives $p = 212$ total parameters. Here $f_0(\mathcal{A}) = 0$ in the notation of equation (2).

When the scoring function f is linear in its parameters, we can solve Inverse Optimal, Near-Optimal, and Unique-Optimal Alignment using linear programming. Recall that the *Linear Programming Problem* is: given a collection of variables $x = (x_1, \ldots, x_n)$, a system of linear inequalities in the variables x, and a linear objective function in the variables x, find an assignment x^* of real values to the variables that satisfies all the inequalities and minimizes the objective function. In matrix notation, given a system of m inequalities in the n variables whose left-hand sides are specified by an $m \times n$ coefficient matrix A and whose right-hand sides are specified by an m-vector b, together with an n-vector c of coefficients for the objective function, Linear Programming finds

$$x^* \;:=\; \operatorname*{argmin}_{x \geq 0} \bigl\{ cx \;:\; Ax \geq b \bigr\}.$$

Here x^* is an *optimal solution* to the linear program, and any $x \geq 0$ that satisfies $Ax \geq b$ is a *feasible solution.* In general a linear program may be *infeasible* (has no feasible solution), *bounded* (has an optimal feasible solution), or *unbounded* (has feasible solutions that are arbitrarily good under the objective).

Inverse Optimal Alignment. We can solve Inverse Optimal Alignment for a linear scoring function in a very natural way by linear programming. The *variables* $x = (x_1, \ldots, x_p)$ in the linear program correspond to the scoring-function parameters $w = (w_1, \ldots, w_p)$. Note that the condition $x \geq 0$ for linear programs is not a restriction. We can scale any linear scoring function by a positive amount (without changing the relative rank of alignments) so its parameters lie in the interval $[-1, 1]$. Then replacing every occurrence of parameter w_i by $x_i - 1$ yields variables satisfying $x \geq 0$.

We use the following system of *inequalities*. Let $\mathcal{S}_i$ be the set of strings that $\mathcal{A}_i$ aligns. For each $\mathcal{A}_i$ and *every* alignment $\mathcal{B}$ of $\mathcal{S}_i$, we have an inequality

$$f_x(\mathcal{B}) \quad \geq \quad f_x(\mathcal{A}_i). \tag{5}$$

These inequalities simply express that $\mathcal{A}_i$ is a minimum-score alignment of strings $\mathcal{S}_i$, and hence $\mathcal{A}_i$ under parameter values x is an optimal alignment. (Note that if the objective is to maximize scoring function f, the direction of inequality (5) should be reversed. Then negating all the inequalities puts them into the canonical form $Ax \geq b$ for the linear program.) Written in terms of $x_1, \ldots, x_p$, inequality (5) is by equation (2) equivalent to the linear inequality

$$\sum_{1 \leq j \leq p} \big(f_j(\mathcal{B}) - f_j(\mathcal{A}_i) \big)\, x_j \quad \geq \quad \big(f_0(\mathcal{A}_i) - f_0(\mathcal{B}) \big). \tag{6}$$

Note that for any given alignments $\mathcal{A}_i$ and $\mathcal{B}$, the quantities $f_j(\mathcal{B}) - f_j(\mathcal{A}_i)$ in inequality (6) are constants that serve as the coefficients of the variables x.

Of course this yields a linear program with a huge number of inequalities. Suppose $\mathcal{A}_i$ aligns two strings of length n. The number of alignments of this pair of strings [5] is $\Theta((3+\sqrt{2})^n / n^{1/2}) = \Omega(4^n)$, which is the number of alignments $\mathcal{B}$. Every such $\mathcal{B}$ generates an inequality in the linear program. So an inverse alignment problem with p parameters and k input alignments, each of which aligns two or more strings of length n or greater, generates a linear program with $\Omega(k\, 4^n)$ inequalities in p variables.

Surprisingly, for many forms of sequence alignment this linear program can be solved in *polynomial time*—even though it has an exponential number of inequalities—due to a deep result that we call the Separation Theorem. In Section 4 we discuss how this theorem guarantees that we can efficiently solve this linear programming formulation.

