INVERSE PARAMETRIC ALIGNMENT FOR ACCURATE BIOLOGICAL SEQUENCE COMPARISON

by

Eagu Kim

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF COMPUTER SCIENCE

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIZONA

2 0 0 8

THE UNIVERSITY OF ARIZONA

GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation prepared by Eagu Kim entitled Inverse Parametric Alignment for Accurate Biological Sequence Comparison and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

_____ Date: 08/06/08

John Kececioglu

_____ Date: 08/06/08

Peter Downey

_____ Date: 08/06/08

Stephen Kobourov

_____ Date: 08/06/08

Robert Indik

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.
I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

_____ Date: 08/06/08

Dissertation Director: John Kececioglu

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Eagu Kim

ACKNOWLEDGMENTS

It is hard to talk about anything about my Ph.D. experience with my advisor, John Keciceoglu. I first met him at the Design and Analysis of Algorithms class during the autumn of year 2000. He was a genuinely gentle, kind, and caring human being and precise, accurate, and yet delivering instructor (and still is). In the following semester he introduced to me computational molecular biology, which was a new, interesting, and promising world to me. I cannot forget the moment.

We started working together as an advisor and a student. He was the first scholar whom I met and who is trying to be precise in defining problems, persistent in solving problems, divergent in approaching to the problem, and honest in interpreting the outcomes. I can only wish that I keep up with what he has shown to me. Of course, he is one of the smartest scientists that I ever met to this moment.

A few years later, we became friends. He was (and still is) kind as a friend and yet harsh as an advisor to his students when it is necessary. There were many nights we went out for movies and dinners to celebrate all sorts of things.

This dissertation would not exist without him. He introduced inverse alignment to me, directed the whole research, and participated in every step with great detail. I am truly in a great debt for that. I also thank Chuong Do and Dan Gusfield for helpful discussion on inverse alignment and parameter learning.

I cannot express enough my gratitudes to my dissertation committee members: Peter Downey and Stephen Kobourov of Computer Science Department, and Robert Indik of Mathematics Department. All of them raised excellent questions to what I once thought were done and made valuable comments on what I believed could not be improved.

Dean Starrett, Cesim Erten, and Travis Wheeler are friends who helped me through my Ph.D. program. Dean Starrett is the one who suggested to me to work with John K. while we were looking for an advisor. He is the most gentle and modest man ever known to me, regardless of nationality. Without his tool `AlignAlign`, I could not finish this work. Cesim Erten is the one who showed me how to approach difficult things with ease. Travis Wheeler is a friend who comforted me with his wits during the darkest hours. He was always kind enough to share his broad knowledge on biology with me. He assisted me with his tool `Opal` for my dissertation.

DEDICATION

To my father who showed me how to love living things and serve people

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

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. In practice, substitution scores are usually chosen by convention, and gap penalties are often found by trial and error. In contrast, a rigorous way to determine parameter values that are appropriate for aligning biological sequences is by solving the problem of Inverse Parametric Sequence Alignment. Given examples of biologically correct reference alignments, this is the problem of finding parameter values that make the examples score as close as possible to optimal alignments of their sequences. The reference alignments that are currently available contain regions where the alignment is not specified, which leads to a version of the problem with partial examples.

In this dissertation, we develop a new polynomial-time algorithm for Inverse Parametric Sequence Alignment that is simple to implement, fast in practice, and can learn hundreds of parameters simultaneously from hundreds of examples. Computational results with partial examples show that best possible values for all 212 parameters of the standard alignment scoring model for protein sequences can be computed from 200 examples in 4 hours of computation on a standard desktop machine. We also consider a new scoring model with a small number of additional parameters that incorporates predicted secondary structure for the protein sequences. By learning parameter values for this new secondary-structure-based model, we can improve on the alignment accuracy of the standard model by as much as 15% for sequences with less than 25% identity.

CHAPTER 1

INTRODUCTION

In this dissertation we present and solve a problem called *Inverse Parametric Sequence Alignment*, which is informally defined as follows. Given examples of biologically correct alignments and a cost function that scores a given alignment, find values for the parameters of the cost function that make the input examples all be optimal alignments under the cost function with the parameter values. In the following we explain why this problem is important in computational molecular biology and how one might approach this problem. We then mention our main contribution on this problem, and close the chapter with the organization of the rest of this dissertation.

1.1 Perspectives

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. For biologists this is an important problem because they wish to find similarity between the biological sequences through sequence alignment for better understanding of functional and evolutionary relationship among the sequences, and the quality of the resulting alignment is a function of alignment parameter values such as substitution and gap costs. While some choices for substitution costs are now common, largely due to convention, there is no standard for choosing gap costs.

An objective solution to this question is important for several reasons. First, without a systematic way of finding gap and substitution costs simultaneously in which the quality of the alignment parameter values is guaranteed in a certain way, the resulting alignment obtained with ad-hoc gap costs cannot be trusted. Second,

even if one finds a choice of gap parameter values which optimizes a certain objective together with a fixed standard substitution cost matrix, such a combination is not necessarily the best under the objective because the conventionally-used substitution matrices could be optimized for a different objective. Third, even if one finds the best alignment by trying every possible choice of alignment parameter values or by picking carefully with hand, one cannot repeat the same procedure every time when newly-sequenced biological molecules are aligned, because the number of such sequences is ever-growing and the time needed for such a procedure is nontrivial.

1.2 Contributions

An objective way to solve this question is to learn appropriate parameter values by solving the *Inverse Parametric Sequence Alignment* problem, or more simply, *Inverse Alignment*. This problem, whose formal definition is given in the next chapter, is informally defined as follows. Given examples of biologically correct alignments, and a cost function that scores a given input alignment, find values for the parameters of the cost function that make the examples all be optimal alignments under the cost function with the parameter values.

Inverse Alignment was introduced by Gusfield and Stelling [26] in their seminal paper. They considered the problem for the case of two alignment parameters and one alignment, and sought parameter values that made the input alignment be an optimal alignment of its sequences. For this version of Inverse Alignment they presented an indirect approach that attempted to avoid computing the complete parametric decomposition of the parameter space [59, 25, 46].

Later, Sun, Fernandez-Baca, and Yu [55] gave the first direct algorithm for Inverse Alignment for the case of three parameters and one alignment of a pair of sequences. (While they consider three parameters, their solution effectively fixes one parameter value at zero.) Given two sequences of length n, their algorithm finds three parameter values that make the input alignment optimal in $\mathrm{O}(n^2 \lg n)$

time. Their algorithm is involved, and does not appear to have been implemented.

In a significant advance, Kececioglu and the author [32], using a completely different approach based on linear programming, gave the first polynomial-time algorithm for arbitrarily many parameters and alignments. This approach is flexible and actually solves the more general problem of *inverse parametric optimization*. The author and Kececioglu [34, 36] later extended this work in several directions, including to partial examples, which contain regions where the reference alignment is not specified.

We summarize our contributions in this dissertation as follows.

- We show the existence of an algorithm for the inverse alignment problem with an arbitrary number of alignment parameters and an arbitrary number of examples, that runs in *polynomial time* in both input measures.

- We give and implement a practical algorithm for the inverse alignment problem that runs *fast* for examples that are biological protein sequences.

- We rigorously address the issue of *partial examples*, which contain regions where the reference alignment is not specified, and present an iterative algorithm that learns parameter values from them.

- We present new alignment cost models that incorporate predicted *secondary structure* of protein sequences to further improve the accuracy of protein sequence alignment.

- We give a polynomial time algorithm for the general *inverse parametric optimization* problem when the objective function for the problem is linear in its parameters, and the problem can be solved in polynomial time for any fixed choice of its parameters.

The organization of the rest of this dissertation is as follows. In Chapter 2, we present and motivate three different formulations of the inverse alignment problem:

Inverse Optimal, Unique-Optimal, and Near-Optimal Alignments. In Chapter 3, we assume that the cost function that scores alignments is linear in its parameters, and show how each variation of inverse alignment can be reduced to linear programming. In Chapter 4, for each formulation of inverse alignment, we present a theoretical algorithm whose running time is a polynomial in the number of alignment parameters and examples, and a practical algorithm using the cutting plane approach to linear programming. In Chapter 5, we extend the inverse alignment problem to partial example, and give a practical iterative algorithm. In Chapter 6, we present new alignment cost models that incorporate predicted secondary structure of the protein sequences to further improve the alignment accuracy. In Chapter 7, we give experimental results with biological alignments from three different pieces of work on inverse alignment. Finally in the last chapter, we conclude the dissertation with some problems for further research.

CHAPTER 2

INVERSE ALIGNMENT AND ITS VARIATIONS

We introduce the notion of sequence alignment, alignment cost function and its parameters, and optimal alignment under a cost function. We then present a problem called *Inverse Parametric Sequence Alignment*, which finds a choice of parameter values that makes the input alignment an optimal alignment under the input alignment cost function with the choice. We present three different formulations of Inverse Alignment along with motivations: Inverse Optimal, Unique-Optimal, and Near-Optimal Alignments. We introduce three variations of Inverse Near-Optimal Alignment: the absolute error criterion, the relative error criterion, and the discrepancy error criterion.

2.1 Sequence alignment

The notion of an alignment

One of the most fundamental and important tasks in computational molecular biology is *sequence alignment*. The task of sequence alignment is to arrange biological sequences in such a way that similar regions across the input biological sequences can be easily identified. Biologists use sequence alignment to study and understand the functional, structural, or evolutionary relationship among these sequences by comparing aligned regions. A formal definition of sequence alignment follows.

Definition 2.1 (Alignment of k sequences) Given sequences $S_1, \ldots, S_k$ for $k \geq 2$ defined over an alphabet Σ that does not include a character -, an *alignment* $\mathcal{A}$ of $S_1, \ldots, S_k$ is an $m \times k$ character matrix $\{c_{ij}\}$ for $1 \leq i \leq m$ and $1 \leq j \leq k$,

```
K--EYSRTVVMQSSITNVINPAGWFPW--------DGNFA-LDTLYYGEYQNTGAGA-AT
G--AYATVVFSNTYMSGIITPEGWNNW------GDST---KEKTVTFGEHKCYGPGADYK
K--PFSRVVVMESYLGAGVQPRGWLEW-----DGDGG---ELATLFYGEYRNYGPGANIG
K--LFSRTVYMMSYIGGHVHTRGWLEW-------NTT-FA-LDTLYYGEYLNSGPGSGLG
K--LYSRVVYMMSDMGDHIDPRGWLEW--------NGPFA-LDSLYYGEYMNKGLGSGIG
DPNAIGQTVFLNTSMDNHI--YGWDKMSGKDKNGNTI-WFNPEDSRFFEYKSYGAGAAVS
K--LYSRTVFIRNNMSDVVRPEGWLEW--------NADFA-LDTLFYGEFMNYGPGSGLS
K--EHSRTVVMQSSISDVINRAGWLEW--------RGKYA-LNTLYYGEYNNSGAGA-AT
K--EYSRTVIMQSSISDVISPAGWREW--------KGRFA-LNTLHFAEYENSGAGA-GT
K--KYSRTVVLQSVVDSHIDPAGWAEW---DAASKDF----LQTLYYGEYLNSGAGA-GT
```

Figure 2.1: A multiple alignment of protein sequences.

```
KEYSRTVVMQSSITNVINPAGWFPW--DGNFA-LDTLYYGEYQNTGAGA-AT
GAYATVVFSNTYMSGIITPEGWNNWGDST---KEKTVTFGEHKCYGPGADYK
```

Figure 2.2: An induced pairwise alignment.

---

where $c_{ij} \in \Sigma \cup \{\texttt{-}\}$, no column of $\mathcal{A}$ consists of only `-`, and removing all `-` from the ith row of $\mathcal{A}$ gives sequence S_i. □

The special character '`-`' in Definition 2.1, which may appear in an alignment but not in input sequences, is called a *space*. A column consisting of spaces only is called a *space column* whereas a column having no space is called a *substitution column*. Note that an alignment must have no space column. An alignment formed by taking at least two rows from an alignment $\mathcal{A}$ and then removing all space columns from the result matrix is called an *induced alignment on* $\mathcal{A}$.

When the number of sequences in an alignment should be stressed, an alignment with two sequences is called a *pairwise alignment*, and an alignment with more than two sequences is called a *multiple alignment*.

Figure 2.1 shows an example of a multiple alignment of 10 sequences in the protein family with `SCOP` [43, 44] identifier `b.80.1.5` in `PALI` benchmark suite [5]. For simplicity only a part of the original is shown. Each sequence in the alignment is defined over an alphabet of twenty amino acids. The alignment has 50 columns

and the dash character '-' represents a space. The first column of the alignment is a substitution column because there is no - character in the column. Note that the alignment has no space column. If we take the first two rows and remove all 8 space columns from the resulting matrix, we obtain an induced alignment shown in Figure 2.2. Since this induced alignment aligns two sequences, it is also called a pairwise induced alignment.

Consider the jth column $c_{1j}, \ldots, c_{kj}$ of alignment $\mathcal{A}$ of k sequences. A pair of distinct members in the column, say (c_{pj}, c_{qj}) for $1 \leq p, q\,(\neq p) \leq k$ is chosen. If $c_{pj} \neq c_{qj}$ and both are not space, then the pair is called a *substitution*. If $c_{pj} = c_{qj}$ and both are not space, then the pair is called an *identity*. When the distinction between substitution and identity is not necessary, we simply call the pair a substitution. If $c_{pj} =$ - and $c_{qj} \neq$ -, then c_{pj} is said *deleted* from sequence S_p or c_{qj} is said *inserted* in sequence S_q. Given two different rows in an alignment, a maximal run of deleted characters and a maximal run of inserted characters are called a *gap*.

For example, in Figure 2.2 the first column is a substitution and the third column is an identity. In the third last column, `D` is inserted or an unknown letter above it is deleted and represented as a space. There are three gaps in the alignment shown in the figure.

An alignment defined in Definition 2.1 is sometimes called a *global alignment* to stress the fact that the whole of each input sequence is in the alignment. A *local alignment* of sequences $S_1, \ldots, S_k$ for $k \geq 2$ is an alignment of a set of sequences $\overline{S}_1, \ldots, \overline{S}_k$, where $\overline{S}_i$ is a substring of S_i for $1 \leq i \leq k$.

Scoring an alignment

Note that by Definition 2.1, for given sequences, the number of alignments that align the input sequences is exponential in the length of sequences [21]. Suppose a pair of sequences of length n are aligned. The number of alignments of this pair of sequences is $\Theta((3+\sqrt{2})^n/\sqrt{n}) = \Omega(4^n)$. This is an exponential number in the

length of sequences that are aligned.

To objectively compare many different alignments that align the same sequences, one may compute *cost* of alignments (or *score* the alignments) and judge the quality of alignments based on their costs. A function that maps an alignment to a non-negative number for this purpose is called an alignment *cost function*. Throughout this dissertation we assume that an alignment of sequences with *smaller* cost is better than other alignments that align the same sequences.

The characteristics of an alignment that an alignment cost function uses are called alignment *features*. Each feature should be quantitatively measurable through a function. A function that measures a characteristic of an alignment is called a *feature measuring function*.

The variables that control the mapping behavior of an alignment cost function are called alignment *parameters*. Let f be a cost function and $w = (w_1, \ldots, w_t)$ be a vector of alignment parameters. When the dependence of alignment cost function f on its alignment parameters $w_1, \ldots, w_t$ should be stressed, we write $f_{w_1,\ldots,w_\ell}$ or simply f_w. When such stress is not necessary, we simply write f. The number of feature measuring functions and the number of parameters for an alignment cost function are not necessarily the same. The constants that are assigned to the parameters are called parameter *values*. An alignment parameter is called *fixed* if it is assigned a value, and *free* otherwise. All parameters for an alignment cost function must be fixed before the cost function is used.

The alignment cost function considered in this dissertation is the standard alignment cost function, which

- is a *linear function* of its parameters and
- scores a *pairwise alignment*.

The standard linear cost function for pairwise alignments considers substitution and gap parameters. A substitution parameter σ_{ab} is the parameter charged for a

substitution involving the unordered pair of letters a and b in a certain alphabet. A *gap-open* parameter γ is the parameter charged per gap and a *gap-extension* parameter λ is the parameter charged per letter in a gap. Under this model a gap of length ℓ costs $\gamma + \lambda\ell$. In Section 3.1 we present a concrete example of the standard alignment cost function for global pairwise alignments of protein sequences with linear gap costs in detail.

Finding an optimal alignment

Given an alignment cost function f, an alignment with the minimum cost under f is of interest. An alignment $\mathcal{A}$ is called an *optimal* alignment under f if

$$f(\mathcal{A}) \quad \leq \quad f(\mathcal{B}), \tag{2.1}$$

where $\mathcal{B}$ is any alignment of the sequences that $\mathcal{A}$ aligns. Finding an optimal alignment leads to the following classic problem.

Problem 2.1 (Optimal Alignment) Given a set of sequences $S_1, \ldots, S_k$, and an alignment cost function f, find an optimal alignment $\mathcal{A}$ of sequences $S_1, \ldots, S_k$ under f. □

The original version of this problem, which is finding an optimal global alignment of two sequences, was discussed and solved by Needleman and Wunsch [45]. Their algorithm finds an optimal global alignment in $O(n^3)$ time via dynamic programming, where n is the length of the sequences. The gap cost that they considered is called *arbitrary* gap cost because the gap-extension parameter can be any function of the gap length.

A decade later a more practical and efficient algorithm that finds an optimal alignment of a pair of sequences with *linear* gap costs in $O(n^2)$ time was found [19, 17, 1]. In the linear gap cost model, as mentioned earlier, a gap of length ℓ costs $\gamma +$

$\lambda\ell$ where γ and λ are respective gap-open and gap-extension parameters. The linear gap cost is most commonly used for alignment of protein sequences by biologists.

Waterman [58] argued and proposed a better gap cost model, which is called *convex* gap costs. In this model the gap-extension parameter is a convex function of gap length. In words, as a gap length increases, the additional cost charged for an additional letter in the gap does not increase. Miller and Myers [41] and independently Galil and Giancarlo [18] found an algorithm that finds an optimal alignment of two sequences with convex gap costs in $O(n^2 \lg n)$ time using candidate-list approach.

We give a description of an algorithm that finds an optimal alignment of a pair of sequences under a given cost function with linear gap costs. This algorithm uses dynamic programming and finds a solution to Optimal Alignment in $O(mn)$ time, where m and n are the sequence lengths. Hence this is an algorithm whose running time is polynomial in the length of input sequences.

Algorithm 2.1 (Algorithm for Optimal Alignment) Given two sequences S and T of lengths m and n, consider prefixes $S[1:i]$ of S and $T[1:j]$ of T. An alignment of these two prefixes ends with either

- a substitution involving $S[i]$ and $T[j]$,
- a gap involving $S[k:i]$ for some $k \leq i$, or
- a gap involving $T[k:j]$ for some $k \leq j$.

These are the only possibilities for the ending of an alignment of the prefixes. In any case, the alignment is preceded by an optimal alignment of shorter prefixes.

We define, for each of three types, a variable or *cell* that keeps the minimum cost of all alignments of the prefixes that end with the type: $D(i,j)$ for substitution, $V(i,j)$ for deletion, and $H(i,j)$ for insertion. A cell $M(i,j)$ is defined to be the minimum of $D(i,j), V(i,j)$, and $H(i,j)$. These four cells together form an *entry* in a dynamic programming *table*. The definitions of these cells lead to general

recurrences. We give the recurrence for M.

$$M(i,j) \;:=\; \min\{D(i,j),\, V(i,j),\, H(i,j)\}, \quad \text{for } 0 \le i \le m \text{ and } 0 \le j \le n.$$

The recurrences for D, V, and H are

$$\begin{aligned} D(i,j) &:= M(i-1,j-1) \;+\; \sigma(S[i],T[j]), \\ V(i,j) &:= \min\Big\{M(i-1,j)+\gamma,\, V(i-1,j)\Big\} \;+\; \lambda, \quad \text{and} \\ H(i,j) &:= \min\Big\{M(i,j-1)+\gamma,\, H(i,j-1)\Big\} \;+\; \lambda, \end{aligned} \tag{2.2}$$

and their boundary conditions are

$$\begin{cases} D(0,0) := 0 \\ V(0,0) := \infty, \\ H(0,0) := \infty \end{cases} \quad \begin{cases} D(i,0) := \infty \\ V(i,0) := \gamma + i\,\lambda, \\ H(i,0) := \infty \end{cases} \quad \text{and} \quad \begin{cases} D(0,j) := \infty \\ V(0,j) := \infty \\ H(0,j) := \gamma + j\,\lambda \end{cases},$$

for $1 \le i \le m$ and $1 \le j \le n$, where $\sigma(S[i],T[j])$ is the substitution cost involving an unordered pair of letters $S[i]$ and $T[j]$, and γ and λ are the gap-open and gap-extension costs respectively.

The cost of an optimal alignment of S and T is obtained from cell $M(m,n)$ and the alignment itself is discovered by tracking back from entry (m,n) to $(0,0)$. The above recurrences can be evaluated by filling out each entry of the dynamic programming table in row-major order in $\mathrm{O}(mn)$ time because $\mathrm{O}(1)$ time is required to fill out each entry in the table of $(m+1) \times (n+1)$ entries. Finding an optimal alignment takes $\mathrm{O}(m+n)$ time. Therefore the algorithm takes time polynomial in the length of sequences to find a solution to Optimal Alignment. □

Given an alignment cost function, an alignment whose cost is as good as all the other alignments that align the same sequences (except possibly an optimal alignment) is called a *next-best* alignment under the cost function. Finding a next-best alignment of two sequences under a given alignment cost function with linear gap costs can be solved in $\mathrm{O}(n^2)$ time by modifying the above algorithm. The description of the algorithm is given in Section 4.1.2.

2.2 Inverse Parametric Sequence Alignment

In Optimal Alignment, a set of sequences and an alignment cost function f_w with alignment parameters w are given, and an optimal alignment under f_w is sought. An inverse of this problem is the following. Given an alignment and an alignment cost function f_w with alignment parameters w, find a choice x of parameter values for w that makes the input alignment optimal under f_w. This problem, which we formally define later, is called *Inverse Parametric Sequence Alignment* [26, 55, 32] or simply *Inverse Alignment*.

Inverse Alignment was introduced by Gusfield and Stelling [26] in their seminal paper. They considered the problem for the case of two alignment parameters and one alignment, and sought parameter values that made the input alignment be an optimal alignment of its sequences. For this version of Inverse Alignment they presented an indirect approach that attempted to avoid computing the complete parametric decomposition of the parameter space [59, 25, 46]. The parametric decomposition is a partition of the parameter space such that an alignment $\mathcal{A}$ which is optimal under *one* choice of parameter values in a partitioned region $\mathcal{R}$ is optimal under *all* choices of parameter values in $\mathcal{R}$, and $\mathcal{R}$ cannot be made larger without $\mathcal{A}$ losing its property of optimality.

Later Sun, Fernandez-Baca, and Yu [55] gave the first direct algorithm for Inverse Alignment for the case of three parameters and an alignment of a pair of sequences. While they consider three parameters, their solution effectively fixes one parameter value at zero. Given two sequences of length n, their algorithm finds three parameter values that make the input alignment optimal in $\mathrm{O}(n^2 \lg n)$ time. Their approach is involved and does not appear to have been implemented.