One advantage of this linear programming-based approach is that we may also specify any linear *objective function* that we wish for the linear program. While every feasible solution $x \geq 0$ that satisfies the above inequalities $Ax \geq b$ yields a choice of parameters that makes the $\mathcal{A}_i$ optimal, some choices may be more biologically desirable. For instance with linear gap penalities, biologists generally prefer a *large* gap-open penalty γ and a *small* gap-extension penalty λ, since real alignments typically consist of a few long gaps. We are free to use any objective function that is linear in x to pick out a feasible solution that is more desirable. Section 5 discusses some objective functions that are appropriate for standard global alignment.

Near-Optimal Alignment. To extend this to Inverse Near-Optimal Alignment simply involves modifying the inequalities (5). Given $\epsilon \geq 0$, we use the system

$$(1+\epsilon)\, f_x(\mathcal{B}) \;\; \geq \;\; f_x(\mathcal{A}_i), \tag{7}$$

which for each $\mathcal{A}_i$ again has an inequality for every alignment $\mathcal{B}$ of strings $\mathcal{S}_i$. This is a linear inequality as well in the variables $x_1, \ldots, x_p$. Note that if inequality (7) holds for every $\mathcal{B}$, then in particular it holds for $\mathcal{B} = \mathcal{A}^*$ where $\mathcal{A}^*$ is an optimal alignment of $\mathcal{S}_i$ under f_x—and vice versa. So by the definition given in inequality (1), the system ensures each $\mathcal{A}_i$ is ϵ-optimal.

Unique-Optimal Alignment. To solve Inverse Unique-Optimal Alignment for a given $\delta > 0$, the system simply has an inequality

$$f_x(\mathcal{B}) \;\; \geq \;\; f_x(\mathcal{A}_i) \; + \; \delta, \tag{8}$$

for each $\mathcal{A}_i$ and every alignment $\mathcal{B}$ of $\mathcal{S}_i$ with $\mathcal{B} \neq \mathcal{A}_i$, which is again a linear inequality in x.

We next explain how this linear programming formulation can be solved in polynomial time for most forms of sequence alignment.

4 Solving the Linear Program

One of the truly far-reaching results in linear programming is what we call the Separation Theorem. This result was discovered in the early 1980's by Grötschel, Lovász and Schrijver [6], Padberg and Rao [14], and Karp and Papadimitriou [10]. To explain it requires a few concepts. Linear programming optimizes a linear function of real variables over a domain given by linear inequalities. Geometrically this domain, which is an intersection of half-spaces, is a convex body called a *polyhedron.* If the inequalities have rational coefficients, the polyhedron is *rational.* A polyhedron that contains no infinite rays is *bounded.*

The *optimization problem* for a rational polyhedron $\mathcal{P} \subseteq \mathcal{R}^d$ is: Given rational coefficients c that specify the objective function, find a point $x \in \mathcal{P}$ that minimizes cx, or determine that $\mathcal{P}$ is empty. The *separation problem* for $\mathcal{P}$ is: Given a point $y \in \mathcal{R}^d$, either (1) find rational coefficients w and b that specify an inequality such that $wx \leq b$ for all $x \in \mathcal{P}$, but $wy > b$; or (2) determine that $y \in \mathcal{P}$. In other words, a *separation algorithm* that solves the separation problem for polyhedron $\mathcal{P}$ determines whether point y lies inside $\mathcal{P}$, and if it lies outside, finds an inequality that is satisfied by all points in $\mathcal{P}$ but is violated by y. Such a *violated inequality* gives a hyperplane that separates y from $\mathcal{P}$.

The Separation Theorem says that, remarkably, optimization and separation are equivalent: an efficient separation algorithm for a linear program yields an efficient algorithm for solving that linear program, and vice versa.

Theorem 1 (Equivalence of Separation and Optimization [6, 14, 10]). *The optimization problem on a bounded rational polyhedron can be solved in polynomial time if and only if the separation problem can be solved in polynomial time.* □

The precise definition of polynomial time in the above is rather technical, but essentially means polynomial in the number n of *variables* in the system of inequalities describing the polyhedron (really, polynomial in its dimension and the number of digits in the rational coefficients). The import is that a linear program that is implicitly described by a list $\mathcal{L}$ of $2^{\Omega(n)}$ inequalities can be solved in $n^{O(1)}$ time if, for any candidate solution y, one can in $n^{O(1)}$ time determine that y satisfies all the inequalities in $\mathcal{L}$, or if it does not, report an inequality in $\mathcal{L}$ that y violates. Of course a separation algorithm that simply scans list $\mathcal{L}$ and tests each inequality will not achieve this time bound.

The proof of Theorem 1 exploits properties of the ellipsoid algorithm for linear programming. As a consequence, the polynomials bounding the running times have high degree, so the theorem does not directly yield algorithms for quickly solving exponentially-large linear programs in practice. Its main use is in proving that a polynomial-time algorithm exists.

To solve a linear program in practice using a separation algorithm, the following iterative approach is usually taken.

(1) Start with a small subset $\mathcal{S}$ of the inequalities in $\mathcal{L}$.
(2) Compute an optimal solution x to the linear program given by subset $\mathcal{S}$.
(3) Call the separation algorithm for $\mathcal{L}$ on x. If the algorithm reports that x satisfies $\mathcal{L}$, output x and halt: x is an optimal solution for $\mathcal{L}$.
(4) Otherwise, add the violated inequality returned by the separation algorithm to $\mathcal{S}$, and loop back to Step (2).

This kind of approach is known as a *cutting-plane algorithm*. Such algorithms often find optimal solutions very quickly in practice, even if they are not guaranteed to run in polynomial time. In Section 6 we show that the resulting cutting-plane algorithm for global alignment is indeed fast, solving instances with hundreds of parameters and alignments in a few minutes of computation.

In the remainder of this section we show that a polynomial-time *alignment algorithm* (in other words, an algorithm that computes an optimal alignment given fixed values for the parameters) yields a polynomial-time *separation algorithm* for our linear programming formulations of inverse alignment. Combined with Theorem 1, this proves our main result.

Theorem 2 (Complexity of Inverse Alignment). *Inverse Optimal and Near-Optimal Alignment can be solved in polynomial time for any form of alignment in which: (1) the alignment scoring-function is* linear *in its parameters, (2) the parameters values can be* bounded*, and (3) for any fixed parameter choice, an* optimal *alignment can be found in polynomial time. Inverse Unique-Optimal Alignment can be solved in polynomial time if in addition, for any fixed parameter choice, a* next-best *alignment can be found in polynomial time.* □

Optimal and Near-Optimal Alignment. We now give separation algorithms for each variation of inverse alignment. Recall that given an assignment of values x for the scoring-function parameters, a separation algorithm decides whether x satisfies all the inequalities in the linear program, and if it does not, identifies a violated inequality.

For inverse *optimal* alignment, the linear program consists of inequalities (5) for each input alignment $\mathcal{A}_i$. Conceptually we have a different separation algorithm for the inequalities associated with each $\mathcal{A}_i$. To separate all inequalities, run the separation algorithms for $\mathcal{A}_1, \ldots, \mathcal{A}_k$ consecutively. As soon as one algorithm finds a violated inequality, we halt and return the inequality. If no algorithm finds a violated inequality, we report x satisfies the linear program.

To separate the system of inequalities associated with a particular $\mathcal{A}_i$, simply compute an optimal alignment $\mathcal{B}^*$ under scoring function f_x over the strings $\mathcal{S}_i$ that $\mathcal{A}_i$ aligns. If $f_x(\mathcal{B}^*) \geq f_x(\mathcal{A}_i)$, then by transitivity inequality (5) holds for all $\mathcal{B}$, so x satisfies this system. On the other hand if $f_x(\mathcal{B}^*) < f_x(\mathcal{A}_i)$, this gives a violated inequality to report.

For inverse *near-optimal* alignment, we use an identical approach on inequalities (7). Note that this runs in $O(kt)$ time, where t is the time to compute an optimal alignment and k is the number of input alignments. So if t is polynomial, this separation algorithm runs in polynomial time.