In a significant advance, Kececioglu and Kim [32], using a completely different approach based on linear programming, gave the first polynomial time algorithm for arbitrarily many parameters and alignments. This approach is flexible and ac-

tually solves the more general problem of *inverse parametric optimization*. For a combinatorial optimization problem $\mathcal{P}$, inverse parametric optimization seeks parameter values for the objective function of $\mathcal{P}$ that make a given example solution to an instance $\mathcal{I}$ of $\mathcal{P}$ be an optimal solution to $\mathcal{I}$. This approach solves the inverse parametric problem in polynomial time for any problem $\mathcal{P}$ where: (1) the objective function for $\mathcal{P}$ is linear in its parameters, and (2) problem $\mathcal{P}$ can be solved in polynomial time for any fixed choice of its parameters. This solves inverse parametric optimization for an extremely wide range of problems $\mathcal{P}$, including many classic problems such as sequence alignment, shortest paths, and network flow. Eppstein [16] also discovered a similar approach to general inverse parametric optimization. Eppstein applied it in the context of minimum spanning trees to find parameter values for edge weights that make an example tree be the unique optimal spanning tree. In the context of biological sequence alignment, this unique-optimal form of the inverse problem rarely has a solution [32].

In the following three sections we present three different formulations of Inverse Alignment called Inverse Optimal, Unique-Optimal, and Near-Optimal Alignments. All address arbitrary numbers of alignments and alignment parameters. Throughout this dissertation, we call the input alignments to inverse alignment *examples*. The input to each problem is the same:

- a set of examples $\mathcal{A}_i$, each of which aligns sequences in a set $\mathcal{S}_i$, for $1 \leq i \leq k$,
- an alignment cost function f_w with alignment parameters w, and
- a parameter domain D.

2.2.1 Inverse Optimal Alignment

We present the most fundamental version of inverse alignment, called Inverse Optimal Alignment, that finds a choice of parameter values that makes the examples to be all *optimal* alignments under the choice, if such a choice exists.

Problem 2.2 (Inverse Optimal Alignment) Given a set of examples $\mathcal{A}_i$, each of which aligns sequences in a set $\mathcal{S}_i$ for $1 \leq i \leq k$, an alignment cost function f_w with alignment parameters w, and a parameter domain D, find a choice x of parameter values for w in D such that each example $\mathcal{A}_i$ is an optimal alignment of its sequences in $\mathcal{S}_i$ under f_x. □

Note that a solution of Inverse Optimal Alignment does not necessarily make the examples all be *uniquely* optimal; there may exist another choice for parameter values that makes the examples all be optimal alignments under that choice. Also note that Inverse Optimal Alignment may have *no* solution; it is possible that no choice for parameter values makes the examples all be optimal. This situation arises when the intersection of all choices of parameter values which make the examples all be optimal becomes empty. An algorithm for Inverse Optimal Alignment must detect this situation, and report that no solution exists. We explain the reason that Inverse Optimal Alignment may not have a solution in detail in Section 2.2.3.

We give theoretical and practical algorithms for Inverse Optimal Alignment in Chapter 4, and demonstrate in Chapter 7 that it is common for Inverse Optimal Alignment with examples of real biological sequences to have no solution.

2.2.2 Inverse Unique-Optimal Alignment

In some applications of inverse alignment we may need to find a choice of parameter values that makes a given alignment be the *unique-optimal* alignment of its sequences. This can arise, for example, in the following setting. Suppose we want to run computational experiments to test how well an *exact* algorithm for Optimal Alignment (that always finds an optimal alignment under given alignment cost function) performs at recovering benchmark alignments, compared to a *heuristic* for the same problem (that may fail to find an optimal alignment). If one can find a choice of parameter values such that the benchmarks are the unique-optimal alignments of their sequences under the choice, then the exact algorithm will recover the

benchmarks perfectly. With this choice, any discrepancy between the alignment computed by the heuristic and the benchmark reflects precisely how much worse the heuristic performs due to failing to optimize the alignment cost function. For an alignment to be unique-optimal, every other alignment of the same sequences must cost more. We quantify how much more in cost as follows.

Definition 2.2 (Unique Optimality) An alignment $\mathcal{A}$ is called α *unique-optimal* under alignment cost function f for a positive real number $\alpha > 0$ if

$$f(\mathcal{A}) \;\leq\; f(\mathcal{B}) \;-\; \alpha, \tag{2.3}$$

where $\mathcal{B}$ is any alignment other than $\mathcal{A}$ of the sequences that $\mathcal{A}$ aligns. □

Note that if alignment $\mathcal{A}$ is α unique-optimal under alignment cost function f, then $\mathcal{A}$ is $\overline{\alpha}$ unique-optimal under f for $0 < \overline{\alpha} \leq \alpha$. Therefore given alignment $\mathcal{A}$, we may want to find the *largest* value of α together with a choice x of parameter values that makes alignment $\mathcal{A}$ be α unique-optimal under f_x.

Problem 2.3 (Inverse Unique-Optimal Alignment) Given a set of examples $\mathcal{A}_i$, each of which aligns sequences in a set $\mathcal{S}_i$ for $1 \leq i \leq k$, an alignment cost function f_w with alignment parameters w, and a parameter domain D, find a choice x of parameter values for w in D such that each example $\mathcal{A}_i$ is an α unique-optimal alignment of its sequences in $\mathcal{S}_i$ under f_x for a positive number $\alpha > 0$ and α is maximized. □

Note that Inequality (2.3) is more strict than Inequality (2.1) for the same alignment $\mathcal{A}$. This implies that a solution x for Inverse Unique-Optimal Alignment with examples is a solution for Inverse Optimal Alignment with the same examples. It also implies that whenever there is no solution for Inverse Optimal Alignment with examples, a solution for Inverse Unique-Optimal Alignment with the same examples does not exist.

We give theoretical and practical algorithms for Inverse Unique-Optimal Alignment in Chapter 4, and demonstrate in Chapter 7 that it is common for Inverse Unique-Optimal Alignment with examples of real biological sequences to have no solution.

2.2.3 Inverse Near-Optimal Alignment

The reason that Inverse Optimal Alignment may not have a solution follows. Suppose we solve the problem with k examples. Let $\mathcal{W}_i$ be the set of choices of parameter values such that example $\mathcal{A}_i$ is optimal under alignment cost function f_x for every $x \in \mathcal{W}_i$ for $i = 1, \ldots, k$. The set of parameter value choices that make the examples all be optimal is obtained by taking the intersection of $\mathcal{W}_1, \ldots, \mathcal{W}_k$. The result can easily be empty. The reason for Inverse Unique-Optimal Alignment is similar.

When it is not possible to find a choice for parameter values that makes examples all be optimal, we may seek the next best thing, which is a choice of parameter values that makes the examples all be *near-optimal*. This problem is called Inverse Near-Optimal Alignment.

Given a set of examples and a choice of parameter values, the closeness to optimality of the examples is different depending on how we measure the closeness to optimality or *errors* of the examples under the choice. This section presents three different variations of Inverse Near-Optimal Alignment, each of which has an different *error measure function*. Given a set of examples and a choice of parameter values, the error function measures a nonnegative error of the examples under the choice. Each variation of Inverse Near-Optimal Alignment minimizes its error measure function while it finds a choice of parameter values. Unlike Inverse Optimal and Unique-Optimal Alignments, Inverse Near-Optimal Alignment has a solution for a large enough error.

Theoretical and practical algorithms for each variation of Inverse Near-Optimal

Alignment are given in Chapter 4, and of the practical algorithms on protein sequence alignments are given in Chapter 7.

Absolute error criterion

The first variation of Inverse Near-Optimal Alignment is called Inverse Alignment under Absolute Error or simply, the absolute error criterion. Given a set $\mathbf{A}$ of alignments and an alignment cost function f_x with a choice x of parameter values, the error function for this problem is the *average absolute error* of alignment costs $f_x(\mathcal{A})$ for $\mathcal{A} \in \mathbf{A}$ with respect to the optimal costs $f_x(\mathcal{A}^*)$ where $\mathcal{A}^*$ is the optimal alignment under f_x of the sequences that $\mathcal{A}$ aligns. This function can be written as follows.

Definition 2.3 (Error measure E_{abs}) Given a set $\mathbf{A}$ of alignments and an alignment cost function f_x with a choice x of parameter values, the error function E_{abs} of set $\mathbf{A}$ under the choice x is:

$$E_{\text{abs}}(x, \mathbf{A}) \;:=\; \frac{1}{|\mathbf{A}|} \sum_{\mathcal{A} \in \mathbf{A}} \left(f_x(\mathcal{A}) \;-\; f_x(\mathcal{A}^*) \right), \tag{2.4}$$

where $|\mathbf{A}|$ is the number of alignments in set $\mathbf{A}$, and $\mathcal{A}^*$ is an optimal alignment under f_x of the sequences that $\mathcal{A}$ aligns. □

The cost difference inside the summation in Equation (2.4) is the absolute error of the cost of alignment $\mathcal{A}$ and the sum of all absolute errors are averaged by the number of the alignments. Hence E_{abs} measures the average absolute error of alignment costs. Note that if $E_{\text{abs}}(x, \mathbf{A})$ is zero, each alignment in $\mathbf{A}$ is an *optimal* alignment under f_x because for the average of nonnegative absolute errors to be zero, every error must be zero. In other words, x is a solution of Inverse Optimal Alignment for examples in $\mathbf{A}$.

Problem 2.4 (Inverse Alignment under Absolute Error) Given a set $\mathbf{A}$ of examples, an alignment cost function f_w with parameter vector w, and a parameter domain D, find a choice x^* of parameter values for w:

$$x^* \quad := \quad \arg\min_{x \in D} E_{\text{abs}}(x, \mathbf{A}), \tag{2.5}$$

where E_{abs} is the error function defined in Equation (2.4). □

This problem finds a choice x^* of parameter values in parameter domain D that minimizes the average absolute error of the example costs for every choice of parameter values.

Relative error criterion

The second variation is called Inverse Alignment under Relative Error or simply the relative error criterion. Given a set $\mathbf{A}$ of alignments and an alignment cost function f_x with a choice x of parameter values, the error function for this problem is the *maximum relative error* of alignment costs $f_x(\mathcal{A})$ for $\mathcal{A} \in \mathbf{A}$ with respect to their corresponding optimal costs $f_x(\mathcal{A}^*)$, where $\mathcal{A}^*$ is the optimal alignment under f_x of the sequences that $\mathcal{A}$ aligns. This function can be written as follows.

Definition 2.4 (Error measure E_{rel}) Given a set $\mathbf{A}$ of alignments and an alignment cost function f_x with a choice x of parameter values, the error measure E_{rel} of set $\mathbf{A}$ under the choice x is:

$$E_{\text{rel}}(x, \mathbf{A}) \quad := \quad \max_{\mathcal{A} \in \mathbf{A}} \left(\frac{f_x(\mathcal{A}) \; - \; f_x(\mathcal{A}^*)}{f_x(\mathcal{A}^*)} \right), \tag{2.6}$$

where $\mathcal{A}^*$ is an optimal alignment under f_x of the sequences that $\mathcal{A}$ aligns. □

The cost fraction inside the maximum in Equation (2.6) is the relative error of the cost of alignment $\mathcal{A}$ and the maximum of all relative errors is taken. Hence E_{rel} measures the maximum relative error of alignment costs. Note that if $E_{\text{rel}}(x, \mathbf{A})$ is zero, each alignment in $\mathbf{A}$ is an *optimal* alignment under f_x. For the maximum of

nonnegative relative errors to be zero, every error must be zero. In other words, x is a solution of Inverse Optimal Alignment for examples in $\mathbf{A}$.

Note that the denominator in Equation (2.6) must not be zero. Since the cost of alignment is nonnegative, the only choice x of parameter values that makes an alignment score zero is $x = 0$. Notice that if $x = 0$, every alignment scores the same under f_x. In other words, x is an optimal solution for Inverse Optimal Alignment and Inverse Near-Optimal Alignment (except the discrepancy error criterion which follows shortly). However this choice x is not biologically meaningful because every alignment scores the same. We discuss this *degeneracy* issue which arises for a linear alignment cost function in detail in Section 4.2.2.

Problem 2.5 (Inverse Alignment under Relative Error) Given a set $\mathbf{A}$ of examples, an alignment cost function f_w with parameter vector w, and a parameter domain D, find a choice x^* of parameter values for w:

$$x^* \quad := \quad \arg\min_{x \in D} E_{\text{rel}}(x, \mathbf{A}), \tag{2.7}$$

where E_{rel} is the error function defined in Equation (2.6). □

This problem finds a choice x^* of parameter values in parameter domain D that minimizes the maximum relative error of the example costs for every choice of parameter values.

Error function E_{rel} measures the *maximum* relative error whereas E_{abs} measures the *average* absolute error. One may attempt to minimize the average relative error across the example costs by modifying the problem. We explain why this problem formulation is not solvable by linear programming in Section 3.3.3.

Discrepancy error criterion

In this variation the error measure function E_{dis} consists of two different types of error, namely absolute error and *recovery error* $d(\mathcal{A}, \mathcal{B})$, which is informally the

fraction of columns of alignment $\mathcal{A}$ that are not present in alignment $\mathcal{B}$ of sequences that $\mathcal{A}$ aligns. The formal definition follows.

Definition 2.5 (Recovery Error) Let $\mathcal{A}$ and $\mathcal{B}$ be alignments that align the same sequences. The *recovery error of* $\mathcal{A}$ *by* $\mathcal{B}$ is

$$d(\mathcal{A}, \mathcal{B}) \;:=\; 1 \;-\; c(\mathcal{A}, \mathcal{B}) \,/\, k, \tag{2.8}$$

where $c(\mathcal{A}, \mathcal{B})$ is the number of columns common in both alignments $\mathcal{A}$ and $\mathcal{B}$ and k is the number of columns in $\mathcal{A}$. □

To understand why the recovery error is introduced, consider the following circumstance. If a solution to Inverse Unique-Optimal Alignment with an example, say $\mathcal{A}$, exists, the example $\mathcal{A}$ can be recovered exactly by finding an optimal alignment $\mathcal{B}$ of the sequences that $\mathcal{A}$ aligns under the solution because the solution makes $\mathcal{A}$ unique-optimal and $\mathcal{B}$ is the optimal under the solution. In other words, $\mathcal{A} = \mathcal{B}$. The recovery error $d(\mathcal{A}, \mathcal{B})$ in this case is zero and the recovery is perfect. As noted earlier, such a solution rarely exists. Suppose we have an alignment cost function in which recovery error is accounted for. If a choice of parameter values that makes the function value of example $\mathcal{A}$ as small as possible is found, the recovery error of an optimal alignment $\mathcal{B}$ under this function is small because the function incorporates the recovery error and it is minimized. Hence this choice of parameter values yields a better recovery of $\mathcal{A}$.

The following error measure, which incorporates the recovery error, was first introduced in [62], where they call the recovery error function d a loss function.

Definition 2.6 (Error measure E_{dis}) Given a set $\mathbf{A}$ of alignments and an alignment cost function f_x with a choice x of parameter values, the error measure E_{dis} of set $\mathbf{A}$ under the choice x is:

$$E_{\text{dis}}(w, \mathbf{A}) \;:=\; \frac{1}{|\mathbf{A}|} \sum_{\mathcal{A} \in \mathbf{A}} \left(f_x(\mathcal{A}) \;-\; f_x(\mathcal{A}^*) \;+\; d(\mathcal{A}, \mathcal{A}^*) \right), \tag{2.9}$$

where $\mathcal{A}^*$ is an alignment that minimizes $f_x(\mathcal{A}^*) - d(\mathcal{A}, \mathcal{A}^*)$ and aligns the sequences that $\mathcal{A}$ aligns. $E_{\text{dis}}(x, \{\mathcal{A}\})$ is called the *discrepancy error* of alignment $\mathcal{A}$ under a choice x of parameter values. □

The sum of all discrepancy errors of alignments in $\mathbf{A}$ is normalized by dividing by the number of the alignments in the set. Hence E_{dis} measures the average discrepancy error of alignments. Note that if $E_{\text{dis}}(x, \mathbf{A})$ is zero, each alignment in $\mathbf{A}$ is an *optimal* alignment under f_x because for the average of nonnegative discrepancy errors to be zero, every discrepancy error must be zero and therefore the absolute cost error must be zero. In other words, x is a solution of Inverse Optimal Alignment for examples in $\mathbf{A}$.

Problem 2.6 (Inverse Alignment under Discrepancy Error) Given a set $\mathbf{A}$ of examples, an alignment cost function f_w with parameter vector w, and a parameter domain D, find a vector x^* of parameter values for w:

$$x^* \quad := \quad \arg\min_{x \in D} \; E_{\text{dis}}(x, \mathbf{A}), \tag{2.10}$$

where E_{dis} is the error measure function defined in Equation (2.9). □

This problem finds a choice x^* of parameter values in parameter domain D that minimizes the average discrepancy error of the examples for every choice of parameter values.

Summary of the chapter We reviewed how to find an optimal alignment of two sequences under a given alignment cost function with linear gap costs (Section 2.1). We explained the problem called inverse alignment and defined and motivated three different formulations of the problem: Inverse Optimal, Unique-Optimal, Near-Optimal Alignments (Section 2.2). For Inverse Near-Optimal Alignment, we defined three variations with error functions: absolute, relative, and discrepancy error criteria. In the next chapter we show how each of the inverse alignment problem can

be reduced to a linear programming problem when the alignment cost function is linear in it parameters.

CHAPTER 3

REDUCTION TO LINEAR PROGRAMMING

We assume that an alignment cost function is a linear function of its parameters throughout this dissertation. We explain how alignment is scored under the standard linear cost model. For each formulation of the inverse alignment problem given in the previous chapter we present a reduction to a linear programming problem. We review the linear programming problem and algorithms that solve linear programs. We conclude that even though the size of the resulting linear program is exponential in the number of variables, it can be solved in polynomial time, combined with a theorem called Equivalence of Separation and Optimization, if Separation, which is a problem referred in the theorem, can be solved in polynomial time.

3.1 Linear cost functions

For most standard forms of alignment cost models, the alignment cost function f is a *linear* function of its parameters. Throughout this dissertation an alignment cost function is assumed to be linear in its alignment parameters, and each feature is scored by an alignment parameter. Formally the linear alignment cost function f is of form:

$$f_w(\mathcal{A}) \;:=\; m_0(\mathcal{A}) \;+\; \sum_{1 \le i \le t} m_i(\mathcal{A})\, w_i, \tag{3.1}$$

where $\mathcal{A}$ is the alignment that f scores, t is the number of free alignment parameters, w is a vector of free alignment parameters w_i, and m_i is a feature measuring function scored by w_i. m_0 is the sum of costs of alignment features whose corresponding parameters are fixed; m_0 is a constant and if no parameter is free, it is zero. In a

linear cost function, the number of alignment features and the number of alignment parameters are the same.

For a concrete example, consider the standard alignment cost function that scores an alignment of two protein sequences with linear gap costs. Alignment parameters for the cost function are:

- *substitution* parameters σ_{ab}, charged for a substitution involving the unordered pair of letters a and b in the protein alphabet of amino acids,
- a *gap-open* parameter γ charged per gap, and
- a *gap-extension* parameter λ charged per residue in a gap.

There are 210 unordered pairs of letters over the alphabet of 20 amino acids. Notice that $\sigma_{ab} = \sigma_{ba}$ for this unordered pair model. Recall that a gap is a maximal run of either inserted letters or deleted letters. A gap of length ℓ costs $\gamma + \lambda\ell$. These 212 alignment parameters σ_{ab}, γ, and λ are $w_1, \ldots, w_{212}$ in the notation of Equation (3.1). The feature measuring functions corresponding to these alignment parameters are:

- $s_{ab}(\mathcal{A})$ which counts the number of substitution pairs a and b in $\mathcal{A}$,
- $g(\mathcal{A})$ which counts the number of gaps in $\mathcal{A}$, and
- $\ell(\mathcal{A})$ which counts the total length of gaps in $\mathcal{A}$.

These feature measuring functions s_{ab}, g, and ℓ are $m_1, \ldots, m_{212}$ in the notation of Equation (3.1).

The cost of alignment $\mathcal{A}$ for this model is

$$f_w(\mathcal{A}) \;:=\; \left(\sum_{a,b} s_{ab}(\mathcal{A})\,\sigma_{ab}\right) \;+\; g(\mathcal{A})\,\gamma \;+\; \ell(\mathcal{A})\,\lambda. \tag{3.2}$$

Remember that an alignment parameter is called *fixed* if it is assigned a value, and *free* otherwise. If some parameters are fixed, the terms in Equation (3.2) involving these fixed parameters are constants in f whose sum is m_0 in the notation of Equation (3.1).

It is worthwhile to compare the values of error functions under two different scenarios. In one scenario all alignment parameters are free and the values for all 212 parameters are learned. In the other scenario substitution parameters are fixed at a standard substitution cost matrix such as a `PAM` [10] or `BLOSUM` [27] matrix and only values for gap parameters are learned. We present our experimental results with these scenarios under different formulations of Inverse Alignment in Chapter 7.

3.2 Linear Programming

When an alignment cost function f is a linear function of its alignment parameters, we can solve all the problem formulations in Chapter 2 using linear programming. Informally the linear programming problem is, given

- a *variable* vector $x = (x_1, \ldots, x_t)$,
- a system of linear *inequalities* in the variables, and
- a linear *objective function* in the variables,

find an assignment x^* of real values to the variables that satisfies all the inequalities and minimizes the objective function. An instance of the linear programming problem is called a *linear program*. The linear programming problem is formally defined as follows.

Problem 3.1 (Linear Programming) Given a matrix $A \in \mathbf{R}^{m \times t}$, a column vector $b \in \mathbf{R}^m$, and an objective coefficient vector $c \in \mathbf{R}^t$, find a value vector x^* for x where

$$x^* \quad := \quad \arg\min_{x \geq 0} \{c \cdot x \ : \ Ax \geq b\} \,. \tag{3.3}$$

□

In matrix notation the system of m inequalities in t variables is described by the $m \times t$ coefficient matrix A and the length-m vector b, and the objective function for the linear program is described by the length-t vector c.

In this problem definition x^* is called an *optimal* solution to the linear program. Any solution x that satisfies $Ax \geq b$ is called a *feasible* solution. A linear program is called *infeasible* if it has no feasible solution, *bounded* if it has an optimal feasible solution, and *unbounded* if it has feasible solutions that are arbitrarily good under the objective function.

In 1947, the first algorithm for linear programming called the *simplex method* was proposed by Dantzig [11]. His algorithm solves the problem in a finite number of steps. The algorithm starts at some vertex of the simplex (or simply polytope) and in each step, it moves from the current vertex to a neighboring vertex at which the objective value is not worse than that of the current vertex. When it reaches a local minimum, the algorithm stops because the simplex is convex and the objective function is linear and hence the local minimum is the global minimum. This algorithm runs in $O(\binom{m}{t})$ steps, and often runs very fast in practice, where m and t are the numbers of inequalities and variables in the linear program respectively. This algorithm, however, was proved to take exponential number $\Omega(2^t)$ of steps for certain input instances [37]. Therefore it is not a polynomial time algorithm. Nevertheless the simplex algorithm and its variants remain a popular choice for Linear Programming.

The *ellipsoid method* given by Khachiyan [31] is the first algorithm whose running time was proved to be polynomial in the numbers of inequalities and variables. The algorithm finds a feasible solution satisfying a system of strict inequalities and uses this solution to solve the linear program. In an iterative approach, each iteration replaces the current ellipsoid that contains a feasible solution to the system of strict linear inequalities if such solution exists with a smaller one containing a solution. After enough iterations either a solution is found or the current ellipsoid is too small to contain a solution, in which case the corresponding linear program is infeasible. Khachiyan's algorithm runs slowly in practice due to the degree of the polynomial too high. The result is still important however because it shows the

existence of a polynomial time algorithm for Linear Programming.

Karmarkar [29] in his revolutionary paper introduced the *interior-point method* which requires in the worst case $O(t^{3.5}B)$ arithmetic operations on $O(B)$-bit numbers, where t is the number of variables in the linear program and B is the number of bits in the input. As opposed to the simplex method which uses vertices at the boundary, the interior point algorithm start with a point that lies inside the set of feasible solutions, and iteratively computes a series of solutions that belong to the strictly feasible set and converges to an optimal solution. With rounding the near optimal solution like the ellipsoid method, the interior point method obtains an exact optimal solution.

To summarize, in the worst case Linear Programming can be solved in polynomial time in the number of variables and inequalities. The methods that are fast in practice may take a polynomial time in the worst case.

3.3 Reduction of Inverse Alignment to Linear Programming

To solve the inverse alignment problem via linear programming, an instance of Inverse Alignment should be reduced to a linear program. We present this reduction for every formulation of Inverse Alignment in Section 2.2.

Each reduction of Inverse Alignment formulations produces a linear program with a huge *system of linear inequalities*. In the reduction, given an example $\mathcal{A}_i$, every alignment $\mathcal{B}$ of the sequences that $\mathcal{A}_i$ aligns generates an inequality in the linear program. The number of such alignments $\mathcal{B}$ is $\Omega(4^n)$ where n is the length of sequences as noted in Section 2.1. An instance of Inverse Alignment with t free alignment parameters and k examples, each of which aligns at least two sequences of length n or greater, generates a linear program with $\Omega(k4^n)$ inequalities in t variables. Surprisingly, for may forms of sequence alignment this linear program can be solved in polynomial time in t variables due to a deep result called *Equivalence*

of Separation and Optimization. We explain this result in Section 3.4 after we give the reduction for each formulation of Inverse Alignment.

The reductions of Inverse Optimal Alignment and Inverse Alignment under Relative Error do not specify the *objective function* in the linear program; any feasible solution of the linear program is a solution to the corresponding problem of Inverse Alignment. We discuss how to exploit the objective function to pick a feasible solution that is biologically more desirable when the object function of the reduced linear program is free and specified by the reduction in Section 3.3.4.

Recall that each formulation of Inverse Alignment contains the following in its input:

- a set of examples $\mathcal{A}_i$, each of which aligns sequences in a set $\mathcal{S}_i$, for $1 \leq i \leq k$,
- an alignment cost function f_w with parameters $w = (w_1, \ldots, w_t)$, and
- a parameter domain D described by linear inequalities of parameters.

3.3.1 Inverse Optimal Alignment

An instance of Inverse Optimal Alignment defined in Problem 2.2 is reduced to a linear program as follows.

The *variables* $x = (x_1, \ldots, x_t)$ in the linear program correspond to free alignment parameters $w = (w_1, \ldots, w_t)$.

The system of *inequalities* in the linear program follows. Every inequality that describes the parameter domain D is added to the system. For each example $\mathcal{A}_i$ and *every* alignment $\mathcal{B}$ of sequences in set $\mathcal{S}_i$, we have an inequality

$$f_x(\mathcal{A}) \quad \leq \quad f_x(\mathcal{B}). \tag{3.4}$$

These inequalities simply express that $\mathcal{A}_i$ is a minimum cost alignment of sequences in $\mathcal{S}_i$, and hence $\mathcal{A}_i$ under parameter values x is an optimal alignment. (Note that if the alignment cost function f is to be maximized, the direction of Inequality (3.4) should be reversed. Negating all these inequalities puts them into the canonical

form $Ax \geq b$ for the linear program.) Written in terms of $x_1, \ldots, x_t$, Inequality (3.4) is by Equation (3.1) equivalent to the linear inequality

$$(m_0(\mathcal{A}_i) - m_0(\mathcal{B})) \leq \sum_{1 \leq j \leq t} (m_j(\mathcal{B}) - m_j(\mathcal{A}_i))\, x_j. \tag{3.5}$$

Note that for any given alignments $\mathcal{A}_i$ and $\mathcal{B}$, the quantities $m_j(\mathcal{B}) - m_j(\mathcal{A}_i)$ in Inequality (3.5) are constants that serve as the coefficients of the variables x and that $m_j(\mathcal{B}) - m_j(\mathcal{A}_i)$ is a constant sum of costs involving fixed parameters. Also note that this inequality is linear in the variables x.

For the *objective function*, one may choose any linear function of the variables x; the objective function is free and every feasible solution $x \geq 0$ that satisfies the system of inequalities $Ax \geq b$ yields a choice of parameter values that makes all examples $\mathcal{A}_i$ optimal.

3.3.2 Inverse Unique-Optimal Alignment

An instance of Inverse Unique-Optimal Alignment defined in Problem 2.3 is reduced to a linear program as follows.

The *variables* $x = (x_1, \ldots, x_t)$ in the linear program correspond to free alignment parameters $w = (w_1, \ldots, w_t)$. We have an error variable $\overline{\alpha}$ in the linear program which corresponds to the unique optimality α .

The system of *inequalities* follows. Every inequality that specifies the parameter domain D is added to the system. For each example $\mathcal{A}_i$ and every other alignment $\mathcal{B} \neq \mathcal{A}_i$ of sequences in set $\mathcal{S}_i$, we have an inequality

$$f_x(\mathcal{A}_i) \leq f_x(\mathcal{B}) - \overline{\alpha}. \tag{3.6}$$

This inequality expresses that $\mathcal{A}_i$ is an $\overline{\alpha}$ unique-optimal alignment under parameter values x. Written in terms of $x_1, \ldots, x_t$ and $\overline{\alpha}$, Inequality (3.6) is by Equation (3.1) equivalent to the linear inequality

$$(m_0(\mathcal{A}_i) - m_0(\mathcal{B})) \leq \sum_{1 \leq j \leq t} (m_j(\mathcal{B}) - m_j(\mathcal{A}_i))\, x_j - \overline{\alpha}. \tag{3.7}$$

Note that this inequality is linear in the variables x and $\overline{\alpha}$.

For the *objective function*, we choose $\min\{-\overline{\alpha}\}$ because the value of the unique optimality α must be maximized as specified in the problem formulation.

When the optimal solution for the linear program is found, $\overline{\alpha} > 0$ must be checked. If $\overline{\alpha} > 0$, the optimal solution x^* for variables x makes examples $\mathcal{A}_i$ all be $\overline{\alpha}$ unique-optimal under the parameter choice x^* and $\overline{\alpha}$ cannot be made any larger. Otherwise there is no solution for Inverse Unique-Optimal Alignment because after the value of $\overline{\alpha}$ is maximized, it is still nonpositive. If no solution is found for the linear program, then there is no solution for the Inverse Unique-Optimal Alignment.

3.3.3 Inverse Near-Optimal Alignment

The reduction of each variation of Inverse Near-Optimal Alignment resembles that of Inverse Optimal Alignment. The reduction of the absolute error criterion has an additional *variable* ϵ_i in the linear program for each example $\mathcal{A}_i$ that measures the absolute error of the cost of example $\mathcal{A}_i$ and the average of all ϵ_i is minimized by objective function, which gives E_{abs}. The reduction of the discrepancy error criterion is similar. The reduction of the relative error criterion generates a series of linear programs, each of which is for a *fixed* ϵ upper bound on E_{rel} that is to be minimized. We find the smallest value ϵ^* of ϵ (and hence E_{rel}) for which there is a feasible solution using binary search on ϵ and by solving the series of linear programs simultaneously.

Absolute error criterion

An instance of Inverse Alignment under Absolute Error defined in Problem 2.4 is reduced to a linear program as follows.

The *variables* $x = (x_1, \ldots, x_t)$ in the linear program correspond to free alignment parameters $w = (w_1, \ldots, w_t)$. For each example $\mathcal{A}_i$ we have an error variable ϵ_i that holds a value of an upper bound on all absolute errors $f_x(\mathcal{A}_i) - f_x(\mathcal{B})$ for every

alignment $\mathcal{B}$ of the sequences in set $\mathcal{S}_i$. The number of variables in the linear program is $t + k$.

The system of *inequalities* follows. Every inequality that specifies the parameter domain D is added to the system. For each $\mathcal{A}_i$ and every alignment $\mathcal{B}$ of sequences in set $\mathcal{S}_i$, we have an inequality

$$f_x(\mathcal{A}_i) \; - \; f_x(\mathcal{B}) \;\; \leq \;\; \epsilon_i. \tag{3.8}$$

Note that this inequality is linear in variables x and ϵ_i.

The *objective function* in the linear program is to minimize

$$\frac{1}{k} \sum_{1 \leq i \leq k} \epsilon_i. \tag{3.9}$$

Theorem 3.1 (Reduction for Absolute Error Criterion) *An optimal solution of the above linear program is a solution of Inverse Alignment under Absolute Error.*

Proof If Inequality (3.8) holds for every alignment $\mathcal{B}$ of the sequences in set $\mathcal{S}_i$ under f_x, then in particular it holds for $\mathcal{B} = \mathcal{A}_i^*$ where $\mathcal{A}_i^*$ is an optimal alignment of the sequences in $\mathcal{S}_i$ under f_x. On the other hand, if Inequality (3.8) holds for $\mathcal{B} = \mathcal{A}_i^*$, it holds for every alignment $\mathcal{B}$ of the sequences in $\mathcal{S}_i$ because $f_x(\mathcal{A}_i^*) \leq f_x(\mathcal{B})$. Hence ϵ_i is bounded from below by $f_x(\mathcal{A}) - f_x(\mathcal{A}_i^*)$. Note that if the objective function (3.9) is minimized, ϵ_i is set to $f_x(\mathcal{A}_i) - f_x(\mathcal{A}_i^*)$. Thus the linear program has the objective value $\frac{1}{k} \sum_{1 \leq i \leq k} (f_x(\mathcal{A}_i) - f_x(\mathcal{A}_i^*))$, which is $E_{\text{abs}}(x, \{\mathcal{A}_i\})$ by Definition 2.3. An optimal solution x^* over all $x \in D$ of this linear program satisfies $\arg\min_{x \in D} E_{\text{abs}}(x, \{\mathcal{A}_i\})$, which is a solution to Inverse Alignment under Absolute Error. □

Relative error criterion

To find a solution of Inverse Alignment under Relative Error, we reduce the problem to a series of linear programs $L_1, L_2, \ldots$, which are explained shortly, and solve the

linear programs L_j iteratively. We first consider the problem assuming a *fixed* upper bound ϵ on the error measure E_{rel}, the maximum relative error of all example costs. At the jth iteration, for a given bound ϵ and linear program L_j, we test whether there is a feasible solution with a maximum relative error at most ϵ by solving L_j. We then iteratively find the smallest value ϵ^* of ϵ for which there is a feasible solution x using binary search on ϵ. After a linear program generated by the reduction of Inverse Alignment under Relative Error (defined in Problem 2.5) is described, we discuss this iterative algorithm in detail.

The *variables* $x = (x_1, \ldots, x_t)$ in the linear program correspond to free alignment parameters $w = (w_1, \ldots, w_t)$.

The system of *inequalities* follows. Every inequality that specifies the parameter domain D is added to the system. For each $\mathcal{A}_i$ and every alignment $\mathcal{B}$ of sequences in set $\mathcal{S}_i$, we have an inequality

$$f_x(\mathcal{A}_i) \quad \leq \quad (1+\epsilon)\, f_x(\mathcal{B}). \tag{3.10}$$

Note that ϵ is a constant and this inequality is linear in the variables x.

For the *objective function*, one may choose any linear function of the variables x; the objective function is free.

Theorem 3.2 (Bounding relative error) *Suppose the cost of alignment is always positive. If x is a feasible solution of the above linear program, then $E_{\text{rel}}(x, \{\mathcal{A}_i\}) \leq \epsilon$.*

Proof If Inequality (3.10) holds for every alignment $\mathcal{B}$ of the sequences in set $\mathcal{S}_i$ under f_x, then in particular it holds for $\mathcal{B} = \mathcal{A}_i^*$ where $\mathcal{A}_i^*$ is an optimal alignment of the sequences in $\mathcal{S}_i$ under f_x. On the other hand, if Inequality (3.10) holds for $\mathcal{B} = \mathcal{A}_i^*$, it holds for every alignment $\mathcal{B}$ of the sequences in $\mathcal{S}_i$ because $f_x(\mathcal{A}_i^*) \leq f_x(\mathcal{B})$. Hence ϵ is bounded from below by $(f_x(\mathcal{A}_i) - f_x(\mathcal{A}_i^*))/f_x(\mathcal{A}_i^*)$ because $f_x(\mathcal{A}_i^*) > 0$. Therefore ϵ is bounded from below by the maximum of $(f_x(\mathcal{A}_i) - f_x(\mathcal{A}_i^*))/f_x(\mathcal{A}_i^*)$ for $i = 1, \ldots, k$, which is $E_{\text{rel}}(x, \{\mathcal{A}_i\})$ by Definition 2.4. □

Notice the assumption of the theorem. Since the cost of alignment is always nonnegative, the only choice x of parameter values that makes an alignment score zero is $x = 0$. This degenerate solution must be eliminated. (As mentioned after the problem formulation in the previous chapter, this solution is not biologically meaningful because all alignments score the same under f_x.) This degeneracy issue is further discussed in Section 4.2.2, but for now it is enough to know such a choice x can be excluded from the parameter domain D by an inequality called a *nondegeneracy inequality*.

Finding the smallest maximum relative error We examined what linear program is generated for a given value of upper bound ϵ on E_{rel}. We now explain how one can find the smallest value ϵ^* of ϵ for which there is a solution to any desired accuracy $\xi > 0$. First set $\epsilon = 0$ and solve the linear program with $\epsilon = 0$. If a solution of this linear program exists, $\epsilon^* = 0$. Note that the solution is a solution of Inverse Optimal Alignment. Otherwise by repeatedly doubling the value of ϵ (in other words, setting $\epsilon = 2^i \xi$ for $i = 0, 1, \ldots$) and solving the linear program for the ϵ, find an upper bound $\overline{\epsilon}$ on ϵ^* until a solution of the linear program for $\epsilon = \overline{\epsilon}$ is found. Given an upper bound $\overline{\epsilon}$ and a lower bound $\underline{\epsilon} = \overline{\epsilon}/2$ on ϵ^* for which no solution of the linear program with $\epsilon = \underline{\epsilon}$ exists, perform binary search on the real values in interval $[\underline{\epsilon}, \overline{\epsilon}]$ that are spaced distance ξ apart. This finds the smallest value ϵ^* to within accuracy ξ by solving $\mathrm{O}(\lg(\epsilon^*/\xi))$ linear programs.

As an aside, one could attempt to minimize the *average* relative error across the examples by modifying this reduction as follows. Each example $\mathcal{A}_i$ would have a corresponding error variable ϵ_i, and Inequality (3.10) would be modified by replacing the global constant ϵ with the variable ϵ_i. The objective function would be to minimize $\frac{1}{k} \sum_{1 \le i \le k} \epsilon_i$. Unfortunately, the modified Inequality (3.10) then becomes quadratic in variable ϵ_i and the vector of variables x, which is no longer solvable by linear programming.

Discrepancy error criterion

An instance of Inverse Alignment under Discrepancy Error defined in Problem 2.6 is reduced to a linear program as follows.

The *variables* $x = (x_1, \ldots, x_t)$ in the linear program correspond to free alignment parameters $w = (w_1, \ldots, w_t)$. For each example $\mathcal{A}_i$, we have an error variable ϵ_i that holds a value of an upper bound on all discrepancy errors $f_x(\mathcal{A}_i) - f_x(\mathcal{B}) + d(\mathcal{A}_i, \mathcal{B})$ for every alignment $\mathcal{B}$ of the sequences in set $\mathcal{S}_i$. The number of variables in the linear program is $t + k$.

The system of *inequalities* follows. Every inequality that specified the parameter domain D is added to the system. For each $\mathcal{A}_i$ and every alignment $\mathcal{B}$ of the sequences in set $\mathcal{S}_i$, we have an inequality

$$f_x(\mathcal{A}_i) \; - \; f_x(\mathcal{B}) \; + \; d(\mathcal{A}_i, \mathcal{B}) \;\; \leq \;\; \epsilon_i. \tag{3.11}$$

Note that this inequality is linear in the variables x and ϵ_i.

The *objective function* in the linear program is to minimize

$$\frac{1}{k} \sum_{1 \leq i \leq k} \epsilon_i. \tag{3.12}$$

Theorem 3.3 (Reduction for Discrepancy Error Criterion) *An optimal solution of the above linear program is a solution of Inverse Alignment under Discrepancy Error.*

Proof If Inequality (3.11) holds for every alignment $\mathcal{B}$ of the sequences in set $\mathcal{S}_i$ under the parameter value choice x, then in particular it holds for $\mathcal{B} = \mathcal{A}_i^*$ where $\mathcal{A}_i^*$ is an alignment of the sequences in $\mathcal{S}_i$ that minimizes $f_x(\mathcal{B}) - d(\mathcal{A}_i, \mathcal{B})$. On the other hand, if Inequality (3.11) holds for $\mathcal{B} = \mathcal{A}_i^*$, it holds for every alignment $\mathcal{B}$ of the sequences in $\mathcal{S}_i$ because $f_x(\mathcal{A}_i^*) - d(\mathcal{A}_i, \mathcal{A}_i^*) \leq f_x(\mathcal{B}) - d(\mathcal{A}_i, \mathcal{B})$. Hence ϵ_i is bounded from below by $f_x(\mathcal{A}) - f_x(\mathcal{A}_i^*) + d(\mathcal{A}_i, \mathcal{A}_i^*)$. Note that if the objective function (3.12) is minimized, ϵ_i is set to $f_x(\mathcal{A}) - f_x(\mathcal{A}_i^*) + d(\mathcal{A}_i, \mathcal{A}_i^*)$. Thus the

objective value of the linear program is $\frac{1}{k}\sum_{1\leq i\leq k}(f_x(\mathcal{A}_i) - f_x(\mathcal{A}_i^*) + d(\mathcal{A}_i, \mathcal{A}_i^*))$ which is $E_{\text{dis}}(x, \{\mathcal{A}_i\})$ by Definition 2.6. An optimal solution x^* over all $x \in D$ of this linear program satisfies $\arg\min_{x\in D} E_{\text{dis}}(x, \{\mathcal{A}_i\})$, which is a solution to Inverse Alignment under Discrepancy Error. □

3.3.4 Exploiting the objective function

As mentioned in Section 3.3, the objective function in a linear program can be used to pick a biologically meaningful choice of parameter values *even when* the reduction specifies the objective function. The idea is to solve a multi-stage linear program. After the minimum value of the objective function given by the reduction, if any, is found, we add the optimality of the objective function value as a constraint to the current system of inequalities and solve this system with a new objective function designed to pick a biologically more desirable solution. We explain this idea and present concrete examples of objective functions that aim to pick a biologically desirable solution.

Formulations with a fixed objective function

When the reduction of Inverse Alignment specifies an objective function, an optimal solution that minimizes the objective function can be a less desirable solution. For such reduction (and for those that do not specify any objective function), we can use the following idea to find a better solution.

Let v^* be the minimum value of the objective function $c \cdot x$ and L be the system of linear inequalities specified by the reduction. To pick a biologically meaningful solution, we add an inequality $c \cdot x \leq v^*$ to the system of inequalities L and set $\widetilde{c} \cdot x$ as a new objective function to be minimized, where $\widetilde{c}$ is the coefficient vector designed to pick a biologically meaningful solution. Note that any feasible solution of the new linear program is an optimal solution of the original linear program. We can use this approach for multiple stages.

Formulations with a free objective function

When the objective function is free, we can exploit the freedom to pick a feasible solution that is biologically more desirable. We give concrete examples of objective functions designed to do so under the standard alignment cost model described in Section 3.1. These objective functions are for the reductions of Inverse Optimal Alignment and Inverse Alignment under Relative Error.

When all substitution parameters σ_{ab} are fixed, the free alignment parameters are the gap-open parameter γ and gap-extension parameter 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 $\max\{\gamma - \lambda\}$. Another option is to solve a two stage linear program: first we solve the objective $\max\{\gamma\}$ to find γ^*, and then in the second stage we solve the objective $\min\{\lambda\}$ with parameter γ fixed at value γ^*.

When no parameter is fixed, the free alignment parameters are all of σ_{ab}, γ, and λ. In this case we might want to maximize the separation between substitution costs σ_{ab} for $a \neq b$ and identity costs σ_{aa}. Then one possibility for the objective is to maximize the difference between the minimum substitution cost and the maximum identity cost so that they are as far apart as possible. We can express this in our linear programming formulation by adding two new variables: σ, which will equal the minimum substitution cost, and ι, which will equal the maximum identity cost. Using the objective $\max\{\sigma - \iota\}$, and adding the inequalities $\sigma \leq \sigma_{ab}$ for all $a \neq b$, and $\iota \geq \sigma_{aa}$ for all a, will achieve this goal. (Another possibility is to maximize the difference between the average substitution cost and the average identity cost, which is also an objective that is linear in the parameters.) This objective on substitution costs can be combined with our objective on gap penalties by $\max\{\sigma - \iota + \gamma - \lambda\}$. Another option is to solve a three stage linear program:

we solve the objective $\max\{\sigma - \iota\}$ to find σ^*_{ab}, and then solve the above two stage objective for gap costs with parameters σ_{ab} fixed at values σ^*_{ab}.

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 examples $\mathcal{A}_i$. This is borne out by our experiments in Chapter 7.

3.4 Equivalence of Separation and Optimization

One of the truly far-reaching results in linear programming is a theorem called *Equivalence of Separation and Optimization* This result was discovered in the early 1980's by and in full generality by Grötschel, Lovász and Schrijver [22], Padberg and Rao [47], and Karp and Papadimitriou [30]. The theorem mentions two mathematical problems in it, and to explain these problems requires a few concepts. Linear Programming optimizes a linear objective function of real variables over a domain defined by a system of 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 called *rational*. A polyhedron that contains no infinite rays is said to be *bounded*. Hence a bounded polyhedron has finite volume. The problems stated in the theorem are the following.