Unique-Optimal Alignment. For inverse unique-optimal alignment, to separate the system of inequalities (8) associated with $\mathcal{A}_i$, we again compute an optimal alignment $\mathcal{B}^*$ of $\mathcal{S}_i$ under f_x. If $\mathcal{B}^* \neq \mathcal{A}_i$, then $f_x(\mathcal{B}^*) \geq f_x(\mathcal{A}_i) + \delta$ is a violated inequality. If $\mathcal{B}^* = \mathcal{A}_i$, compute a *next-best* alignment $\mathcal{C}^*$ of $\mathcal{S}_i$. If $f_x(\mathcal{C}^*) \geq f_x(\mathcal{A}_i) + \delta$, then by transitivity x satisfies the system; otherwise, this gives a violated inequality. Note that this runs in polynomial time if a next-best alignment can be computed in polynomial time (which is the case for standard string alignment [3]).

In the next section we use Theorem 2 to show that for *global alignment*, all variations of inverse alignment can be solved in polynomial time. The key point is showing how to bound the values of alignment parameters.

5 Application to Global Alignment

To obtain a cutting-plane algorithm for a particular form of alignment, such as global or local alignment of two strings, several details must be worked out to apply the general approach of Section 4. These include how to find an *initial subset* of the inequalities that yields a bounded linear program, and how to choose an appropriate *objective function.* Here we discuss these in the context of global alignment, but similar ideas apply to local alignment as well.

We use the definition of *standard global alignment* given at the beginning of Section 3, in which matches between pairs of letters are weighted by arbitrary substitution scores, and gaps are penalized using gap open and extension penalties. For inverse global alignment we separately consider two forms of the scoring function: when substitution scores are *varying* as given by equation (4), and when they are *fixed* as given by equation (3).

Initializing the Cutting-Plane Algorithm. Typically cutting-plane algorithms take as their initial set of inequalities just the trivial inequalities $x \geq 0$ of the linear program. For objective functions of biological interest, however, this trivial linear program is unbounded in the direction of the objective function. Consequently with this choice the first iteration of the cutting-plane algorithm would fail, as the first call to the linear programming solver to find an initial candidate solution x would report that the problem is unbounded, and return no solution.

When the substitution costs and gap penalties are all *varying*, we can set absolute upper and lower limits on the values of the parameters, and solve the linear program within the resulting bounding box as follows. By scaling the linear scoring-function by a positive factor (which does not change the relative rank of alignments), we can always make the largest parameter value hit 1. Then all parameters lie in the bounding box $0 \leq x \leq 1$, which we take as the initial set of inequalities for the cutting-plane algorithm.

When substitution costs are *fixed*, however, the bounding-box approach does not work (as the linear program may be unbounded). Instead we take the following approach. The linear programming problem is now a two-dimensional problem in the (γ, λ)-plane, where we associate γ with the vertical axis and λ with the horizontal axis. We say inequality I is a *bounding inequality* if the linear program consisting of I and the trivial inequalities $(\gamma, \lambda) \geq 0$ is bounded. In general, the linear program is bounded if and only if there exists (1) a bounding inequality, or (2) two inequalities where one is a downward halfspace, the other is an upward halfspace, and the slope of the downward inequality is less than the slope of the upward inequality. Furthermore, if they exist, these inequalities together with the trivial inequalities yield an initial set for the cutting-plane algorithm of at most four inequalities that give a bounded linear program.

We can find this set if it exists by identifying a downward inequality D of minimum slope and an upward inequality U of maximum slope. If D or U is a bounding inequality, or D's slope is less than U's, the linear program is bounded, and if not it is unbounded. For near-optimal inverse alignment, the general form of an inequality is $\gamma\,\Delta g + \lambda\,\Delta\ell \leq -\Delta s$, where $\Delta g := g(\mathcal{A}) - (1+\epsilon)\,g(\mathcal{B})$ for input alignment $\mathcal{A}$, and similarly for $\Delta\ell$ and Δs. This inequality is downward if $\Delta g > 0$, upward if $\Delta g < 0$, and its slope is $-\frac{\Delta\ell}{\Delta g}$. Thus for fixed $\mathcal{A}$ and ϵ, the direction and slope of an inequality is strictly a function of $g(\mathcal{B})$ and $\ell(\mathcal{B})$. For the two strings $\mathcal{A}$ aligns, functions g and ℓ range over a linear number of integer values, so the problem of finding a downward or upward inequality of optimal slope is certainly solvable. With further analysis, one can find the optimal inequalities in $O(1)$ time. Due to page limits we omit the details.

Choosing an Objective Function. As mentioned in Section 3, we are free to use any objective function we wish for the linear program, and we can exploit this freedom to pick a feasible solution that is biologically more desirable.

With *fixed* substitution scores, the parameters are γ and λ. Biologists generally prefer large γ and small λ, as in this regime optimal alignments tend to consist of a few long gaps, which is observed in biologically correct alignments. So one possibility for an objective is the linear combination $\max\{\gamma - \lambda\}$.

With *varying* substitution scores, the parameters are γ, λ, and all σ_{ab}. When the alignment problem seeks to minimize the alignment scoring function, so the σ_{ab} are treated as costs, we might want to maximize the separation between true substitution costs σ_{ab} (where $a \neq b$) and identity costs σ_{aa}. Then one possibility for the objective is to maximize the difference between the minimum true substitution cost and the maximum identity cost (so they are as far apart as possible). We can express this in our linear programming formulation by adding two new variables: s, which will equal the minimum true substitution cost, and i, which will equal the maximum identity cost. Using the objective $\max\{s-i\}$, and adding the inequalities $s \leq \sigma_{ab}$ for all $a \neq b$, and $i \geq \sigma_{aa}$ for all a, will achieve this goal. (Another possibility is to maximize the difference between the average true substitution cost and the average identity cost, which is also an objective that is linear in the parameters.) This objective on substitution scores can be combined with our objective on gap penalties by $\max\{s - i + \gamma - \lambda\}$.

Finally, note that for every objective, we can select two extreme solutions: x_{large}, which is the optimal solution under the objective, and x_{small}, which is the optimal solution in the direction opposite to the objective. Since the domain of feasible solutions for a linear program is convex, any convex combination of these two extremes, $x_\alpha := (1-\alpha)\, x_{\text{large}} + \alpha\, x_{\text{small}}$, where $0 \leq \alpha \leq 1$, is also a feasible solution. For example, $x_{1/2}$ may tend to be a more central parameter choice that generalizes to alignments outside the training set of input alignments $\mathcal{A}_i$ (which is borne out by our experiments of the next section).

6 Computational Results

We now present results from computational experiments on biological data with an implementation of our algorithms for inverse optimal and near-optimal global alignment. The implementation solves the problem both with *fixed* substitution scores (where $p = 2$ gap-penalty parameters are found), and with *varying* substitution scores (where for protein sequences all $p = 212$ parameters of the scoring function are simultaneously found). To solve linear programs we use the `GNU` Linear Programming Kit. For the linear programs we use the objective function $\max\{s - i + \gamma - \lambda\}$, where s and i are the minimum substitution and maximum identity costs, as described in Section 5. To find violated inequalities quickly, we maintain a queue Q of alignments $\mathcal{A}_i$ that generated a violated inequality the last time their separation algorithm was called. To find the next violated inequality, we remove an $\mathcal{A}_i$ from the front of Q, call its separation algorithm, and add it to the rear of Q if it generates another violated inequality. Figure 1 illustrates solving an instance with this implementation.