Problem 3.2 (Optimization) Given a bounded rational polyhedron $P \subseteq \mathbf{R}^n$ and a rational vector $c \in \mathbf{R}^n$ that specifies the objective function, either

- find a point $x^* \in P$ that minimizes $c \cdot x$ over all $x \in P$, or
- conclude that P is empty. □

Note that an instance of this problem is equivalent to a linear program under three assumptions: (1) the linear program is *bounded*, (2) the coefficient vector of the objective function is *rational*, and (3) all the coefficients in the system of inequalities are *rational*. Suppose the assumptions hold. Then the instance of Optimization is a linear program with a vector x of n variables, a system of inequalities $Ax \geq b$ which defines the bounded rational polyhedron P, and an objective function $c \cdot x$. If an optimal point x^* to the instance of Optimization exists, x^* is an optimal solution to the linear program. If P in the instance of Optimization is empty, the linear program is infeasible.

Problem 3.3 (Separation) Given a bounded rational polyhedron $P \subseteq \mathbf{R}^n$ and a rational point $v \in \mathbf{R}^n$, either

- find a rational vector $w \in \mathbf{R}^n$ and a rational number b that specify an inequality such that $w \cdot x \leq b$ for all $x \in P$, but $w \cdot v > b$, or
- conclude that $v \in P$. □

In other words, an algorithm for Separation, which we call a *separation algorithm*, for polyhedron P determines whether point v lies inside or boundary of P. If v lies outside, then the algorithm finds an inequality that is satisfied by all points in P but is violated by v. Such a *violated inequality* gives a hyperplane that separates v from P.

Equivalence of Separation and Optimization 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 [23]. The following statement of the theorem is in terms of bounded polyhedra, though it can also be extended to unbounded polyhedra as long as they are bounded in the direction of the objective function [23].

Theorem 3.4 (Equivalence of Separation and Optimization [22, 47, 30])

Optimization on a bounded rational polyhedron can be solved in polynomial time if and only if Separation 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 t 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 system L of $2^{\Omega(t)}$ inequalities can be solved in $t^{O(1)}$ time if, for any candidate solution $\widetilde{x}$, one can in $t^{O(1)}$ time determine that $\widetilde{x}$ satisfies all the inequalities in L, or if it does not, report an inequality in L that $\widetilde{x}$ violates. Of course a separation algorithm that simply scans the system L and tests each inequality will not achieve this time bound.

The proof of Theorem 3.4 exploits properties of the ellipsoid algorithm for linear programming described in Section 3.2. 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.

Summary of the chapter We gave the reduction of each inverse alignment formulation to Linear Programming (Section 3.3), for which polynomial time algorithms exist (Section 3.2). Even if the resulting linear program has a system of inequalities with size exponential in the number of variables, as long as there exists a separation algorithm for the linear program which solves Separation in polynomial time, the linear program can be solved in polynomial time (Section 3.4). In the following chapter, we give a polynomial time separation algorithm for each of the linear programs resulting from the reductions for all inverse alignment formulations. These separation algorithms, combined with Theorem 3.4, will show that these formulations of inverse alignment can be solve in polynomial time.

CHAPTER 4

ALGORITHM AND IMPLEMENTATION

The separation algorithms for each linear programming formulation of Inverse Alignment, which all run in polynomial time, are presented. These separation algorithms, combined with Theorem 3.4, yield theoretical polynomial time algorithms for all Inverse Alignment formulations under certain assumptions. We give a practical algorithm which also uses the separation algorithms and runs fast although it is not guaranteed to run in polynomial time. These algorithms are the main results of this dissertation. In the later part of this chapter, various implementation issues are discussed: eliminating degeneracy solutions, managing examples for fast separation, seeding inequalities to find a solution fast, managing inequalities to keep linear programs small and, finding bounding inequalities for fixed alignment parameters.

4.1 Practical and theoretical algorithms

Due to the Equivalence of Separation and Equivalence theorem, each formulation of Inverse Alignment can be solved in polynomial time, if the separation algorithms for each linear programming formulation run in polynomial time. After such separation algorithms are given, we articulate this result as a theorem. This theorem, however, does not directly yield algorithms for quickly solving exponentially large linear programs in practice. We give an algorithm called a *cutting plane algorithm* which runs fast in practice although it is not guaranteed to run in polynomial time.

4.1.1 Practical cutting plane algorithm

In practice a polynomial time solution to Separation (Problem 3.3) is typically leveraged in an algorithm called a *cutting plane algorithm* [6, 9] which solves a linear program with a system L of inequalities. Even though cutting plane algorithms are not guaranteed to run in polynomial time in the numbers of inequalities and variables in L, they run fast in practice.

The following cutting plane algorithm assumes that the linear program with L is *bounded*. This assumption certainly holds when all alignment parameters are free as explained in 4.2.1. If at least one of alignment parameters is fixed, the linear program could be unbounded. In this case it is important to find certain types of inequalities, which are called *bounding inequalities*, in the system to know the feasibility of the linear program. When some parameters are fixed, the linear program is unbounded if and only if bounding inequalities are missing in the system of inequalities for the linear program. Bounding inequalities are discussed further in Section 4.2.1.

Algorithm 4.1 (Cutting Plane Algorithm)

(1) Start with a small subset S of the inequalities in L.

(2) Compute an optimal solution x to the linear program given by subset S. If no such solution exists, report that the linear program with L is infeasible and halt.

(3) Call the separation algorithm for L on x. If the algorithm reports that x satisfies L, output x as an optimal solution to the linear program with L and halt.

(4) Otherwise, add the violated inequality returned by the separation algorithm in Step (3) to set S, and loop back to Step (2). □

Lemma 4.1 (Correctness of Step (2) in Algorithm 4.1) *Assume that the lin-*

ear program with S is bounded. If an optimal solution x to the linear program with S does not exist, the linear program with L is infeasible.

Proof If an optimal solution x to the linear program with S does not exist, the linear program with S is infeasible because it is bounded by assumption. Note that adding more inequalities to S does not make the linear program with S feasible. Since L contains all the inequalities in S, the linear program with L is infeasible. □

Lemma 4.2 (Correctness of Step (3) in Algorithm 4.1) *If the separation algorithm reports that a candidate solution x satisfies the linear program with L, x is an optimal solution to the linear program with L.*

Proof If x is not an optimal solution to the linear program with L, there is an inequality in L separating x from the polyhedron defined by S. This inequality is a solution to Separation but the separation problem did not return it in Step (3). This is a contradiction. Therefore x is not an optimal solution to the linear program with L. □

Theorem 4.1 (Correctness of Algorithm 4.1) *If the linear program with S in Algorithm 4.1 is bounded, the algorithm solves the linear program with L.*

Proof Note the linear program with L is bounded because L contains all the inequalities in S and the linear program with S is bounded. The possible outcome of a bounded linear program is either the linear program is infeasible, or an optimal solution to the linear program is found. The first case is correctly reported at Step (2) by Lemma 4.1. The second case is correctly reported at Step (3) by Lemma 4.2. Therefore the algorithm solves the linear program with L. □

4.1.2 Separation algorithm

This section presents a separation algorithm for each linear programming formulation of Inverse Alignment. Recall that given a parameter value assignment x for alignment parameters, a separation algorithm either concludes that x satisfies the system L of all the inequalities in the linear program or identifies a violated inequality in L.

Conceptually we have a different separation algorithm for the subset L_i of L which is associated with each example $\mathcal{A}_i$ for $1 \leq i \leq k$. To separate the system L, the separation algorithms for $\mathcal{A}_1, \ldots, \mathcal{A}_k$ run consecutively. As soon as one algorithm finds a violated inequality, the algorithm returns the violated inequality and the separation of the system L stops. If no algorithm finds a violated inequality, it is reported that x satisfies the linear program with L.

The separation algorithm on subset L_i uses an algorithm for Optimal Alignment to compute an optimal alignment of given sequences. In addition, for the linear programming formulation of Inverse Unique-Optimal Alignment, a next-best alignment is computed by the separation algorithm. Note that the separation for L takes $O(kT)$ time, where k is the number of examples and T is the time to compute an optimal alignment (and a next-best alignment). Hence if T is polynomial, the separation algorithm runs in polynomial time.

The input to the following separation algorithms consists of

- a linear program with a system L_i of inequalities that are associated with example $\mathcal{A}_i$,
- a value vector x for the variables in the linear program,
- example $\mathcal{A}_i$ which aligns the sequences in set $\mathcal{S}_i$, and
- an alignment cost function f_w with alignment parameters w.

Inverse Optimal Alignment

The system L_i of inequalities associated with example $\mathcal{A}_i$ resulting from the reduction for Inverse Optimal Alignment in Section 3.3.1 consists of the following inequalities:

$$f_x(\mathcal{A}_i) \quad \leq \quad f_x(\mathcal{B}), \tag{4.1}$$

where $\mathcal{B}$ is any alignment of the sequences in the set $\mathcal{S}_i$ and x is a value assignment of alignment parameters.

Algorithm 4.2 (Separation for Inverse Optimal Alignment)

(1) Compute an optimal alignment $\mathcal{B}^*$ of the sequences in $\mathcal{S}_i$ under f_x with parameter choice x.

(2) If Inequality (4.1) does not hold for $\mathcal{B} = \mathcal{B}^*$, return it as a violated inequality and halt.

(3) Report that x is an optimal solution to the linear program. □

Theorem 4.2 (Correctness of Algorithm 4.2) *Algorithm 4.2 separates a value vector x for the system L_i of inequalities resulting from the reduction of Inverse Optimal Alignment.*

Proof Let $\mathcal{B}^*$ be the optimal alignment computed in Step (1). We show that Inequality (4.1) holds for $\mathcal{B} = \mathcal{B}^*$ if and only if the inequality holds for every alignment $\mathcal{B}$ of the sequences in $\mathcal{S}_i$. ($\Rightarrow$) If Inequality (4.1) holds for $\mathcal{B} = \mathcal{B}^*$, then the inequality holds for every alignment $\mathcal{B}$ of the sequences in $\mathcal{S}_i$ because $f_x(\mathcal{B}^*)$ is the smallest of all $f_x(\mathcal{B})$. ($\Leftarrow$) If Inequality (4.1) holds for all alignments $\mathcal{B}$ of the sequences in $\mathcal{S}_i$, then the inequality holds $\mathcal{B} = \mathcal{B}^*$ because $\mathcal{B}^*$ is one of the alignments $\mathcal{B}$. Therefore the algorithm separates x for the linear program with the system L_i. □

Inverse Unique-Optimal Alignment

The system L_i of inequalities associated with example $\mathcal{A}_i$ resulting from the reduction for Inverse Unique-Optimal Alignment consists of the following inequalities:

$$f_x(\mathcal{A}_i) \;\leq\; f_x(\mathcal{B}) \;-\; \overline{\alpha}, \tag{4.2}$$

where $\mathcal{B}$ is any alignment other than $\mathcal{A}_i$ of the sequences in the set $\mathcal{S}_i$, x is a value assignment of alignment parameters, and $\overline{\alpha} > 0$ is a value for the unique optimality in Inverse Unique-Optimal Alignment.

Algorithm 4.3 (Separation for Inverse Unique-Optimal Alignment)

(1) Compute an optimal alignment $\mathcal{B}_1^*$ of sequences in $\mathcal{S}_i$ under f_x with parameter choice x.

(2) If $\mathcal{B}_1^* \neq \mathcal{A}_i$, return Inequality (4.2) for $\mathcal{B} = \mathcal{B}_1^*$ as a violated inequality and halt.

(3) Compute a next-best alignment $\mathcal{B}_2^*$ of sequences in $\mathcal{S}_i$ under f_x.

(4) If Inequality (4.2) does not hold for $\mathcal{B} = \mathcal{B}_2^*$, return it as a violated inequality and halt.

(5) Report that x is an optimal solution to the linear program. □

Theorem 4.3 (Correctness of Algorithm 4.3) *Algorithm 4.3 separates a value vector $(x, \overline{\alpha} > 0)$ for the system L_i of inequalities resulting from the reduction of Inverse Unique-Optimal Alignment.*

Proof Let $\mathcal{B}_1^*$ and $\mathcal{B}_2^*$ be the optimal and the next-best alignments computed in Steps (1) and (3) respectively. We show that Inequality (4.2) holds for $\mathcal{B} = \mathcal{B}_2^* \neq \mathcal{A}_i$ if and only if the inequality holds for every alignment $\mathcal{B}$ other than $\mathcal{A}_i$ of the sequences in $\mathcal{S}_i$. ($\Rightarrow$) Assume that Inequality (4.2) holds for $\mathcal{B} = \mathcal{B}_2^*$. This implies $\mathcal{A}_i = \mathcal{B}_1^*$ because $\mathcal{B}_2^*$ is computed only if the condition is true. Hence $\mathcal{A}_i = \mathcal{B}_1^* \neq \mathcal{B}_2^*$ because an optimal alignment is not a next-best alignment. $f_x(\mathcal{B}_2^*) \leq f_x(\mathcal{B})$ holds

for every alignment $\mathcal{B}$ other than $\mathcal{A}_i$ of the sequences in $\mathcal{S}_i$ because $f_x(\mathcal{B}_2^*)$ is the smallest of all $f(\mathcal{B})$. Hence Inequality (4.2) holds for every alignment $\mathcal{B}$ other than $\mathcal{A}_i$ of the sequences in $\mathcal{S}_i$. ($\Leftarrow$) Assume that Inequality (4.2) holds for all alignments $\mathcal{B}$ other than $\mathcal{A}_i$ of the sequences in $\mathcal{S}_i$. This implies $\mathcal{B} \neq \mathcal{B}_1^*$ because $f(\mathcal{B}_1^*)$ is the smallest and the inequality cannot hold for $\mathcal{B} = \mathcal{B}_1^*$ and $\overline{\alpha} > 0$. Then Inequality (4.2) holds for $\mathcal{B} = \mathcal{B}_2^*$ because $\mathcal{B}_2^*$ is one of the alignments $\mathcal{B}$. Therefore the algorithm separates $(x, \overline{\alpha} > 0)$ for the linear program with the system L_i. $\square$

We give a description of an algorithm that finds a next-best alignment of two sequences under a given alignment cost function with affine gap costs. This algorithm is a simple modification of Algorithm 2.1.

Algorithm 4.4 (Algorithm for Next-Best Alignment) Let $\mathcal{A}_1^*$ denote an optimal alignment of two sequences of length m and n found by Algorithm 2.1 filling out the dynamic programming table $\mathcal{F}$. Consider a path that $\mathcal{A}_1^*$ walking through in $\mathcal{F}$. For a fixed row i in the table, the optimal alignment $\mathcal{A}_1^*$ visits entry (i, k) for some $0 \leq k \leq n$. A next-best alignment $\mathcal{A}_2^*$ visits either a *different entry* or the *same entry* on the same row in the table. To find an optimal alignment that visits a *specified entry or cell* we use the following approach.

We call table $\mathcal{F}$ a *forward* table because each cell in entry (i, j) contains the cost of an optimal alignment of *prefixes* $S[1:i]$ and $T[1:j]$ that ends with a certain type of operations: substitution, deletion, or insertion. Consider another algorithm that finds an optimal alignment $\mathcal{C}^*$ by filling out a dynamic programming table $\mathcal{B}$ in which each cell in (i, j) contains the cost of an optimal alignment of *suffixes* $S[i:m]$ and $T[j:n]$ that starts with a certain type. We call table $\mathcal{B}$ a *backward* table. If alignment $\mathcal{C}^*$ is not the same as alignment $\mathcal{A}_1^*$, we report $\mathcal{C}^*$ as a next-best alignment and stop. Otherwise the path of $\mathcal{A}_1^*$ in $\mathcal{F}$ and that of $\mathcal{C}^*$ in $\mathcal{B}$ are the same. To avoid confusion, we use the table symbol as a subscript for each cell and entry. For example $D_{\mathcal{F}}(i, j)$ denotes $D(i, j)$ in table $\mathcal{F}$. The cost of an optimal alignment

visiting (i, j) whose prefix alignment ending at (i, j) is optimal is given by $g(i, j)$:

$$g(i,j) \;:=\; \min\Big\{g_D(i,j)\ ,\ g_V(i,j)\ ,\ g_H(i,j)\Big\},$$

where

$$\begin{aligned}
g_D(i,j) &:= D_{\mathcal{F}}(i,j) \;+\; x(i,j),\\
g_V(i,j) &:= V_{\mathcal{F}}(i,j) \;+\; \min\Big\{V_{\mathcal{B}}(i+1,j)-\gamma\ ,\ x(i,j)\Big\},\\
g_H(i,j) &:= H_{\mathcal{F}}(i,j) \;+\; \min\Big\{H_{\mathcal{B}}(i,j+1)-\gamma\ ,\ x(i,j)\Big\}, \quad \text{and}\\
x(i,j) &:= \min\Big\{M_{\mathcal{B}}(i+1,j)\ ,\ M_{\mathcal{B}}(i,j+1)\ ,\ M_{\mathcal{B}}(i+1,j+1)\Big\}.
\end{aligned}$$

(For simple presentation we omitted boundary cases.) The idea of this computation g is for a given entry (i, j) that an *optimal prefix* alignment visits, we consider all the possible extensions for the suffixes of sequences, which are given by g_D, g_V, and g_H. (Note that when either sequence has a gap which spans over (i, j) and its extension entry, the gap-open cost γ should not be charged twice as shown in function g_V and g_H.) Note that one can compute an optimal alignment that visits a specified entry (or cell) using g (or either g_D, g_V, or g_H). The optimal alignment can be recovered from $\mathcal{F}$ and $\mathcal{B}$ by stitching two paths together, each of which starts from the cell that contributed to the minimum cost given by $g(i, j)$.

Suppose $\mathcal{A}_1^*$ visits entry (i, k) for $i = \lfloor m/2 \rfloor$. As we mentioned before, $\mathcal{A}_2^*$ visits either a different entry or the same entry on row i. For the case that $\mathcal{A}_2^*$ visits a different entry on row i, we compute the minimum cost a of all alignments visiting (i, j) for every column $j \neq k$, which is $g(i, j)$. For the case that $\mathcal{A}_2^*$ visits the same entry (i, k), we need to consider *every* alignment that visits (i, j). Assume that $\mathcal{A}_2^*$ diverges from $\mathcal{A}_1^*$'s path on or below the ith row but never above. (Note that this assumption is valid. If $\mathcal{A}_2^*$ diverges from the path somewhere above row i too, then the gain of $\mathcal{A}_2^*$'s detour is none because $\mathcal{A}_2^*$ visits (i, j) by the assumption and $\mathcal{A}_1^*$'s path is optimal.) For each entry (v, w) visited by $\mathcal{A}_1^*$ on or below row i, we compute the minimum cost $b(v, w)$ of all alignments visiting (v, w) but *not* the

cell in (v, w) that $\mathcal{A}_1^*$ visits. This cost is given by

$$b(v,w) \;:=\; \begin{cases} \min\Big\{g_V(v,w)\ ,\ g_H(v,w)\Big\} & \text{if } \mathcal{A}_1^* \text{ visits } D_{\mathcal{F}}(v,w), \\ \min\Big\{g_D(v,w)\ ,\ g_H(v,w)\Big\} & \text{if } \mathcal{A}_1^* \text{ visits } V_{\mathcal{F}}(v,w), \\ \min\Big\{g_D(v,w)\ ,\ g_V(v,w)\Big\} & \text{if } \mathcal{A}_1^* \text{ visits } H_{\mathcal{F}}(v,w). \end{cases}$$

Assume that $\mathcal{A}_2^*$ diverges from $\mathcal{A}_1^*$'s path above row i but not on or below. For entry (v, w) visited by $\mathcal{A}_1^*$ above row i, we similarly compute the minimum cost $c(v, w)$ of all alignments visiting (v, w) but *not* the cell in (v, w) that $\mathcal{A}_1^*$ visits. For a given cell in (v, w) at which an *optimal suffix* alignment visits, we consider all the possible extension for the prefixes of sequences. The cost for entry (v, w) is

$$c(v,w) \;:=\; \begin{cases} \min\Big\{h_V(v,w)\ ,\ h_H(v,w)\Big\} & \text{if } \mathcal{A}_1^* \text{ visits } D_{\mathcal{B}}(v,w), \\ \min\Big\{h_D(v,w)\ ,\ h_H(v,w)\Big\} & \text{if } \mathcal{A}_1^* \text{ visits } V_{\mathcal{B}}(v,w), \\ \min\Big\{h_D(v,w)\ ,\ h_V(v,w)\Big\} & \text{if } \mathcal{A}_1^* \text{ visits } H_{\mathcal{B}}(v,w), \end{cases}$$

where

$$\begin{aligned} h_D(i,j) &:= D_{\mathcal{B}}(i,j) \;+\; y(i,j), \\ h_V(i,j) &:= V_{\mathcal{B}}(i,j) \;+\; \min\Big\{V_{\mathcal{F}}(i-1,j)-\gamma\ ,\ y(i,j)\Big\}, \\ h_H(i,j) &:= H_{\mathcal{B}}(i,j) \;+\; \min\Big\{H_{\mathcal{F}}(i,j-1)-\gamma\ ,\ y(i,j)\Big\}, \quad \text{and} \\ y(i,j) &:= \min\Big\{M_{\mathcal{F}}(i-1,j)\ ,\ M_{\mathcal{F}}(i,j-1)\ ,\ M_{\mathcal{F}}(i-1,j-1)\Big\}. \end{aligned}$$

Note that we cover all the cases. The cost of $\mathcal{A}_2^*$ is the minimum of all a, b, and c.

Filling out the backward table takes $O(mn)$ time, computing all a, b, and c takes $O(m + n)$ time, and recovering the actual alignment takes $O(m+n)$ time. Therefore this algorithm takes $O(mn)$ time to compute a next-best alignment of two sequences of length m and n. □

Inverse Near-Optimal Alignment

A separation algorithm for each variation of Inverse Near-Optimal Alignment can be obtained from the separation algorithm for Inverse Optimal Alignment (Algo-

rithm 4.2) by replacing Inequality (4.1) in Step (2) with another inequality, which will be given below. The correctness of the resulting separation algorithm then can be shown with the exact same argument used for the correctness of Algorithm 4.2. In the following we explain how the separation algorithm for Inverse Optimal Alignment is modified to obtain a separation algorithm for each Inverse Near-Optimal Alignment variation.

Absolute error criterion The system L_i of inequalities associated with example $\mathcal{A}_i$ resulting from the reduction for Inverse Alignment under Absolute Error consists of the following inequalities:

$$f_x(\mathcal{A}_i) \; - \; f_x(\mathcal{B}) \;\; \leq \;\; \epsilon_i, \tag{4.3}$$

where $\mathcal{B}$ is any alignment of the sequences in the set $\mathcal{S}_i$, x is a value assignment of alignment parameters, and ϵ_i is a value for the upper bound on the absolute error of the cost of example $\mathcal{A}_i$. We obtain a separation algorithm for the absolute error criterion from Algorithm 4.2 by replacing Inequality (4.1) in Step (2) with Inequality (4.3).

Theorem 4.4 (Correctness of Absolute Error Criterion Separation) *The separation algorithm obtained by replacing Inequality (4.1) in Step (2) of Algorithm 4.2 with Inequality (4.3) separates a value vector (x, ϵ_i) for the system L_i of inequalities resulting from the reduction of Inverse Alignment under Absolute Error.* □

The proof for this theorem can be obtained by the exact same argument used for the correctness of Algorithm 4.2 and is omitted.