We ran several types of experiments on biological data. For the experiments, we chose six multiple sequence alignments from the `PALI` database [1] of structural protein alignments. (For each protein family in the `SCOP` protein classification database [12], `PALI` contains a multiple sequence alignment of the family based on aligning protein structures.) Table 1 describes the `PALI` families we chose, which are: T-boxes (`box`), NADH oxidoreductases (`nad`), Kunitz

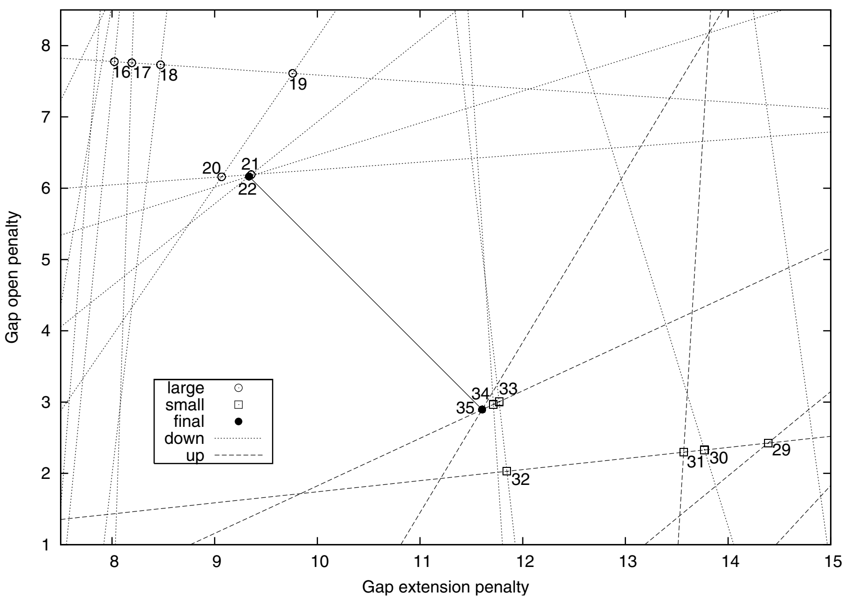

Fig. 1. Finding gap penalties by the cutting-plane algorithm. Labeled points are successive solutions for inverse near-optimal global alignment using `PAM250` substitution scores on the `pec` dataset at the best possible $\epsilon = 0.06$ (see Tables 1 and 2). Solutions found when maximizing the linear program objective $\gamma - \lambda$ are *large* points; those found when minimizing this objective are *small* points. The optimal solutions for maximization and minimization are the two *final* points. Successive violated inequalities are also plotted, along with their half-space direction of *up* or *down*. Numbers labeling points give the order in which solutions were computed. The solid line-segment between the final large and small points is the *blend* line between these extremes.

inhibitors (`kun`), Sir2 transcription regulators (`sir`), apolipoproteins (`apo`), and pectin methylesterases (`pec`). Each family was reduced to 20 members by removing outlier sequences. For the training and testing experiments described next, these 20 members were partitioned into two groups of 10 members each, called the *training set* and the *test set.*

To investigate whether it is possible to learn scoring parameters from a training set of pairwise alignments that will apply to other alignments, we ran the following experiment. For each dataset with varying substitution scores, we found the smallest ϵ for the training set such that each induced pairwise alignment on the training set is ϵ-optimal; we call this smallest value $\epsilon_{\mathrm{train}}$. We also computed the same quantity for the test set, called ϵ_{test}. Then for each training set, we computed two extreme choices of parameters at $\epsilon_{\mathrm{train}}$, x_{large} and x_{small}, which are the parameter choices that respectively maximized and minimized the linear programming objective. We then searched for the convex combination between these two extremes that yielded a parameter choice for the test set with the smallest ϵ. For a given $0 \le \alpha \le 1$, the convex combination is $\alpha \, x_{\mathrm{large}} + (1-\alpha) \, x_{\mathrm{small}}$. The α that gave the smallest ϵ for the test set is called α_{blend}, and its corresponding ϵ is called ϵ_{blend}. These values are shown for the six datasets in Table 1. The table also gives the value of ϵ for the convex combinations $\alpha \in \{0, \frac{1}{2}, 1\}$. Notice that $\epsilon_{1/2}$ is surprisingly close to ϵ_{blend}, which is within roughly one percent of ϵ_{test}. This indicates that the best blend from the training set was nearly a best possible parameter choice for the test set. Notice also on this data the central parameter $\alpha = 1/2$ always yields a smaller ϵ than the extremes $\alpha = 0, 1$, indicating that central parameters generalize better.

To get a feel for how much closer to optimal we could make the induced pairwise alignments in a dataset by varying the substitution scores versus using a standard substitution matrix, we performed the following experiment. For all 20 members of each of the PALI datasets, we computed the minimum ϵ for the set of all 190 induced pairwise alignments, with varying substitution scores and the fixed PAM [2] and BLOSUM [9] substitution scores shown in Table 2. As might

Table 1. Generalizing from training to test sets. For each multiple sequence alignment dataset listed below, best possible parameters computed on a *training* subset are applied to a disjoint *test* subset. For each dataset the table lists the PALI accession number, the average sequence length, and the average percent-identity over all induced pairwise alignments. The meaning of the closeness entries is given in the text; values for ϵ are reported as percentages. All substitution scores and gap penalties are free parameters in these experiments.

Dataset	PALI number	Sequence length	Percent identity	Closeness ϵ						
				ϵ_{train}	ϵ_{test}	ϵ_{blend}	α_{blend}	ϵ_0	$\epsilon_{1/2}$	ϵ_1
box	333	183	14.3	1.7	**1.2**	1.5	0.47	1.7	**1.5**	2.6
nad	419	151	16.1	2.8	**2.8**	3.1	0.37	3.7	**3.1**	4.7
kun	409	172	15.0	3.9	**3.7**	4.2	0.49	4.4	**4.2**	4.8
sir	633	197	16.8	3.1	**2.2**	3.0	0.55	4.3	**3.0**	4.4
apo	99	143	17.8	1.9	**1.7**	3.2	0.22	3.6	**3.4**	4.6
pec	483	299	17.0	1.2	**2.0**	3.1	0.65	4.4	**3.1**	4.4

Table 2. Closeness to optimality for fixed and varying substitution scores. For each PALI dataset and for fixed or varying substitution scores, the smallest ϵ such that all induced pairwise alignments are ϵ-optimal is reported as a percentage.

Dataset	Closeness ϵ				
	varying	fixed			
		PAM250	PAM120	BL045	BL080
box	**2**	**5**	9	8	9
nad	**4**	**9**	15	12	18
kun	**5**	**8**	11	9	12
sir	**4**	**11**	16	12	16
apo	**3**	**21**	45	34	58
pec	**3**	**6**	9	8	11

be expected, the minimum ϵ when substitution scores are free parameters is smaller than the minimum ϵ for a fixed substitution matrix. Notice that on these datasets PAM250 gave the smallest ϵ of the four matrices considered. On the other hand, the optimal substitution matrix found when substitution scores were free parameters has roughly at most half the ϵ for PAM250, and is sometimes much better. Notice also that with varying substitution scores, one can consistently come very close to optimal on every dataset.

Finally, Table 3 gives running times and number of violated inequalities for the cutting-plane algorithm on the experiments of Table 2. Running times are wall-clock times in seconds to solve Inverse Near-Optimal Alignment for a given ϵ during the binary search for the smallest ϵ. Each binary search concluded in at most 8 iterations. Every iteration, while considering an input of 190 alignments, finished in roughly under a minute for fixed substitution scores, and under roughly 4 minutes for varying substitution scores while finding values for 212 parameters. Experiments were on a 3 GHz Pentium 4 with 1 GB of RAM.

Table 3. Running time and number of violated inequalities. For each PALI dataset, times and number of inequalities are reported for computing the ϵ of Table 2. Columns report the median and extreme values across the binary search iterations.

Dataset	Time (sec)				Violated inequalities					
	fixed		varying		fixed			varying		
	med	max	med	max	min	med	max	min	med	max
`box`	24	25	5	123	4	11	23	5	32	972
`nad`	17	18	7	46	3	12	21	5	118	590
`kun`	23	30	11	83	2	13	22	7	176	681
`sir`	29	31	13	96	2	11	20	5	196	832
`apo`	16	16	14	264	6	17	22	13	236	1398
`pec`	63	65	23	226	2	10	20	3	238	1087

7 Conclusion

We have presented a new approach to inverse parametric sequence alignment. The approach is actually quite general, and solves inverse parametric optimization in polynomial time for *any* optimization problem (not just sequence alignment) whose objective function is linear in its parameters, whose parameters can be bounded, and that can be solved in polynomial time when all parameters are fixed. Experiments on structural alignments from a protein family database show we can find all 212 parameters of the standard protein-sequence scoring-model from hundreds of pairwise alignments in a few minutes of computation.