Relative error criterion The system L_i of inequalities associated with example $\mathcal{A}_i$ resulting from the reduction for Inverse Alignment under Relative Error

consists of the following inequalities:

$$f_x(\mathcal{A}_i) \;\leq\; (1+\epsilon)\, f_x(\mathcal{B}), \tag{4.4}$$

where $\mathcal{B}$ is any alignment of the sequences in the set $\mathcal{S}_i$, x is a value assignment of alignment parameters, and ϵ is a constant that bounds from above the maximum relative error of costs of all examples $\mathcal{A}_1 \ldots, \mathcal{A}_k$. We obtain a separation algorithm for the relative error criterion from Algorithm 4.2 by replacing Inequality (4.1) in Step (2) with Inequality (4.4).

Theorem 4.5 (Correctness of Relative Error Criterion Separation) *The separation algorithm obtained by replacing Inequality (4.1) in Step (2) of Algorithm 4.2 with Inequality (4.4) separates a value vector x for the system L_i of inequalities resulting from the reduction of Inverse Alignment under Relative Error.* □

The proof for this theorem can be obtained by the exact same argument used for the correctness of Algorithm 4.2 and is omitted.

Discrepancy error criterion The system L_i of inequalities associated with example $\mathcal{A}_i$ resulting from the reduction for Inverse Alignment under Discrepancy Error consists of the following inequalities:

$$f_x(\mathcal{A}_i) \;-\; f_x(\mathcal{B}) \;+\; d(\mathcal{A}_i, \mathcal{B}) \;\leq\; \epsilon_i, \tag{4.5}$$

where $\mathcal{B}$ is any alignment of the sequences in the set $\mathcal{S}_i$, x is a value assignment of alignment parameters, and ϵ_i is a value for the upper bound on the discrepancy error of example $\mathcal{A}_i$.

Algorithm 4.5 (Separation for Discrepancy Error Criterion)

(1) Compute an optimal alignment $\mathcal{B}^*$ of the sequences in $\mathcal{S}_i$ under the function $f_x(\cdot) - d(\mathcal{A}_i, \cdot)$ with parameter choice x.

(2) If Inequality (4.1) does not hold for $\mathcal{B} = \mathcal{B}^*$, return it as a violated inequality and halt.

(3) Report that x is an optimal solution to the linear program. □

Theorem 4.6 (Correctness of Algorithm 4.5) *Algorithm 4.5 separates a value vector (x, ϵ_i) for the system L_i of inequalities resulting from the reduction of Inverse Alignment under Discrepancy Error.*

Proof Let $\mathcal{B}^*$ be the optimal alignment computed in Step (1). We show that Inequality (4.5) holds for $\mathcal{B} = \mathcal{B}^*$ if and only if the inequality holds for every alignment $\mathcal{B}$ of the sequences in $\mathcal{S}_i$. ($\Rightarrow$) If Inequality (4.5) holds for $\mathcal{B} = \mathcal{B}^*$, then the inequality holds for every alignment $\mathcal{B}$ of the sequences in $\mathcal{S}_i$ because $f_x(\mathcal{B}^*) - d(\mathcal{A}_i, \mathcal{B})$ is the smallest of all $f_x(\mathcal{B}) - d(\mathcal{A}_i, \mathcal{B})$. ($\Leftarrow$) If Inequality (4.5) holds for all alignments $\mathcal{B}$ of the sequences in $\mathcal{S}_i$, then the inequality holds $\mathcal{B} = \mathcal{B}^*$ because $\mathcal{B}^*$ is one of the alignments $\mathcal{B}$. Therefore the algorithm separates $(x, epsilon_i)$ for the linear program with the system L_i. □

We give a description of an algorithm that given an alignment $\mathcal{A}$ finds an optimal alignment of two sequences under function $f(\cdot) - d(\mathcal{A}, \cdot)$, where f is an linear cost function with affine gap costs, d is the recovery error function defined in Definition 2.5. Note that this algorithm differs from Algorithms 2.1 and 4.4 in the sense that this algorithm takes as input an alignment, not a pair of sequences. The algorithm is, however, a trivial modification of Algorithm 2.1 which computes an optimal alignment under f.

Algorithm 4.6 (Optimal alignment for separation of discrepancy error) Let m and n be the lengths of sequences that the input alignment $\mathcal{A}$ aligns, and k be the number of columns of $\mathcal{A}$. We integrate the successful recovery rate into our recurrences in Equation (2.2) and the boundary conditions for the optimal alignment. By Definition 2.5, the cost of an alignment $\mathcal{B}$ under $f(\cdot) - d(\mathcal{A}, \cdot)$ can be

written as $f(\mathcal{B}) - 1 + c(\mathcal{A}, \mathcal{B})/k$, where $c(\mathcal{A}, \mathcal{B})$ is the number of columns common in both $\mathcal{A}$ and $\mathcal{B}$. Note that the constant term is of no importance for minimization of the function. We rewrite a new cost function without it:

$$g_{\mathcal{A}}(\mathcal{B}) \;\; := \;\; f(\mathcal{B}) \;+\; \frac{1}{k}\, c(\mathcal{A}, \mathcal{B}).$$

Our algorithm will find an optimal alignment $\mathcal{A}^*$ under $g_{\mathcal{A}}$. Consider the dynamic programming table filled by Algorithm 2.1 to find an optimal alignment $\mathcal{B}^*$ under f. The input alignment $\mathcal{A}$ goes through a certain path in the table, which is not necessarily the same as the path of $\mathcal{B}^*$. Each *cell* except M, which holds the minimum of all other cells, on the path of $\mathcal{A}$ corresponds to a *column* in $\mathcal{A}$.

Whenever $\mathcal{A}^*$, which is optimal under $g_{\mathcal{A}}$, visits a cell on the path of $\mathcal{A}$, we give the right amount of recovery rate, which is $1/k$, to $\mathcal{A}^*$. This is done by adding the fraction into the recurrences in Equation (2.2) and the boundary conditions in Algorithm 2.1. To do so, we build a binary table $\mathcal{T}$ that records $\mathcal{A}$'s path. Each entry in $\mathcal{T}$ has three cells: $D_{\mathcal{T}}$ for substitution, $V_{\mathcal{T}}$ for deletion, and $H_{\mathcal{T}}$ for insertion. The value of each cell is 1 if $\mathcal{A}$ visits the cell, and 0 otherwise. The recurrences that replace Equation (2.2) for the new cost function $g_{\mathcal{A}}$ are

$$\begin{aligned}
D(i,j) \;\; &:= \;\; M(i-1, j-1) + \sigma(S[i], T[j]) \;+\; (D_{\mathcal{T}}(i,j) \,/\, k)\,, \\
V(i,j) \;\; &:= \;\; \min\Big\{M(i-1,j) + \gamma\,,\, V(i-1,j)\Big\} + \lambda \;+\; (V_{\mathcal{T}}(i,j) \,/\, k)\,, \quad \text{and} \\
H(i,j) \;\; &:= \;\; \min\Big\{M(i,j-1) + \gamma\,,\, H(i,j-1)\Big\} + \lambda \;+\; (H_{\mathcal{T}}(i,j) \,/\, k)\,,
\end{aligned}$$

and the boundary conditions which are changed are

$$\begin{aligned}
V(i,0) \;\; &:= \;\; \gamma \;+\; i\,\lambda \;\;+\;\; (V_{\mathcal{T}}(i,j) \,/\, k)\,, \quad \text{and} \\
H(0,j) \;\; &:= \;\; \gamma \;+\; j\,\lambda \;\;+\;\; (H_{\mathcal{T}}(i,j) \,/\, k)\,,
\end{aligned}$$

for $1 \le i \le m$ and $1 \le j \le n$. The rest of them are the same as before.

Filling out the binary table takes $\mathrm{O}(mn)$ time, evaluating the recurrences takes $\mathrm{O}(mn)$ time, and recovering the optimal alignment under $g_{\mathcal{A}}$ takes $\mathrm{O}(m+n)$ time. Therefore the algorithm runs in $\mathrm{O}(mn)$ time. □

4.1.3 Theoretical algorithm

Section 4.1.1 presented a cutting plane algorithm which solves all of our linear programming formulations of Inverse Alignment. Section 4.1.2 gave a polynomial time separation algorithm for each of these programming formulations. This section shows the existence of a polynomial time algorithm for a linear programming formulation of Inverse Alignment if the alignment cost function is linear in its alignment parameters, values for the parameters are bounded, and the separation algorithm for the linear programming formulation runs in polynomial time. Combined with Theorem 3.4, this proves our main result.

Theorem 4.7 (Complexity of Inverse Alignment) *Inverse Optimal Alignment and Inverse Near-Optimal Alignment can be solved in polynomial time for any form of alignment in which:*

(1) *the alignment cost function is linear in its alignment parameters,*
(2) *the parameter values can be bounded, and*
(3) *for any choice of parameter values, an optimal alignment under a given function can be found in polynomial time.*

Inverse Unique-Optimal Alignment can be solved in polynomial time if in addition, for any choice of parameter values, a next-best alignment can be found in polynomial time.

Proof Assume conditions (1) and (2) are true. Consider an instance of Inverse Near-Optimal Alignment with t alignment parameters and k examples. Consider the linear program with a system L of inequalities reduced for this instance of inverse alignment. Note that all coefficients in L are rational. In other words, L describes a bounded rational polyhedron. Assume condition (3) holds. Given the system L and a choice x of parameter value, the separation algorithm in Section 4.1.2 either finds a violated inequality in L or concludes that x satisfies all the

inequalities in L in $O(tk)$ time. By Theorem 3.4 we can solve the linear program in polynomial time. (The proof for Inverse Optimal Alignment can be given with the exact same argument.) Assume that a next-best alignment can be found in polynomial time. The separation algorithm for Inverse Unique-Optimal Alignment in Section 4.1.2 either finds a violated inequality in L or concludes x satisfies all inequalities in L in $O(tk)$ time. By Theorem 3.4, we can solve the corresponding linear program in polynomial time. □

4.2 Implementation issues

To obtain a practical algorithm for a particular form of alignment several issues must be worked out. These issues include

- how to find an *initial subset* of the inequalities that yields a bounded linear program (Algorithm 4.1)
- how to eliminate *degenerate solutions* that score all alignments of the same sequences the same (Sections 3.3.1 and 3.3.3),
- how to manage examples to find a *violated inequality* fast (Section 4.1.2), and
- how to manage inequalities in a system of inequalities to have a *small size* linear program.

In this section we discuss these issues in the context of the standard cost model of two sequences given in Section 3.1, in which substitutions between pairs of letters are arbitrary and gaps are scored by the gap-open and gap-extension parameters. Similar ideas, however, may apply to other forms of alignment as well. All the issues mentioned below are currently implemented in a software tool called `IPA` [35], which we used for the experimental results in Chapter 7.

4.2.1 Bounding inequalities

The cutting plane algorithm (Algorithm 4.1) assumes that the linear program with the initial subset S of inequalities is bounded. To use the algorithm, this assumption should hold. In a system of inequalities, the inequalities that together make the linear program bounded are called *bounding inequalities*. To find bounding inequalities, we separately consider two forms of the alignment cost function: when all parameters are *free*, and when substitution parameters are *fixed* and gap parameters are free. Because the values of the parameters must be nonnegative, inequalities $x \geq 0$ are always added to the initial set for the cutting plane algorithm in both cases.

When substitution and gap parameters are all *free*, we can set absolute upper limits such as 1 on the values of the parameters. Then all parameters lie in the bounding box $0 \leq x \leq 1$, which are together bounding inequalities. We take them 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. In general, the linear program is bounded if and only if there exists (1) an inequality with a negative slope, or (2) two inequalities D and U 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. 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. With further analysis, one can find inequalities D and U in $O(1)$ time.

4.2.2 Degenerate solutions

In general all Inverse Alignment formulations except Inverse Unique-Optimal Alignment, when applied to global sequence alignment, have a set $\mathcal{F}$ of *degenerate solutions*,

$$(\sigma_{ab}, \gamma, \lambda) \;\in\; \mathcal{F} \;:=\; \{(2c, 0, c) \;:\; c \geq 0\}\,.$$

Every parameter choice $x \in \mathcal{F}$ from this set makes *all* global alignments of the sequences that an example aligns score the *same*, namely $c(m+n)$ for two sequences of lengths m and n. Such an x makes each example be an optimal alignment, hence such an x is an optimal solution for every instance of Inverse Alignment (possibly except under the discrepancy error criterion). Of course these degenerate solutions are not of biological interest. To eliminate them, we use the following approach.

Let *nondegeneracy threshold* τ be the difference between the expected cost of a random substitution (of different letters) and the expected cost of a random identity, measured for some default substitution cost matrix. For a sound scoring scheme that distinguishes between substitutions and identities, this difference should be positive. The value of τ for the commonly used `BLOSUM` [27] and `PAM` [10] matrices for standard amino acid frequencies is around 0.4 when substitution scores are translated and scaled to costs in the range $[0, 1]$.

Given a value for threshold τ, we add to the linear program the *nondegeneracy inequality*,

$$\frac{\sum_{a<b} p_a p_b\, \sigma_{ab}}{\sum_{a<b} p_a p_b} \;-\; \frac{\sum_a p_a^2\, \sigma_{aa}}{\sum_a p_a^2} \;\geq\; \tau, \tag{4.6}$$

where in the summations a and b range over all amino acids, and p_a is the probability of amino acid a appearing in a random protein sequence. Note that this is a linear inequality in parameters σ_{ab} and σ_{aa}.

Inequality (4.6) is equivalent to requiring that for the learned parameters, the difference between the expected score of a random substitution and the expected score of a random identity is at least τ. When τ is positive, which holds for standard

cost schemes, this inequality cuts off the degenerate solution $(\sigma_{ab}, \gamma, \lambda) = (2c, 0, c)$, as the following theorem states. Furthermore, desirable solutions (such as the cost scheme from which τ was measured) are not eliminated.

Theorem 4.8 (Eliminating degeneracy) *When degeneracy threshold $\tau > 0$, Inequality (4.6) eliminates all degenerate solutions $x \in \mathcal{F}$.*

Proof The degenerate solutions have $\sigma_{ab} = \sigma_{aa} = 2c$, so their expected substitution and identity scores are equal. The nondegeneracy inequality then reduces to $\tau \leq 0$, contradicting $\tau > 0$. In other words, every solution from $\mathcal{F}$ violates Inequality (4.6). □

In essence, the nondegeneracy inequality forces the substitution and identity costs in the optimal solution x^* to the linear program to be at least as nondegenerate as the default substitution cost matrix from which τ was measured. In Chapter 7, we show that the value of degeneracy threshold τ is robust across different `BLOSUM` matrices and use the value of τ corresponding to `BLOSUM`, namely $\tau = 0.4$.

We emphasize that the nondegeneracy inequality is crucial, since without it algorithms for Inverse Optimal and Near-Optimal Alignment immediately output the optimal trivial solution $x = (0, \ldots, 0)$.

4.2.3 Example management

To find new violated inequalities without solving too many Optimal Alignments, we need to manage the order of examples. We use the following heuristic for this purpose.

Given a set of examples $\mathcal{A}_i$, we maintain a circular list L of all $\mathcal{A}_i$ whose separation algorithm was not satisfied when $\mathcal{A}_i$ was last considered. When an $\mathcal{A}_i$ is examined on L, if $\mathcal{A}_i$ does not generate a violated inequality, it is removed from L. Otherwise, it stays. When L becomes empty, we reset L to be all $\mathcal{A}_i$, and perform

another scan to verify that all $\mathcal{A}_i$ are satisfied by the current solution x. When all are satisfied, the procedure has found an optimal solution, and terminates. This heuristic examines $\mathcal{A}_i$ that continue to generate violated inequalities on the assumption that $\mathcal{A}_i$ that were satisfied will continue to be satisfied on adding new inequalities. In Chapter 7, we illustrate solving an instance of Inverse Alignment using this heuristic for fixed substitution parameters.

4.2.4 Inequality management

Notice that the cutting plane algorithm in Section 4.1.1 adds violated inequalities returned by the separation algorithm in each iteration step. A linear program solver solves this ever growing linear program until no violated inequality is reported. As the size of the linear program grows, solving the linear program takes more time due to the large system of inequalities. An optimal solution x to a linear program with t variables is, however, decided by t inequalities in the system of the linear program which intersect each other at point x. To reduce the time spent by a linear program solver, we take the following approach.

Two constants $\alpha \geq 1$ and $\beta \geq \alpha$ are maintained. Each time we solve a linear program and find an optimal solution x, the system of inequalities is tested to see if it contains more than βt inequalities. If it does not, we proceed to the cutting plane algorithm in the usual manner. Otherwise, we measure the shortest distance from each inequality in the system to the point x and sort all the distances in a nondecreasing order to choose the first αt inequalities in the sorted list. Note that since the inequalities defining x touch the solution x, they are all included in this choice.

Summary of the chapter We gave a polynomial time separation algorithm for each linear programming formulation of Inverse Alignment and proved the correctness of these algorithms (Section 4.1.2). We gave a polynomial time theoreti-

cal algorithm (Section 4.1.3) and a practical cutting plane algorithm which is not guaranteed to run in polynomial time but is fast in practice (Section 4.1.1). We addressed important issues which arise when the cutting plane algorithm is implemented (Section 4.2). In the next chapter, we rigorously address an important issue of examples available today, namely the issue that the examples used to learn alignment parameters from are not perfect. We formalize this issue as a problem and present an iterative approach to this problem which works well in practice.

CHAPTER 5

EXTENSION TO PARTIAL EXAMPLES

All of our inverse alignment formulations assumed the input alignments are reliable; the examples are good to learn alignment parameters from. In reality the best multiple alignments of protein sequences available today are not perfect and certain parts of these alignments are left unspecified. We formally address this issue on the input alignments and formalize the issue as a new problem. We extend all variations of Inverse Near-Optimal Alignment to learn parameter values from *partially reliable* examples with an iterative approach. We show the new iterative scheme guarantees to improve the error under absolute, relative, and discrepancy criteria.

5.1 Partial examples

For Inverse Alignment of protein sequences, the best examples that are available come from multiple sequence alignments of protein families that are determined by aligning the three-dimensional structures of family members. Several suites of such benchmark alignments are now available [42, 4, 5, 57] and are widely used for evaluating the accuracy of software for multiple alignment of protein sequences. Most all of these benchmark alignments, however, are partial alignments. The benchmark alignment has regions that are reliable and where the alignment is specified, but between these regions the alignment of the sequences is effectively left unspecified. These reliable regions are usually the *core blocks* of the multiple sequence alignment, which are typically gapless sections of the alignment where structure is conserved across the family. For our purpose, we define *partial example*, and *reliable regions* and *unreliable regions* in an alignment formally.

```
eea-kq-------iKPNLDYPAGPTYNLmgf-DTEMFTPLFIAARITGWTAHIMEQVAdnalir-plseyngpeqrqvp
eey-lsk---k-giSINVDYWSGLVFYGmki-PIELYTTIFAMGRIAGWTAHLAEYVShnr-iirprlqyvgeigkkyl
gkv-lnp---r-giYPNVDFYSGVVYSDlgf-SLEFFTPIFAVARISGWVGHILEYQEldnrllrpgakyvgeldvpyv
ikq-fss---k-giYPNTDFYSGIVFYAlgf-PVYMFTALFALSRTLGWLAHIIEYVEeqhrlirpralyvgpe-----
lndpyfi--ek-klYPNVDFYSGIILKAmgi-PSSMFTVIFAMARTVGWIAHWSEMHSdgmkiarprqlytgyekrdfk
pnv-lleqgkaknpWPNVDAHSGVLLQYygmtEMNYYTVLFGVSRALGVLAQLIWSRAlgfplerpksmstdg------
pnv-lleqgkaknpWPNVDAHSGVLLQYygmtEMNYYTVLFGVSRALGVLAQLIWSRAlgfplerpksmstag------
pnv-lleqgaaanpWPNVDAHSGVLLQYygmtEMNYYTVLFGVSRALGVLAQLIWSRAlgfplerpksmstdg------
```

Figure 5.1: An alignment containing unreliable regions.

Definition 5.1 (Partial Example) Assume an alignment $\mathcal{A}$ of column length ℓ is given with an indicator vector $v = (v_1, \ldots, v_\ell)$ where v_i is 1 if column i of $\mathcal{A}$ is reliable, and 0 otherwise. A *reliable region* of alignment $\mathcal{A}$ is a maximal run of reliable columns. An *unreliable* region of $\mathcal{A}$ is a maximal run of unreliable columns. An example whose indicator vector does not have a member with value 0 is called a *complete example*. Otherwise the example is called a *partial example*. □

Note that in this definition, a partial example may contain a gap in its reliable region. Consequently, partial examples do not simply specify which pairs of positions to align by substitution, but may also specify which positions should be in gaps. To avoid confusion, a barred alignment symbol denotes a partial example.

Figure 5.1 shows a part of an alignment of 8 sequences in a protein family with `SCOP` identifier `a.102.5.1` in `PALI` benchmark suites [5]. The columns in lowercase letters are those whose corresponding members in an indicator vector are 0. Otherwise, the columns are written in capital letters. This alignment is hence by Definition 5.1 a partial example, and it contains two reliable regions and three unreliable regions. Note that the type of regions alternate. In this example reliable regions do not contain a gap and show strong similarities among the sequences.

When learning parameters by Inverse Alignment from partial examples, we treat the unreliable regions as missing information, and such regions do not specify the alignment of the sequences. Given a partial example, we can think an alignment

of the sequences that the partial example aligns and all the reliable regions of the partial example agree with the alignment. We formally define such an alignment.

Definition 5.2 (Completion) Given a partial example $\overline{\mathcal{A}}$ of sequences $S_1, \ldots, S_\ell$ with an indicator vector, a *completion* $\mathcal{A}$ of $\overline{\mathcal{A}}$ is a complete example that aligns sequences $S_1, \ldots, S_\ell$ and agrees with all the reliable regions of $\overline{\mathcal{A}}$. □

In other words, a completion $\mathcal{A}$ can change $\overline{\mathcal{A}}$ on the substrings that are in unreliable regions, but must not alter $\overline{\mathcal{A}}$ in reliable regions. Note that in general a partial example does not have a unique completion.

We define inverse alignment from partial examples as the problem of finding the optimal parameter choice over all possible completions of the examples.

Problem 5.1 (Inverse Alignment from Partial Examples) Given a set of partial examples $\overline{\mathcal{A}}_i$, each of which aligns sequences in a set $\mathcal{S}_i$ and has its indicator vector, for $1 \leq i \leq k$, an alignment cost function f_w with alignment parameters w, and a parameter domain D, find a choice x^* of parameter values for w:

$$x^* \;:=\; \arg\min_{x \in D} \min_{\mathcal{A}_1, \ldots, \mathcal{A}_k} E\left(x, \{\mathcal{A}_1, \ldots, \mathcal{A}_k\}\right), \tag{5.1}$$

where error measure function E is E_{abs}, E_{rel}, or E_{dis}. □

In other words, vector x^* minimizes the error measure function E over all completions of the partial examples. We make a few remarks on this problem definition. First, this formulation finds parameter values for which optimally aligning the unreliable regions yields a completion that costs as close as possible to an optimal unconstrained alignment of the example sequences. Second, even when the reliable regions are all gap-free (which is common when partial examples are obtained from the standard suites from benchmark protein alignments), this formulation still learns useful values for gap penalties, as appropriate values for gaps in the unreliable regions are necessary for the reliable regions to be part of alignments that

have close to optimal cost. Third, note that this formulation for partial examples contains the prior formulations on complete examples as special case. Fourth, in the discrepancy error criterion the unreliable columns should not be taken into account for the recovery error function d (Definition 2.5), as these columns are not trustworthy and the recovery on these columns is not meaningful.

In the next section we tackle this problem by solving a series of Inverse Alignments on complete examples.

5.2 Iterative scheme

Inverse Alignment from Partial Examples involves optimizing over all possible completions of the examples. While for partial examples we do not know how to efficiently find an optimal solution, we present a practical iterative scheme, which as demonstrated in Chapter 7, finds a good solution.

Note that for a given partial example there could be exponentially many completions in the sequence lengths. Under a fixed choice of parameter values, it is of interest to find a completion of the example with the minimum cost.

Problem 5.2 (Optimal Completion) Given a partial example $\overline{\mathcal{A}}$ with its indicator vector, and an alignment cost function f_x with parameter values x, find a completion that is optimal under f_x. □

A solution to Optimal Completion is called an *optimal completion*. Given a partial example and a choice of parameter values, one can solve the problem by solving a series of Optimal Alignment with substrings in each unreliable region of the partial example and then concatenating the resulting optimal alignments of substrings and reliable regions of the partial example together. Optimal Alignment should not charge a sequence S for opening a gap at the beginning (or end) of an alignment if S is a substring in a unreliable region, and the left-hand-side (or right-hand-side) neighboring region has a gap opened by S at the right (or left) end.

(Notice that the adjacent neighbors of a unreliable region are reliable regions.)

We describe a practical iterative algorithm for Inverse Alignment from Partial Example. Again, it is does not find an optimal solution but it finds a good solution. The initial completion in this algorithm is either a *default completion* in which a standard substitution cost matrix [27] and carefully chosen gap parameters are used for the optimal completion or the *trivial completion* which means the partial example is used as a completion.

Algorithm 5.1 (Iterative scheme for partial examples) Given a threshold value ϵ,

(1) Set $j = 0$ and $e^{(0)} = \infty$.

(2) Compute initial completion $\mathcal{A}_i^{(0)}$ for each partial example $\overline{\mathcal{A}}_i$.

(3) Repeat the followings until the termination condition is met.

   (a) Compute an optimal parameter choice $x^{(j)}$ by solving Inverse Alignment on complete examples $\mathcal{A}_i^{(j-1)}$.

   (b) Set $e^{(j)} = E\left(x^{(j)}, \left\{\mathcal{A}_i^{(j-1)}\right\}\right)$.

   (c) If $e^{(j)}/e^{(j-1)} \leq \epsilon$, report $x^{(j)}$ as a solution and halt.

   (d) Compute an optimal completion $\mathcal{A}_i^{(j)}$ of each example $\overline{\mathcal{A}}_i$ by solving Optimal Completion under parameter choice $x^{(j)}$.

   (e) Increase j by 1. □

Algorithm 5.1 alternates Step (a) of finding an optimal choice of parameter values in and Step (d) of finding an optimal completion of each partial example, yielding a sequences of parameter choices and completions,

$$\left\{\mathcal{A}_i^{(0)}\right\} \longmapsto x^{(1)} \longmapsto \left\{\mathcal{A}_i^{(1)}\right\} \longmapsto x^{(2)} \longmapsto \ldots$$

As the following result shows, the error of successive parameter estimates decreases monotonically for all three error criteria.

Theorem 5.1 (Error monotonicity for partial example) *For the iterative scheme for Inverse Alignment from Partial Examples (Algorithm 5.1), denote the error in cost for iteration $j \geq 1$ by*

$$e_j \;:=\; E\left(x^{(j)}, \left\{\mathcal{A}_i^{(j-1)}\right\}\right),$$

where function E is error function E_{abs}, E_{rel}, or E_{dis}. Then

$$e_1 \;\geq\; e_2 \;\geq\; \cdots \;\geq\; e^*,$$

where e^* is the optimum error for Inverse Alignment from Partial Examples with example $\overline{A}_i$ under error criterion E.

Proof Since $\mathcal{A}_i^{(j)}$ is an optimal completion of $\overline{A}_i$ with respect to parameters $x^{(j)}$,

$$f_{x^{(j)}}\left(\mathcal{A}_i^{(j-1)}\right) \;\geq\; f_{x^{(j)}}\left(\mathcal{A}_i^{(j)}\right).$$

In other words, under parameters $x^{(j)}$, the new completions $\mathcal{A}^{(j)}$ cost just as close to optimal as the old completions $\mathcal{A}^{(j-1)}$. (Note that the inequality holds for the discrepancy error criterion because any completion has zero recovery error.) This implies that with respect to error

$$E\left(x^{(j)}, \left\{\mathcal{A}_i^{(j-1)}\right\}\right) \;\geq\; E\left(x^{(j)}, \left\{\mathcal{A}_i^{(j)}\right\}\right).$$

So for the new completions $\mathcal{A}_i^{(j)}$, error e_j is achievable, as witnessed by $x^{(j)}$. Since the optimum error for $\mathcal{A}_i^{(j)}$, given by the new parameters $x^{(j+1)}$, cannot be worse,

$$e_j \;\geq\; e_{j+1}.$$

Furthermore e^* lower bounds the error for all completions. □

Corollary 5.1 (Error convergence for partial examples) *Theorem 5.1 implies that the error of the iterative scheme for Inverse Alignment from Partial Examples converges to some constant bounded from below.*

Proof Since the error across iterations forms a nonincreasing sequence which is bounded from below, the error converges to some constant bounded from below.

□

Note that Corollary 5.1 only says that the error converges to some value, not necessarily to the optimum error e^*. Therefore Algorithm 5.1 does *not* solve Problem 5.1.

By Corollary 5.1 the termination condition in Step (3) of Algorithm 5.1 is not required. As witnessed in Chapter 7, the error approaches its lower bound fast (within a few iterations) but it takes many iterations to reach the bound. In practice the terminating condition is useful for most of the cases because the improvement on the error over many iterations is not significant. The termination condition may take different forms. For example, the difference between the errors of two consecutive iterations, or a bound on the number of iterations, or both can be used.

As the error on completions improves across the iterations, the recovery of partial examples *tends* to improve under all variations of Inverse Near-Optimal Alignment as shown in Chapter 7. The reason that the recovery is not guaranteed to improve over iterations is that the recovery function is not linear in alignment parameters and the recovery error cannot be expressed in terms of the variables in our linear programming formulations.

A well-chosen initial completion can reduce the final error as demonstrated later in Chapter 7. The best choice would be the parameter values learned in the last iteration. This makes Algorithm 5.1 off-line; if the parameter values learned in the last iteration are used for the initial completions of the same partial examples in the next learning session.

Summary of the chapter We formalized the problem of finding a choice of parameter values from unreliable real-world sequence alignments (Section 5.1). While we do not know how to solve Inverse Alignment from Partial Examples, we gave a

practical iterative scheme, which finds a good choice of alignment parameter values, and proved the error measure under the absolute, relative, and discrepancy error criteria is guaranteed to improve over the iterations (Section 5.2). In the following chapter, we present another extension of Inverse Alignment which uses secondary structure prediction on the input protein molecules to improve the quality of the parameter values under the error and recovery. We introduce a few alignment cost models which incorporate the prediction information.

CHAPTER 6

ALIGNMENT COST MODELS WITH SECONDARY STRUCTURE

We introduce new alignment cost models based on the secondary structure prediction of input protein sequences to improve the accuracy of protein sequence alignment. These models reflect the impact of residues within certain structures being aligned to substitution costs and the disruption of structures in different contexts to gap costs. By solving the inverse alignment problems based on these cost models, parameter values for substitution and gap parameters can be directly learned. The secondary structure of protein sequence is reviewed for this purpose. We show that finding an optimal alignment under most gap contexts does not increase the time and space complexity, and the time complexity under the most computationally challenged gap context increases by a sublinear factor.

6.1 Protein secondary structure

Studies have showed that the protein structure is more conservative than the sequence itself [8, 51, 50]. As a result, our goal is to incorporate protein secondary structure information into alignment scoring models to improve the accuracy of multiple alignments. For this purpose we review the protein structure hierarchy and the secondary structure in detail.

A protein molecule consists of amino acid residues. In a protein molecule, an amino acid binds to an adjacent residue by forming a peptide bond with dehydration. A polypeptide chain is a chain of amino acid residues linked by these peptide bonds in three-dimensional space. This chain is not a linear structure but folds as it comes into existence in three-dimensional space. The protein structure is important because it determines the function of the protein. Therefore understanding

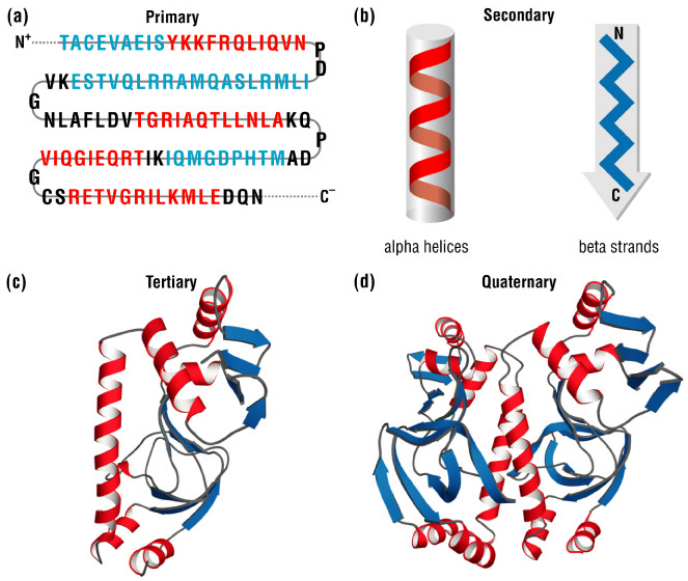

Figure 6.1: The hierarchy of protein structures [49].

---

and possibly predicting the structure of protein sequences are important tasks in computational molecular biology.

Protein structure is hierarchically organized from primary structure to quaternary structure as shown in Figure 6.1 [49]. The *primary structure* is the sequence of residues in the polypeptide chain or simply the protein sequence (Figure 6.1(a)). The *secondary structure* is a highly regular structure in a molecule that is locally defined (Figure 6.1(b)). In secondary structure, the three-dimensional location information of atoms is not specified, rather residues are classified using three basic structural types: *α-helix*, *β-sheet*, and *loop*. (There is also a scheme involving eight types that is sometimes used.) On this level of the hierarchy, the protein molecule is a linear sequence over structures of α-helices, β-sheets, and loops. For example, in Figure 6.1(a) each maximal run of residues in red (dark) corresponds to an α-helix, that in blue (light) corresponds to a β-sheet, and that in black (darkest) corresponds to a loop. In Figure 6.1(c), loop is represented as gray threads connecting structures of α-helices or β-sheets. In this chapter we employ the sec-

ondary structure with three basic structural types in our alignment cost models. The *tertiary structure* results from folding the secondary structures into a three-dimensional configuration (Figure 6.1(c)). The stability of the resulting structure is caused by hydrophobic chemical forces dictating that amino acids with nonpolarity must be guarded from water by burying them in the core. As a result, α-helix and β-sheet are stable structures, located in the core of the folded molecule, whereas loop usually is outside the core, exposed to water under the surface of the molecule. The *quaternary structure* associates at least two tertiary structured polypeptides (Figure 6.1(d)). Not every protein forms quaternary structure.

In the following we consider an alphabet Γ of the standard three protein structural types α-helix, β-sheet, and loop, which are respectively denoted by α, β, and ϕ.

6.2 Secondary-structure-based cost models

We introduce several cost models for an alignment of two protein sequences that make use of predicted secondary structures for the sequences. (These alignment cost schemes can be used for standard multiple sequence alignment models such as sum-of-pairs multiple alignment or tree alignment.)

6.2.1 Alignment cost function

Like the standard cost model we considered so far, our alignment cost function f in these models is linear in its parameters, and an optimal alignment minimizes given f. Before new cost models are introduced, we define two terms related to gap parameters. As defined in Section 2.2, a gap is a maximal run of either insertions or deletions. A gap is called *external* if it inserts or deletes a prefix or a suffix of sequences, and *internal* otherwise. (The distinction between internal and external gaps here is not related to the predicted secondary structure being incorporated into the models; this refinement is to improve the quality of gap parameters. Internal gap

parameters introduced below are, however, learned from the predicted secondary structures of the input sequences.)

The alignment parameters used in these models are

- *substitution* parameters σ_{ab}, charged for a substitution involving the unordered pair of amino acids a and b
- *modifier* parameters μ_{uv}, charged for a substitution involving the unordered pair of secondary structural types u and v
- *gap-open* parameters γ_i and *gap-extension* parameters λ_i for $1 \leq i \leq \ell$, charged for *internal gap*, and
- a *gap-open* parameter $\widetilde{\gamma}$ and a *gap-extension* parameter $\widetilde{\lambda}$, charged for *external gap*,

where ℓ is some positive number. The first two parameter types are discussed later in Section 6.2.2, and the third parameter type together with ℓ are discussed in Section 6.2.3 in detail. There are 210 unordered pairs of letters over the protein alphabet Σ of 20 amino acids, and 6 unordered pairs of letters over the alphabet $\Gamma = \{\alpha, \beta, \phi\}$ of three predicted secondary structural types. Except the modifier parameters, values for parameters are always *nonnegative*. The number of parameters is $218 + 2\ell$ for these models for $\ell \geq 1$.

Substitutions and gaps are scored in a *position-dependent manner* that takes into account the predicted secondary structure. Therefore, unlike the standard alignment cost model, the positions of residues need to be specified in an alignment cost function f for these models. Assume that an alignment $\mathcal{A}$ aligns protein sequences A and B. A substitution in alignment $\mathcal{A}$ of residues $A[i]$ and $B[j]$ is denoted by a tuple (A, i, B, j). An internal gap is denoted by a tuple (S, i, j, T, k), where sequence pair S, T is either A, B or B, A, and substring $S[i : j]$ is deleted from sequence S and inserted between positions k and $k+1$ of sequence T. An external gap is denoted by a tuple (i, j), where prefix or suffix $S[i : j]$ is deleted

from one of the sequences.

In our alignment cost function f, the cost of substitutions is given by function s, the cost of internal gaps is given by function g, and the cost of external gaps is the standard affine gap costs [19]. The form of the alignment cost function f is

$$\begin{aligned} f(\mathcal{A}) \;\; := \;\; & \sum_{\substack{\text{all substitutions} \\ (A,i,B,j) \in \mathcal{A}}} s(A,i,B,j) \\ + \;\; & \sum_{\substack{\text{all internal gaps} \\ (S,i,j,T,k) \in \mathcal{A}}} g(S,i,j,T,k) \\ + \;\; & \sum_{\substack{\text{all external gaps} \\ (i,j) \in \mathcal{A}}} \left(\widetilde{\gamma} \; + \; (j-i+1)\, \widetilde{\lambda} \right), \end{aligned} \tag{6.1}$$

where the sequence pair S, T in the second summation is either A, B or B, A.

We next describe the substitution cost function s and the internal gap cost function g for the secondary structure based models.

6.2.2 Substitution cost function

Consider a substitution of two residues in an alignment, where at these residues are amino acids a and b, and these two residues are involved in secondary structures of types u and v. Function s scores a substitution of amino acids a and b in Σ with secondary structure types u and v in Γ using two costs:

- σ_{ab}, the cost for substituting amino acids a and b, and
- μ_{uv}, an additive *modifier* to the substitution cost that reflects the impact of the residues being in secondary structures of types u and v.

The form of function s in terms of these costs is given below. For better reading, we write $\sigma(a,b)$ for σ_{ab} and $\mu(u,v)$ for μ_{uv} when necessary. In these models, both (a,b) and (u,v) are unordered pairs. This results in 210 substitution costs σ_{ab} plus 6 secondary structure modifiers μ_{uv}, for a total of 216 alignment parameters that must be specified for the substitution scoring model.

We consider two forms of the substitution cost function s, depending on the kind of the secondary structure prediction that is performed on input sequences A and B.

Single prediction

In the simpler setting, at each residue a *single* prediction is made for the secondary structure type of that residue. The predicted secondary structure for protein sequence A can then be represented by a string S_A, where the residue at the ith position of A has predicted type $S_A[i] \in \Gamma$.

For single prediction, the substitution cost function is

$$s(A, i, B, j) \;\; := \;\; \sigma(A[i], B[j]) \; + \; \mu(S_A[i], S_B[j]). \tag{6.2}$$

The modifier $\mu(u, v)$ may be positive or negative. When the residues have the same secondary structure type, $\mu(u, u) \leq 0$, which reduces the cost of the substitution, making it more favorable to align the residues. When the residues have different secondary structure types, $\mu(u, v) \geq 0$ for $u \neq v$, making it less favorable to align them. These constraints on the modifiers can be enforced during the reduction to a linear program as inequalities.

Multiple prediction

The most accurate tools for secondary structure prediction make *multiple* predictions for a residue, outputting a confidence that the residue is in each possible type. For the residue at the ith position of A, we denote the predictor's confidence that the residue is in secondary structure type u by $P_A(i, u) \geq 0$. In practice, we normalize the confidence output of the predictor at the ith residue so that $\sum_{u \in \Gamma} P_A(i, u) = 1$.

For multiple prediction, the substitution cost function is

$$s(A, i, B, j) \;\; := \;\; \sigma(A[i], B[j]) \; + \; \sum_{u, v \, \in \, \Gamma} P_A(i, u) \; P_B(j, v) \; \mu(u, v). \tag{6.3}$$

Note that when the predictor puts all its confidence on one secondary structure type for a residue, this reduces to the single prediction substitution function.

To sum up, all information from the secondary structure prediction is encapsulated in the strings S_A and S_B for the case of single prediction, and the vectors P_A and P_B for the case of multiple prediction. Given a secondary structure prediction and values of substitution related parameters σ_{ab} and μ_{uv}, the substitution cost function s can be evaluated in O(1) time.

6.2.3 Gap cost function

(In this section, we assume that a sequence pair S, T is either A, B or B, A.) With standard affine gap costs [19] the cost of inserting or deleting a substring of length k is $\gamma + \lambda k$, where γ is the gap-open cost (charged per gap), and λ is the gap-extension cost (charged per residue). The new gap cost function generalizes this to an ensemble of gap-open and gap-extension costs whose values depend on the secondary structure around the gap.

In the new models for internal gaps, the basic idea is that the gap-open cost γ depends on a *global measure* of how disruptive the entire gap is to the secondary structure of the proteins, while the gap-extension cost λ charged per residue depends on a *local measure* of disruption at the position of that residue. We define these notions more precisely below.

For an internal gap that deletes substring $S[i:j]$ and inserts it between residues $T[k]$ and $T[k+1]$, the gap cost function g has the general form,

$$g(S,i,j,T,k) \;:=\; \gamma\left(H(S,i,j,T,k)\right) \;+\; \sum_{i \le p \le j} \lambda\left(h(S,p,T,k)\right), \qquad (6.4)$$

where functions H and h are respectively the global and local measures of secondary structure disruption. The first term is a per gap cost and the second term is a sum of per residue costs. Both H and h return integer values in the range $L = \{1, 2, \ldots, \ell\}$ that give the discrete level of disruption. For better reading we write $\gamma(i)$ for γ_i and

$\lambda(i)$ for λ_i when necessary. For ℓ *disruption levels*, the internal gap cost function has 2ℓ parameters γ_i and λ_i that must be specified.

The gap costs at these levels satisfy $0 \leq \gamma_1 \leq \ldots \leq \gamma_\ell$ and $0 \leq \lambda_1 \leq \ldots \leq \lambda_\ell$. In other words, a higher level of disruption incurs a greater gap cost. These constraints on gap parameters can be enforced during the reduction to a linear program as inequalities.

Functions H and h, which measure the level of secondary structure disruption, depend on two aspects of a gap:

- which gap positions are considered when determining the level, and
- how strongly a position is involved in secondary structure.

We call the first aspect the *gap context*, and the second aspect the *degree* of secondary structure. We explain the degree of secondary structure first.

Measuring the degree of secondary structure

As described in Section 6.2.2, the predicted secondary structure for the residues of a protein sequence A may be represented by string S_A of secondary structure types in the case of single prediction, or a vector P_A of confidences for each type in the case of multiple prediction. We consider three ways of using such predictions to determine the degree $\Psi_A(i)$ of involvement in secondary structure at a residue position i in sequence A. This degree Ψ is a value in the range $[0, 1]$, where 0 corresponds to no involvement in secondary structure, and 1 corresponds to full involvement.

We consider the following approach for measuring the degree of secondary structure.

- **Binary-single approach** A single prediction at position i is assumed, and

the output is a binary value for the degree, where

$$\Psi_A(i) \;=\; \begin{cases} 1, & \text{if } S_A[i] \in \{\alpha, \beta\} \\ 0, & \text{otherwise} \end{cases} .$$

- **Binary-multiple approach** A multiple prediction at position i is assumed, and output is a binary value, where

$$\Psi_A(i) \;=\; \begin{cases} 1, & \text{if } P_A(i,\alpha) + P_A(i,\beta) > P_A(i,\phi) \\ 0, & \text{otherwise} \end{cases} .$$

- **Continuous-multiple approach** A multiple prediction at position i is assumed, and output is the real value

$$\Psi_A(i) \;=\; P_A(i,\alpha) + P_A(i,\beta).$$

Specifying the gap context

The gap context is specified by the positions that functions H and h consider when measuring the global and local secondary structure level. Both functions make use of the secondary structure degree $\Psi(i)$ discussed above.

To measure the local disruption level h at position i in a sequence S of length n, we consider a small window $W(i,n)$ of consecutive positions centered around i:

$$h(S,i) \;:=\; \left\langle \sum_{p \in W(i,n)} \Psi_S(p) \Big/ |W(i,n)| \right\rangle, \tag{6.5}$$

where $\langle x \rangle \;:=\; \lfloor (\ell - 1)x \rfloor + 1$. In words, $\langle x \rangle$ maps real value $x \in [0,1]$ to the discrete disruption levels $1, 2, \ldots, \ell$, and the local disruption level h at position i is the average secondary structure degree Ψ for the residues in a window around position i. Generally all windows have the same width $|W(i,n)| = w$, except when i is too close to 1 or n to be centered in a window of width w, in which case $W(i,n)$ shrinks on one side of i.

We consider three different ways of specifying the gap context, depending on whether we take an insertion view, a deletion view, or a mixed view of a gap.