Many lines of investigation remain open. Can parameter values be learned from both positive and *negative* examples? Can our algorithm aid the *evaluation* of alignment software by testing programs at parameter settings that make benchmark alignments score as close to optimal as possible? Can cutting planes be efficiently found for inverse alignment that are *facet-defining* inequalities?

References

1. Balaji, S., S. Sujatha, S.S.C. Kumar and N. Srinivasan. "`PALI`: a database of alignments and phylogeny of homologous protein structures." *Nucleic Acids Research* 29:1, 61–65, 2001.
2. Dayhoff, M.O., R.M. Schwartz and B.C. Orcutt. "A model of evolutionary change in proteins." In M.O. Dayhoff, editor, *Atlas of Protein Sequence and Structure* 5:3, National Biomedical Research Foundation, Washington DC, 345–352, 1978.
3. David Eppstein. "Finding the k shortest paths." *SIAM Journal on Computing* 28:2, 652–673, 1998.
4. Fernández-Baca, D., T. Seppäläinen and G. Slutzki. "Bounds for parametric sequence comparison." *Discrete Applied Mathematics* 118:3, 181–192, 2002.
5. Griggs, J.R., P. Hanlon, A.M. Odlyzko and M.S. Waterman. "On the number of alignments of k sequences." *Graphs and Combinatorics* 6, 133–146, 1990.
6. Grötschel, M., L. Lovász and A. Schrijver. "The ellipsoid method and its consequences in combinatorial optimization." *Combinatorica* 1, 169–197, 1981.
7. Gusfield, Dan, K. Balasubramanian and Dalit Naor. "Parametric optimization of sequence alignment." *Algorithmica* 12, 312–326, 1994.
8. Gusfield, Dan and Paul Stelling. "Parametric and inverse-parametric sequence alignment with `XPARAL`." *Methods in Enzymology* 266, 481–494, 1996.
9. Henikoff, S. and J.G. Henikoff. "Amino acid substitution matrices from protein blocks." *Proceedings of the National Academy of Sciences USA* 89, 10915–10919, 1992.
10. Karp, R.M. and C.H. Papadimitriou. "On linear characterization of combinatorial optimization problems." *SIAM Journal on Computing* 11, 620–632, 1982.
11. Kececioglu, John and Dean Starrett. "Aligning alignments exactly." Proceedings of the 8th ACM *Conference on Research in Computational Molecular Biology*, 85–96, 2004.
12. Murzin, A.G., S.E. Brenner, T. Hubbard and C. Chothia. "`SCOP`: a structural classification of proteins database for the investigation of sequences and structures." *Journal of Molecular Biology* 247, 536–540, 1995.
13. Pachter, Lior and Bernd Sturmfels. "Parametric inference for biological sequence analysis." *Proceedings of the National Academy of Sciences USA* 101:46, 16138–16143, 2004.
14. Padberg, M.W. and M.R. Rao. "The Russian method for linear programming III: bounded integer programming." Technical Report 81-39, Graduate School of Business and Administration, New York University, 1981.
15. Sun, Fangting, David Fernández-Baca and Wei Yu. "Inverse parametric sequence alignment." *Journal of Algorithms* 53, 36–54, 2004.
16. Waterman, M., M. Eggert and E. Lander. "Parametric sequence comparisons." *Proceedings of the National Academy of Sciences USA* 89, 6090–6093, 1992.
17. Zimmer, Ralf and Thomas Lengauer. "Fast and numerically stable parametric alignment of biosequences." Proceedings of the 1st ACM Conference on *Research in Computational Molecular Biology*, 344–353, 1997.