- **Deletion context**

  The disruption is assumed to be caused by the gap in terms of the secondary structure lost by deleting substring $S[i:j]$. For the global disruption measure H, we take the maximum local level of secondary structure over the positions in the deleted substring. This gives the gap cost function,

$$g(S,i,j,T,k) \;:=\; \gamma\left(\max_{i\le p\le j} h(S,p)\right) \;+\; \sum_{i\le p\le j} \lambda\Big(h(S,p)\Big). \tag{6.6}$$

- **Insertion context**

  The disruption is assumed to be in terms of the secondary structure displaced at residues $T[k]$ and $T[k+1]$ where the insertion occurs. For both the global and local measures of disruption this context uses

$$H(T,k) \;:=\; \left\langle \frac{1}{2}\left(\sum_{p\,\in\, W(k,n)} \frac{\Psi_T(p)}{|W(k,n)|} \;+\; \sum_{q\,\in\, W(k+1,n)} \frac{\Psi_T(q)}{|W(k+1,n)|} \right) \right\rangle, \tag{6.7}$$

  which gives the gap cost function,

$$g(S,i,j,T,k) \;:=\; \gamma\Big(H(T,k)\Big) \;+\; (j-i+1)\,\lambda\Big(H(T,k)\Big). \tag{6.8}$$

- **Mixed context** This context combines the global disruption measure H of the insertion context with the local disruption measure h of the deletion context, which give the gap cost function,

$$g(S,i,j,T,k) \;:=\; \gamma\Big(H(T,k)\Big) \;+\; \sum_{i\le p\le j} \lambda\Big(h(S,p)\Big). \tag{6.9}$$

To summarize, the alignment parameters of the cost models for protein sequence alignment are the 210 substitution costs σ_{ab}, the 6 substitution modifiers μ_{uv}, the 2ℓ gap costs γ_i and λ_i for internal gaps, and the two gap costs γ and λ for external gaps. This is a total of $218 + 2\ell$ parameters for ℓ disruption levels. In general, each model depends on the window width w, whether single or multiple secondary structure prediction is done at residues, the choice of measure Ψ for the degree of secondary structure, and the choice of gap context.

6.3 Optimal alignments under new models

Given an alignment cost function f described in previous section (Equation 6.1)), we can compute an optimal alignment of two protein sequences under f using dynamic programming. We can use Algorithm 2.1 for the *insertion and mixed contexts* because

- all the substitution cost functions s employing a different approach to measuring the degree of secondary structure are an *arbitrary* function in the substitution and modifier parameters, and
- all the gap cost functions g of the insertion and mixed gap context are an *affine* function in the internal gap parameters,

and these function properties were assumed by Algorithm 2.1. For the alignment cost function f for the *deletion context* we need a similar but new algorithm with new recurrences for f.

A simple application of Algorithm 2.1 on our new cost function f for the insertion and mixed gap contexts leads to running time $\mathrm{O}(n^3)$, where n is the length of the sequences. The key to achieving better time bound for these contexts lies in efficiently evaluating the gap cost function g. As explained in the following sections, we achieve $\mathrm{O}(n^2)$ running time for the insertion and mixed gap contexts through *preprocessing* on the input sequences, and $\mathrm{O}(n^2 \lg n)$ for the deletion gap context via a *candidate list approach* with preprocessing.

The idea of the preprocessing is this. All of the gap cost functions g for the deletion, insertion, and mixed gap contexts (Equations (6.6), (6.8), and (6.9) respectively) are of form: $g_\gamma + g_\lambda$, where g_γ, called *gap-open cost function*, involves the gap-open parameter term (the first term) and g_λ, called *gap-extension cost function*, involves the gap-extension parameter terms (all the other terms) in the equations for the contexts. With preprocessing input sequences, these functions g_γ

and g_λ are evaluated in constant time for the insertion and mixed gap context. Note that g_γ and g_λ are the parameters γ and λ in the recurrences (Equation (2.2)) used in Algorithm 2.1.

Before we start describing the preprocessing and the candidate list approach, we give recurrences for external gaps because these recurrences rather complicate the presentation and they are irrelevant to what we describe for the gap contexts. The gaps occurring at the ends of alignments are trivially handled as boundary conditions and special cases in the general recurrences. Let $\widetilde{\gamma}$ and $\widetilde{\lambda}$ be respectively external gap-open and gap-extension parameters. Consider the recurrences in Algorithm 2.1 which aligns two sequences of lengths m and n. For an external gap which occurs at the beginning of an alignment

$$\begin{aligned} V(i,0) &:= \widetilde{\gamma} + i\,\widetilde{\lambda} \quad \text{and} \\ H(0,j) &:= \widetilde{\gamma} + j\,\widetilde{\lambda} \end{aligned}$$

replace the corresponding boundary conditions in the algorithm, and for an external gap which occurs at the end of an alignment, for $1 \leq i \leq m$ and $1 \leq j \leq n$,

$$\begin{aligned} V(i,n) &:= \min\Big\{M(i-1,n)+\widetilde{\gamma}\,,\, V(i-1,n)\Big\} + \widetilde{\lambda} \quad \text{and} \\ H(m,j) &:= \min\Big\{M(m,j-1)+\widetilde{\gamma}\,,\, H(m,j-1)\Big\} + \widetilde{\lambda} \end{aligned}$$

replace the corresponding recurrences in Equation (2.2). From now on, we ignore the external gaps.

6.3.1 Insertion and mixed contexts

Let g_γ and g_λ be the respective gap-open and gap-extension cost functions for the insertion and mixed gap contexts (Equations (6.8) and (6.9)). The recurrences (Equation (2.2)) in Algorithm 2.1 can be used for both contexts if we can evaluate g_γ and g_λ in constant time and use the values of these functions for the γ and λ parameters in the recurrences. By *preprocessing* input sequences, we build look-up

lists for evaluation of g_γ and g_λ which can be referenced in O(1) time with O(1) space, and can find an optimal alignment under our new cost function f in $O(n^2)$ time using Algorithm 2.1.

Given a sequences A of length n, we build a look-up list L_1 for $h(A, i)$ in Equation (6.5) for $1 \leq i \leq n$, and then build a look-up list L_2 for $H(A, k)$ in Equation (6.7) for $1 \leq k \leq n-1$ using list L_1. This preprocessing takes $O(n)$ time with $O(n)$ space. Note that g_γ and g_λ for the insertion context can be evaluated in O(1) time using L_2. For the gap-extension cost g_λ of the mixed gap context we build a prefix sum array. The evaluation of g_λ is then simply the difference of two precomputed prefix sums. Let L_3 be an accumulated array of length $n+1$ whose 0th entry is zero and kth entry is $\sum_{1 \leq i \leq k} \lambda(L_1(i))$. Then $L_3(j) - L_3(i-1) = \sum_{i \leq k \leq j} \lambda(h(A, k))$. This preprocessing is done with $O(n)$ time and space. Note that g_γ and g_λ for the mixed context can be evaluated in O(1) time using L_2 and L_3. Hence the gap cost function g for both gap contexts can be evaluated in O(1) time.

6.3.2 Deletion context

We write the recurrences for the alignment cost function f of the deletion context that scores an alignment of two sequences A and B. Let $M(i, j)$ be the cost of an optimal alignment of prefixes $A[1 : i]$ and $B[i : j]$. This alignment ends with either

- a substitution involving residues $A[i]$ and $B[i]$,
- a gap involving substring $A[k : i]$ for some $k \leq i$, or
- a gap involving substring $B[k : j]$ for some $k \leq j$.

In each of these cases, the alignment is preceded by an optimal solution over shorter prefixes. This leads to the recurrences,

$$M(i,j) \;:=\; \min \begin{cases} M(i-1,j-1) \;+\; s(A,i,B,j), \\ \min\limits_{1\le k\le i}\Big\{M(k-1,j) \;+\; g_\gamma(A,i,k)\Big\} \;+\; g_\lambda(B,j), \\ \min\limits_{1\le k\le j}\Big\{M(i,k-1) \;+\; g_\gamma(B,k,j)\Big\} \;+\; g_\lambda(A,i), \end{cases}$$

where g_γ and g_λ are the respective gap-open and gap-extension cost functions for the deletion context (Equation (6.6)), and s is the substitution cost function (Equations (6.2) or (6.3)) (To simplify the presentation this ignores boundary conditions and the special case of external gap costs.) The difficulty for the deletion context lies in the fact that g_γ in the inner minimizations is a function of k.

A naive approach to the recurrence computation takes $\Theta(n^3)$ time, where n is the length of the input sequences. To compute the recurrences efficiently, we use the *candidate list approach* originally developed for alignment with convex gap costs by Miller and Myers [41] and Galil and Giancarlo [18]. While our gap-open cost function g_γ is not convex in their original sense, their technique still applies. Below we briefly review the ideas behind this technique without proving properties on *candidates*.

The candidate list approach speeds up the evaluation of the two inner minimizations in the above recurrence for $M(i,j)$. The inner minimization involving $g_\gamma(B,k,j)$ can be viewed as computing the function

$$F_i(j) \;:=\; \min_{1\le k\le j}\Big\{G_i(j,k)\Big\},$$

where $G_i(j,k) := M(i,k-1) + g_\gamma(B,k,j)$. Similarly, the minimization involving $g_\gamma(A,i,k)$ can be viewed as computing a function $\overline{F}_j(i)$ that is the minimum of another function $\overline{G}_j(k,j)$. When filling in row i of the dynamic programming table for $M(i,j)$, the candidate list approach maintains a data structure for row i that enables fast computation of $F_i(j)$ across the row for increasing j. Similarly,

it maintains a separate data structure for each column j that enables fast computation of $\overline{F}_j(i)$ down the column. When processing entry (i, j) of the dynamic programming table, the data structures for row i and column j permit evaluation of $F_i(j)$ and $\overline{F}_j(i)$ in $O(\lg n)$ amortized time. Evaluating the recurrence at the $O(n^2)$ entries of the table then takes a total of $O(n^2 \lg n)$ time.

For a *fixed* row i, the candidate list technique computes $F(j)$ as follows. (When i is fixed, we drop the subscript i on F_i and G_i.) Each index k in the minimization of $G(k, j)$ over $1 \leq k \leq j$ is viewed as a *candidate* for determining the value of $F(j)$. A given candidate k contributes the *curve* $G(k, j)$ which is viewed as a function of j for $j \geq k$. Geometrically, the values of $F(j)$ at all $1 \leq j \leq n$ are given by the *lower envelope* [3] of these n candidate curves.

When computing $F(j)$ across the row for each successive j, a representation of this lower envelope is maintained at all $j' \geq j$, only considering curves for candidates k with $k \leq j$. The lower envelope for $j' \geq j$ is represented as a partition of the interval $[j, n]$ into maximal subintervals such that in each subinterval the lower envelope is given by the curve for exactly one candidate.

The partition can be described by a list of the right endpoints of the subintervals, say $p_1 < \ldots < p_t = n$ for t subintervals, together with a list of the corresponding candidates $k_1, \ldots, k_t$ that specify the lower envelope at these subintervals. Note that once the partition is known at column j, the value $F(j)$ is given by the curve for the candidate corresponding to the first subinterval $[j, p_1]$ of the partition, namely $G(k_1, j)$. When advancing to column $j+1$ in the row, the partition is updated by considering the effect of the curve for candidate $j+1$ on the lower envelope.

Given a new candidate j, we compare it on intervals of the current partition $p_1, \ldots, p_t$, starting with the leftmost. In general when comparing against the ith interval $x = (p_{i-1}, p_i]$, we first examine its right endpoint $c = p_i$. If $G(j, c) \leq G(k_i, c)$, then by a property called *crossing once property* (which states for given any interval

and two candidates, either one candidate dominates the other in the whole interval or they split the interval into at most two subintervals where each dominates the other), candidate j dominates across x, so we *delete* interval i from the partition (effectively merging it with the next interval), and continue comparing j against interval $i+1$. If $G(j, c) > G(k_i, c)$, then by a property called *dominance property* (which states for given two candidate, if one candidate starts dominating the other, it dominates later on too), j is dominated on intervals $i + 1, \ldots, t$ by their corresponding candidates, so those intervals do not change in the partition. Interval i may change, though if it does, by the cross once property it at worst splits into two pieces with j dominating in the left piece. In the case of a split, we *insert* a new leftmost interval $[j, p]$ into the partition with j as the corresponding candidate. To find the split point p (if it exists), we can use binary search to identify the rightmost position such that $G(j, p) < G(k, p)$, where k is the candidate corresponding to the current interval. (The dominance property on forward differences implies the difference between the curves for candidates k and j is nonincreasing.) During the binary search, gap-open cost function g_γ is evaluated $\mathrm{O}(\lg n)$ times.

The final issue is the time to evaluate g_γ. (For the evaluation of g_λ, we take the same approach in the previous section.) We build a look-up table for g_γ with $\mathrm{O}(n^2)$ time and space preprocessing. So we can evaluate both of g_γ and g_λ in $\mathrm{O}(1)$ time.

To summarize, given two sequences of lengths n and fixed alignment parameters, after $\mathrm{O}(n^2)$ time and space preprocessing on these input sequences, we can compute the recurrences in $\mathrm{O}(n^2 \lg n)$ time by evaluating the gap cost function g in $\mathrm{O}(1)$ time with the candidate list approach.

Summary of the chapter We presented new alignment cost models based on predicted secondary structure to improve the accuracy of protein sequence alignment (Section 6.2). We explained how prediction on input sequences is leveraged

in these models: the substitution cost reflects the structures of residues being substituted (Section 6.2.2), and the gap cost takes into account the disruption in structures around a gap (Section 6.2.3). We showed how to compute an optimal alignment under these models efficiently (Section 6.3).

CHAPTER 7

EXPERIMENTAL RESULTS

We present experimental results from three different pieces of work on Inverse Alignment [32, 34, 36]. The first set of experiments focuses on how much closer to optimal we could make *complete examples* under the *relative error criterion*, and how small the error measure on a test set is achievable with a choice of parameter values learned on a training set [32]. The second set of experiments compares the *absolute error criterion* to the relative error criterion with *partial examples*, in terms of *recovery error* instead of score measures [34]. This includes an experiment on the effect of parameter values learned from partial examples on the *multiple alignment* on which the examples are induced. The last set of experiments compares different alignment cost models under the *discrepancy error criterion* based on protein *secondary structure prediction* to study the improvement in accuracy resulting from incorporating predicted secondary structure [36].

All of experiments used an implementation of the practical *cutting plane algorithm* for Inverse Alignment presented in Section 4.1.1, with the corresponding *separation algorithm* given in Section 4.1.2. The implementation, called `IPA` [35], is written in `C` language and uses `GLPK` (`GNU` Linear Programming Kit) [40] to solve the linear programs, and handles all the implementation issues mentioned in Section 4.2.

To assess the quality of the learned parameter values, we either used an error measure function E or the recovery error function d on the parameter values, which are learned from examples in a *training set*, and with a *test set* in which these values are applied to the examples. In general, these two sets are different datasets, but when they are the same dataset in an experiment this will be explicitly mentioned.

7.1 Cost suboptimality under relative error criterion

We present results from computational experiments on how much closer to optimal we could make given complete examples by setting the substitution parameters free versus fixed at standard cost matrices such as PAM and BLOSUM under the relative error model. We investigate on how small the relative error on a test set is achievable with a choice of parameter values learned from a training set. For the experiments we used biological multiple alignments in a benchmark suite called PALI with an implementation of our algorithm for Inverse Alignment under Relative Error (Section 4.1.2).

7.1.1 Experimental setup

We describe alignment parameters, examples, and the objective function used in the experiments.

Parameters The standard 212 alignment parameters described in Section 3.1 are used: 210 substitution parameters σ_{ab}, one gap-open parameter γ, and one gap-extension parameter λ. We used the alignment parameters under two different scenarios. In one scenario, all parameters are *free* and all 212 parameter values are learned using the relative error criterion. In the other scenario, all substitution parameters σ_{ab} are *fixed* at various standard substitution matrices such as PAM and BLOSUM, and only the gap parameter values γ and λ are learned.

Examples For the experiments, we used six multiple sequence alignments from the PALI database benchmark suite of structural multiple alignments of proteins [5]. PALI contains 1655 benchmark multiple alignments, accounting for every family from the SCOP [43, 44] classification of proteins, computed by structural alignment without hand curation. Table 7.1 describes the PALI families we chose: T-boxes from p53-like transcription factors (b.2.5.4), NADH FMN oxidoreductase-like pro-

Table 7.1: Dataset characteristics.

SCOP identifiers	PALI number	Sequence length	Percent identity	Gap density	Space density
b.2.5.4	333	183	14.3	4.0	8.5
b.45.1.2	419	151	16.1	6.5	18.4
b.42.4.1	409	172	15.0	10.3	16.2
c.31.1.5	633	197	16.8	5.7	28.7
a.24.1.1	99	143	17.8	3.4	22.5
b.80.1.5	483	299	17.0	3.5	7.7

teins (b.45.1.2), Kunitz STI inhibitors (b.42.4.1), Sir2 transcriptional regulators (c.31.1.5), Apolipoproteins (a.24.1.1), and Pectin methylesterases (b.80.1.5). Each family was reduced to 20 sequences by removing outlier sequences. For each family in the table we give the SCOP identifier, the PALI accession number, the average sequence length, the average percent identity over all induced pairwise alignments, the average gap-open density percentage over all pairwise alignments, and the average gap-extension density percentage over all pairwise alignments. For the training and testing experiments which are described later, these 20 sequences in each family were partitioned into two groups of 10 sequences each, and we took these groups as training and test sets. Given a dataset all pairwise alignments induced on the multiple alignment were taken as *complete examples*.

Objective function For the reduced linear programs we use the objective function $\max\{\sigma-\iota+\gamma-\lambda\}$, where σ and ι are the minimum substitution and maximum identity costs, as described in Section 3.3.4. The reason for choosing this particular objective function is that biologists prefer few gaps of long length to lot of gaps of short length, which leads to $\gamma-\lambda$, and want a big separation between the minimum substitution cost and the maximum identity cost, which gives $\sigma-\iota$.

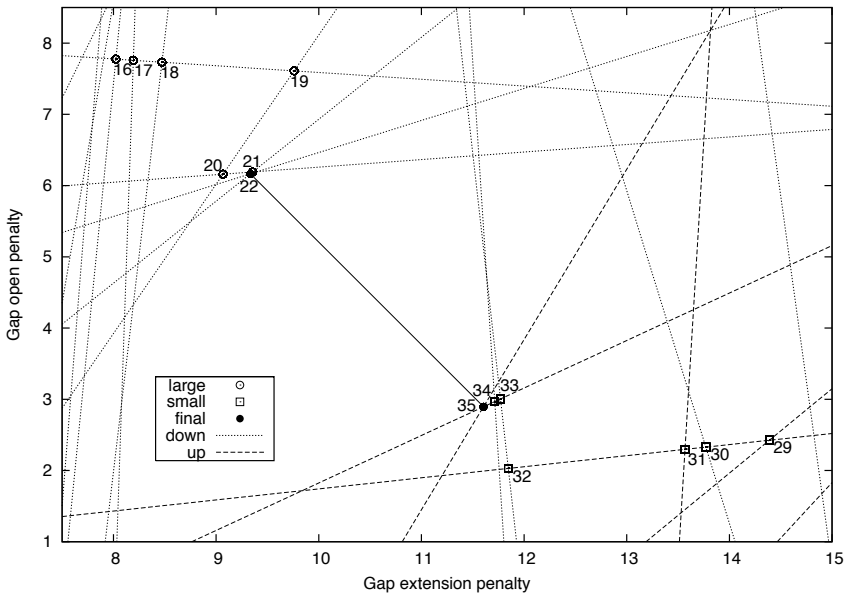

Figure 7.1: Finding gap costs by the cutting plane algorithm.

7.1.2 Effect of parameter blending

Before we describe the experiment on the effect of parameter blending, we illustrate how the cutting plane algorithm works. In Figure 7.1.2, the labeled points are successive solutions for Inverse Alignment under Relative Error using the `PAM250` substitution cost matrix on the dataset with `SCOP` identifier `b.80.1.5` at the smallest relative error $E_{\text{rel}} = 0.06$ (see Tables 7.2 and 7.3). 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.

We now describe the experiment. This experiment investigated how small a maximum relative error E_{rel} is achievable on a test set with a choice of parameter

Table 7.2: Generalizing from training set to test set.

SCOP identifier	Sequence length	Percent identity	Closeness E_{rel}						
			ϵ_{train}	ϵ_{test}	ϵ_{blend}	α_{blend}	ϵ_0	$\epsilon_{1/2}$	ϵ_1
b.2.5.4	183	14.3	1.7	**1.2**	1.5	0.47	1.7	**1.5**	2.6
b.45.1.2	151	16.1	2.8	**2.8**	3.1	0.37	3.7	**3.1**	4.7
b.42.4.1	172	15.0	3.9	**3.7**	4.2	0.49	4.4	**4.2**	4.8
c.31.1.5	197	16.8	3.1	**2.2**	3.0	0.55	4.3	**3.0**	4.4
a.24.1.1	143	17.8	1.9	**1.7**	3.2	0.22	3.6	**3.4**	4.6
b.80.1.5	299	17.0	1.2	**2.0**	3.1	0.65	4.4	**3.1**	4.4

values learned on a training set. All alignment parameters are free in this experiment. Two choices of parameter values are learned from complete examples in a training set for which the maximum relative error on the training set is minimized. The convex combination of these two parameter choices that gives the smallest value of the error measure on a test set is sought.

For each dataset with all parameters free, we found the smallest error E_{rel} on the training set along with a choice of parameter values. We call this smallest error value ϵ_{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 parameter values at ϵ_{train}, x_{large} and x_{small}, which are the parameter choices that respectively *maximized* and *minimized* the linear program objective $\{\sigma - \iota + \gamma - \lambda\}$.

We searched for the convex combination between these two extremes x_{large} and x_{small} that yielded a parameter choice for the test set with the smallest maximum relative error. For a given $0 \leq \alpha \leq 1$, the convex combination of these extremes is $\alpha\, x_{\text{large}} + (1-\alpha)\, x_{\text{small}}$. The α that gave the smallest error on the test set is called α_{blend}, and its corresponding error value is called ϵ_{blend}.

These values are shown for the six datasets in Table 7.2. For each multiple sequence alignment dataset, best possible parameters computed on a training subset of 10 members are applied to a disjoint test subset of 10 members. For each dataset the table lists the SCOP identifier, the average sequence length, and the

Table 7.3: Closeness to optimality for fixed and free substitution parameters.

SCOP identifier	Closeness E_{rel} (%)				
	free	fixed			
		PAM250	PAM120	BLOSUM45	BLOSUM80
b.2.5.4	**2**	**5**	9	8	9
b.45.1.2	**4**	**9**	15	12	18
b.42.4.1	**5**	**8**	11	9	12
c.31.1.5	**4**	**11**	16	12	16
a.24.1.1	**3**	**21**	45	34	58
b.80.1.5	**3**	**6**	9	8	11

average percent identity over all induced pairwise alignments. The meaning of the closeness entries is described above. The table also gives the values ϵ_α of maximum relative errors 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 $\alpha = \frac{1}{2}$ always yields a smaller error than the extremes $\alpha = 0, 1$, indicating that central parameters generalize better.

7.1.3 Results for free versus fixed substitution parameters

This experiment investigated how much closer to optimal one could make the examples in a dataset by freeing the substitution parameters versus fixing them at a standard substitution matrix.

For all 20 sequences of each of the PALI datasets, we computed the smallest relative error E_{rel} on the set of all 190 induced pairwise alignments as examples, with substitution parameters free and fixed at PAM [10] and BLOSUM [27] shown in Table 7.3. For each PALI dataset and for fixed or free substitution parameters, the table lists the smallest relative error E_{rel} as a percentage. As might be expected, the minimum error under free substitution parameters is smaller than the minimum error under fixed substitution parameters. Notice that on these datasets PAM250

Table 7.4: Running time and number of violated inequalities.

SCOP identifier	Time (sec)				Violated inequalities					
	fixed		varying		fixed			varying		
	med	max	med	max	min	med	max	min	med	max
b.2.5.4	24	25	5	123	4	11	23	5	32	972
b.45.1.2	17	18	7	46	3	12	21	5	118	590
b.42.4.1	23	30	11	83	2	13	22	7	176	681
c.31.1.5	29	31	13	96	2	11	20	5	196	832
a.24.1.1	16	16	14	264	6	17	22	13	236	1398
b.80.1.5	63	65	23	226	2	10	20	3	238	1087

gave the smallest error among the four matrices considered. On the other hand, the optimal substitution matrix found when substitution parameters were free has roughly at most half the error for PAM250, and is sometimes much better. Notice also that with substitution parameters free, one can consistently come very close to optimal on every dataset.

Finally, Table 7.4 gives running times and the number of violated inequalities for the cutting plane algorithm on the experiments of Table 7.3. The running times are wall-clock times in seconds and the columns report the median and extreme values across the iterations of the binary search for the smallest relative error E_{rel}. 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 parameters, and under roughly 4 minutes for free substitution parameters while finding values for 212 parameters. Experiments were on a 3 GHz Pentium 4 with 1 GB of RAM.

7.2 Recovery rates under absolute and relative error criteria

We present results from computational experiments on comparing the absolute error criterion to the relative error criterion. We investigated whether parameters

learned from pairwise alignment examples can improve the recovery of benchmarks when used for multiple sequence alignment. We also examined how the initial completion and the nondegeneracy threshold affect the final recovery. For the experiments we used biological multiple alignments in the benchmark suite PALI with an implementation of our algorithms for Inverse Alignment under Absolute and Relative Errors described in Section 4.1.2.

7.2.1 Experimental setup

We describe alignment parameters and examples used in the experiments.

Parameters The standard 212 parameters described in Section 3.1 are used: 210 substitution parameters σ_{ab}, one gap-open parameter γ, and one gap-extension parameter λ. We learned the alignment parameters under two different scenarios described in Section 7.1.1: all parameters are *free*, and substitution parameters are *fixed* while gap parameters are free.

Examples To obtain the examples for our experiments, we used benchmarks from the PALI [5, 20] suite of structural multiple alignments of proteins, which is described in Section 7.1.1. In total, PALI has 1655 benchmark alignments, from which we selected a subset U consisting of *all* PALI alignments on at least 7 sequences with nontrivial gap structure. (If an alignment had essentially no gaps other than at the terminal ends of its sequences, it was deemed to have *trivial* gap structure, and was not included in set U.) Set U was further partitioned into two equal-sized subsets P and Q for the purpose of conducting cross validation experiments on training set and test set. We also performed a detailed study of a smaller subset $S \subset U$ containing the 25 benchmarks from U with the most sequences. For each benchmark in the smaller set S, where the benchmark is a multiple alignment on n sequences, we took for the examples the $\binom{n}{2}$ pairwise alignments induced on all

Table 7.5: Dataset characteristics.

Dataset	Number of benchmarks	Number of sequences	Sequence length	Percent identity	Core blocks (%)	
					coverage	identity
U	102	14	239	29.1	40.4	35.8
P	51	15	239	29.7	41.0	37.1
Q	51	12	239	28.1	39.9	33.8
S	25	20	245	27.4	33.2	33.5

pairs of rows of the multiple alignment. For the benchmarks in the larger sets U, P, and Q, we took as the examples a much sparser set of induced pairwise alignments, as described later.

Table 7.5 summarizes the characteristics of these datasets. For each dataset, the table lists the name of the set, the number of benchmark multiple alignments in the set, the average number of sequences in each multiple alignment, the average length of sequences in the set, the percent identity of over all induced pairwise alignments, the average percentage of core block columns in induced pairwise alignment, and the average percent identity over all core blocks. Each of these datasets consists of *partial examples*. The reliable regions in the partial examples are given by the *core blocks* of the benchmark. (`PALI` explicitly identifies the core blocks in each benchmark, which consist of those columns of the multiple alignment in which all pairs of residues are within 3 Å on their C^{α} atoms in the structural multiple alignment.) For sets U, P, Q, and S, the table reports the number of benchmarks in each set and, averaged across the benchmarks in the set, their number of sequences, their sequence length, and the percent identity of their induced pairwise alignments. Also shown averaged over the core blocks of the benchmarks are their percent coverage of the sequences and their percent identity.

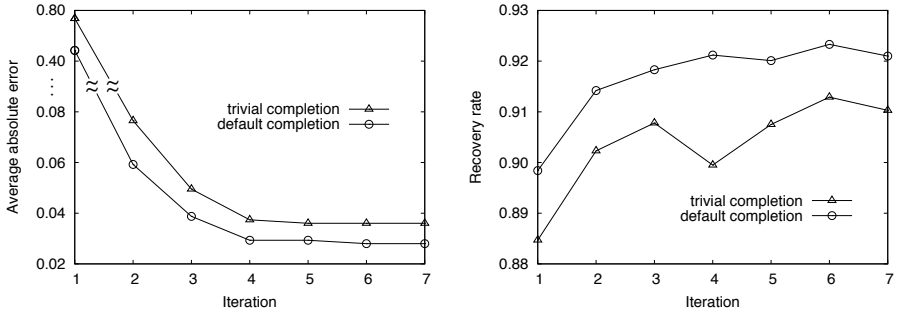

Figure 7.2: Improvement in error and recovery for the iterative approach.

7.2.2 Correlation of recovery rate and cost error

This section explains the correlation of recovery rate and cost error, and the robustness of recovery rate to nondegeneracy threshold value.

Effect of iterations

Figure 7.2 illustrates the improvement in error for the iterative approach to partial examples described in Section 5.2. The plots show the average absolute error and the recovery rate across the iterations, starting from a given initial completion. The *recovery rate* is the percentage of columns from reliable regions that are present in an optimal alignment computed using the estimated parameters. Results are plotted for two initial completions: the *default completion*, which optimally aligns the unreliable regions using default parameters, and the *trivial completion*, which uses all columns of the partial alignment including unreliable ones. (The default parameters are those used by the multiple alignment tool `Opal` [60, 61], which are the `BLOSUM` substitution matrix with carefully chosen gap costs.) The examples for the plot are all induced pairwise alignments of `PALI` benchmark `b.1.8.1`, and the error criterion is average absolute error. As the figure shows, the error in alignment scores decreases across the iterations, and the corresponding recovery of the example alignments tends to improve as well. (Also note that starting from the

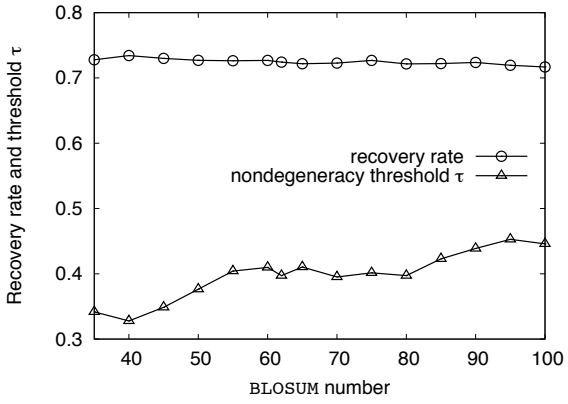

Figure 7.3: Robustness of recovery rate to nondegeneracy threshold τ.

default completion has better recovery and less error than starting from the trivial completion.) Generally, smaller error correlates with higher recovery.

Robustness with respect to nondegeneracy threshold

Figure 7.3 shows that across a very wide range of conventional substitution matrices, using the corresponding value of threshold τ in nondegeneracy inequality (defined in Inequality (4.6)) does not adversely affect recovery. In other words, the choice of which substitution matrix is used to establish τ is not critical. (Nevertheless, enforcing the nondegeneracy inequality *is* crucial: without it, the inverse alignment algorithm immediately outputs an optimal *degenerate* solution.) In the plot, the value i along the horizontal axis specifies the index of a `BLOSUM` substitution matrix. The `BLOSUM` matrices we consider have been transformed into cost matrices with extreme values 0 and 1. The bottom curve plots the value of τ measured on matrix `BLOSUM`i. (Note that `BLOSUM`i matrices that correspond to a higher percent identity generally have greater discrimination between substitutions and identities, as evidenced by the bottom curve for τ tending to increase in i.) The top curve plots recovery rates when the corresponding value of τ is used in the nondegeneracy

inequality. At every point on this curve, parameters are learned using the same collection of pairwise alignment examples (set $\widetilde{U}$, which is described later), and the average recovery rate on these examples is plotted. Recovery is robust with respect to the particular matrix used to establish τ[1]. Across the range of `BLOSUM` matrices, τ varies up to 40%, while there is little variation in recovery. In short, recovery is robust to the particular value τ used in the nondegeneracy inequality.

7.2.3 Results on variations of Inverse Alignment

Table 7.6 shows a detailed comparison of recovery rates across several different scenarios for Inverse Near-Optimal Alignment. The rows of the table correspond to the `PALI` benchmarks in set S, which are named by their `SCOP` identifier. (The rows are sorted in increasing order on column "default (c)," which is described in more detail later.) For each benchmark, the table reports the average recovery rate across its examples, which consist of all induced pairwise alignments of the benchmark, for various scenarios. In these scenarios, parameters learned from the examples for a given benchmark are applied to the sequences in this benchmark, either to compute an optimal global *pairwise alignment* of the example sequences (corresponding to the first group of columns), or to compute a global *multiple alignment* of the benchmark sequences using the tool `Opal` [60, 61] (corresponding to the second group of columns). `Opal` [60, 61] is a multiple sequence alignment tool, based on optimally aligning alignments [33, 54], that seeks to optimize the sum-of-pairs objective [7]. In the first group of five columns, the table reports the recovery rates on the examples for optimal pairwise alignments computed using

(1) the *default* parameters in `Opal`, namely the `BLOSUM` matrix [27] with carefully chosen affine gap costs [60],

[1]The plot gives the recovery rate when computing pairwise alignments of the example sequences. Later, Table 7.7 gives the recovery rate when computing multiple alignments of benchmark sequences (substantially higher).

Table 7.6: Recovery rates on dataset S for variations of Inverse Alignment.

SCOP identifier	Pairwise alignment					Multiple alignment		
	default (a)	BLOSUM	relative	absolute (b)	difference ($b-a$)	default (c)	absolute (d)	difference ($d-c$)
c.95.1.1	38.5	39.0	34.3	47.8	**9.3**	45.2	70.1	**24.9** ←
d.32.1.3	46.2	45.4	38.9	56.6	**10.4**	64.3	84.7	**20.4**
c.95.1.2	44.1	46.8	32.9	69.4	**25.3**	64.9	82.9	**18.0**
e.3.1.1	46.6	46.2	33.1	50.6	**4.0**	66.6	77.4	**10.8**
d.185.1.1	46.3	47.8	37.4	60.5	**14.2**	68.0	83.1	**15.1**
b.29.1.11	56.5	56.5	64.5	73.4	**16.9**	69.2	90.1	**20.9**
d.54.1.1	51.3	51.9	54.3	67.4	**16.1**	70.4	91.7	**21.3**
c.56.2.1	59.2	61.6	61.5	84.8	**25.6** ←	78.2	90.9	**12.7**
c.1.8.6	64.8	65.8	61.0	72.3	**7.5**	80.6	93.2	**12.6**
b.43.3.1	57.7	58.6	57.2	76.0	**18.3**	82.2	93.4	**11.2**
a.127.1.1	68.9	71.1	47.5	73.1	**4.2**	82.9	89.8	**6.9**
c.67.1.1	60.7	62.2	46.1	66.6	**5.9**	83.6	88.5	**4.9**
d.81.1.1	67.2	69.2	67.8	85.6	**18.4**	85.2	98.4	**13.2**
b.22.1.1	66.6	67.9	75.1	85.0	**18.4**	86.1	91.8	**5.7**
c.1.11.2	68.4	71.4	64.8	81.2	**12.8**	88.4	95.7	**7.3**
b.1.8.1	78.3	79.2	85.9	91.0	**12.7**	89.6	98.6	**9.0**
a.104.1.1	78.8	80.7	69.0	80.0	**1.2**	89.6	90.7	**1.1**
b.43.4.2	59.9	61.4	65.1	82.0	**22.1**	91.6	92.7	**1.1**
a.93.1.1	76.8	81.1	81.0	88.2	**11.4**	94.2	95.2	**1.0**
c.1.7.1	88.9	89.6	88.9	97.1	**8.2**	94.6	99.5	**4.9**
c.1.1.1	91.0	91.3	93.1	97.4	**6.4**	94.8	96.1	**1.3**
b.50.1.2	86.5	87.0	85.3	88.5	**2.0**	94.9	96.5	**1.6**
d.165.1.1	90.2	91.3	92.6	96.8	**6.6**	95.8	98.9	**3.1**
d.162.1.1	86.2	88.4	90.0	96.3	**10.1**	98.0	100.0	**2.0**
e.1.1.1	94.5	96.0	95.3	96.4	**1.9**	98.9	99.1	**0.2**
average	65.7	68.3	64.9	78.6	**12.9**	82.3	91.5	**9.2**

(2) the BLOSUM matrix with affine gap costs learned using the absolute error criterion,

(3) the substitution matrix and gap costs learned using the *relative* error criterion,

(4) the substitution matrix and gap costs learned using *absolute* error criterion, and

(5) the *difference* between the recovery rates of the absolute error criterion and default parameters.

In the second group of columns, the table reports the recovery rates of a multiple sequence alignment of the benchmark sequences computed by Opal using

(6) the *default* parameters of Opal,

(7) the substitution matrix and costs learned under *absolute* error criterion, and

(8) the *difference* between these two.

In these experiments, the precision of the binary search in the relative error criterion is 0.01%. Parameters for alignment are rounded to integers using a scale of 100.

A key conclusion from examining Table 7.6 is that the *absolute error criterion* substantially outperforms the relative error criterion with respect to recovery of example alignments (seen by comparing the table columns labeled absolute and relative in the pairwise alignment group). In addition, from inspecting the two difference columns in the table (whose maximum entries are marked by a left arrow), parameters learned under absolute error criterion when used for both global pairwise and multiple alignment outperform the default parameters of Opal, which uses the standard BLOSUM matrix, by as much as 25% in recovery. (This particular matrix had the best recovery among the BLOSUM family.) Also note that the recovery rates of parameters when used for multiple alignment are generally much higher than when used for pairwise alignment. To summarize: by performing inverse alignment from partial examples one can learn parameters for global pairwise or multiple

Table 7.7: Recovery rates for cross validation experiments.

Characteristics (training set)			Recovery (test set)		
dataset	examples	identity	U	P	Q
$\widetilde{U}$	204	33.1	83.4	84.8	82.0
$\widetilde{P}$	153	34.5	82.9	86.1	82.2
$\widetilde{Q}$	153	31.9	82.8	84.4	81.2

sequence alignment that are tailored to a given protein family and that yield very high recovery.

7.2.4 Results on parameter generalization

Table 7.7 presents recovery results from cross validation experiments. Parameters learned on sparse training sets using the absolute error criterion are applied to complete test sets. The recovery rate for these parameters is measured on multiple alignments of the benchmarks computed by `Opal`. In the table, parameters learned on training sets $\widetilde{U}$, $\widetilde{P}$, and $\widetilde{Q}$ using the absolute error criterion are applied to test sets U, P, and Q. For a generic set X of `PALI` benchmarks, the examples in training set $\widetilde{X}$ are a subset of the induced pairwise alignments of the benchmarks in X. Set $\widetilde{X}$ contains pairwise alignments selected according to their recovery rate in an initial multiple alignment of the benchmark computed by `Opal` using its default parameters. For $\widetilde{P}$ and $\widetilde{Q}$, the subset contains the induced pairwise alignments from each benchmark whose initial recovery rates rank at the 25th, 50th, and 75th percentiles; for $\widetilde{U}$, the examples rank at the 33rd and 66th percentiles. Parameters from a given training set were used in `Opal` to compute multiple alignments of the benchmarks in the test set, and the table reports the average recovery.

Note there is only a small difference in recovery when parameters are applied for multiple sequence alignment to disjoint test sets, compared to their recovery on their training set. This suggests that the absolute error method is not overfitting the parameters to the training data.

To give a sense of running time, performing inverse alignment on training set $\widetilde{U}$ (of more than 200 examples) involved around 6 iterations for completing partial examples and took about 4 hours on a 3.2 GHz Pentium 4 with 1 GB of RAM, using `GLPK` [40] to solve the linear programs. An iteration took roughly 40 minutes and required around 4,000 cutting planes.

7.3 Improving recovery through secondary-structure-based models

We present results from computational experiments comparing different substitution modifier and gap context models based on predicted protein secondary structure under Inverse Alignment under Discrepancy Error. In these experiments we investigated whether parameters learned from a training set can improve the recovery of a disjoint test set by differing underlying models based on the protein secondary structure prediction. We then examined how the size of a training set affects the generalization of recovery on a test set. For the experiments we used biological multiple alignments from the benchmark suites `BAliBASE`, `HOMSTRAD`, `PALI`, and `SABmark` with an implementation of our algorithm for Inverse Alignment under Discrepancy Error described in Section 4.1.2.

7.3.1 Experimental setup

To measure the improvement in recovery of the alignment cost models based on the secondary structure prediction, we compared these models to a base model, which does not incorporate any prediction of the protein secondary structure. We review the alignment parameters for the base model and the secondary structure based cost models. We explain how the datasets were obtained for the experiments.

Parameters The standard 212 parameters (described in Section 3.1) plus 2 external gap parameters (described in Section 6.2.1) are used in the *base model*: 210

substitution parameters σ_{ab}, two gap parameters γ and λ for internal gaps, and two gap parameters γ_e and λ_e for external gaps.

The *substitution modifier model* has 6 additional modifier parameters μ_{uv} (described in Section 6.2.1) charged for a substitution involving an unordered pair of secondary structural types u and v to the base model. The *gap context model* with ℓ discrete levels of structure disruption has 2ℓ, instead of two, gap parameters for internal gaps (described in Section 6.2.3): ℓ gap-open parameters $\gamma_1, \ldots, \gamma_\ell$ and ℓ gap-extension parameters $\lambda_1, \ldots, \lambda_\ell$.

Examples We first give a general description of the benchmark suites from which alignments were chosen and then explain how examples are selected from these alignments. We explain how the prediction of secondary structure for the sequences in each example were obtained.

For the experiments we chose alignments from four suites of protein alignment benchmarks: `BAliBASE` [4], `HOMSTRAD` [42], `PALI` [5], and `SABmark` [57]. `BAliBASE` contains 218 benchmark multiple alignments based on structural alignments with hand-curated gaps. `HOMSTRAD` contains 590 benchmarks based on structural alignment of homologous protein family with hand-curated gaps. Each benchmark is a pairwise alignment of sequences representing the protein family. `PALI` contains 1655 multiple alignments of all families from the `SCOP` [43, 44] classification and is explained in Section 7.1.1. `SABmark` contains 627 benchmarks with at most 25 sequences that cover the space of folds in the `SCOP` classification of proteins. Each benchmark is a set of pairwise structural alignments that are not necessarily consistent with a multiple alignment.

From each suite of `BAliBASE`, `HOMSTRAD`, `PALI`, and `SABmark`, we chose 100 protein families in a greedy manner such that these families cover as many classes, folds, superfamilies, and families in the `SCOP` classification hierarchy as possible in that order. Then we scored all pairwise alignments induced on a multiple alignment

in each set of 100 families from BAliBASE and PALI suites using the default parameters of Opal multiple aligner [60, 61] to find a median. For HOMSTRAD and SABmark, we scored all pairwise alignments in a family to find a median. This gave us 100 pairwise alignments with SCOP identifiers from each benchmark suite. We took each set of 100 pairwise alignments from BAliBASE, HOMSTRAD, PALI, and SABmark suites as a *dataset* for our experiments, and named them B100, H100, P100, and S100 respectively. BAliBASE and PALI benchmark suites specify the *core blocks* in each multiple alignment in them, while HOMSTRAD and SABmark do not. Hence for each alignment in datasets B100 and P100, we took the core blocks specified in the alignment as the reliable regions to obtain a *partial example*. For each alignment in datasets H100 and S100, we took all maximal runs of substitution columns in the pairwise alignment as the reliable regions.

To obtain the secondary structure prediction of every protein sequence in each dataset, we used a predictor called PSIPRED [28], which uses a two-stage neural network to predict protein secondary structure based on the position-specific scoring matrices for the sequence generated by PSI-BLAST [2]. For each protein sequence, both of single and multiple prediction (described in Section 6.2.2) are obtained using PSIPRED. In case of multiple prediction, the confidence output for each residue in the sequence was normalized to form a distribution. In summary, each of datasets B100, H100, P100, and S100 consists of 100 partial examples, each of which aligns two protein sequences and is associated with the single and multiple prediction of each sequence in the example.

For the experiment on how the size of a training set affects the generalization of recovery on a full-sized test set, we chose 50 members from B100 in the exact same greedy manner described above to obtain a smaller dataset B50, and then chose 25 members from B50 to acquire an even smaller dataset B25. We repeated this procedure with datasets H100, P100, and S100. So our datasets satisfy a property $x\mathtt{25} \subset x\mathtt{50} \subset x\mathtt{100}$ for $x \in \{\mathtt{B}, \mathtt{H}, \mathtt{P}, \mathtt{S}\}$.

Table 7.8: Comparison of recovery for substitution modifier models.

Training set	Test set	Structure prediction (%)		
		none	*single*	*multiple*
$\overline{\texttt{B100}}$	B100	73.52	80.99	81.57
$\overline{\texttt{H100}}$	H100	79.92	82.44	83.58
$\overline{\texttt{P100}}$	P100	67.87	74.67	76.60
$\overline{\texttt{S100}}$	S100	68.76	72.58	72.55

Finally, we took the union of all datasets with the same number of examples, say y, and obtained dataset $\texttt{U}y$ for $y \in \{\texttt{25}, \texttt{50}, \texttt{100}\}$. Notice that the number of examples in $\texttt{U}y$ is $4y$. In the following $\overline{xy}$ denotes all the examples in set $\texttt{U}y - xy$, where $x \in \{\texttt{B}, \texttt{H}, \texttt{P}, \texttt{S}\}$ and $y \in \{\texttt{25}, \texttt{50}, \texttt{100}\}$.

7.3.2 Comparison of secondary structure based models

This experiment investigated how much improvement in recovery can be achieved under different secondary structure based models compared to the base model.

Substitution modifier models

Table 7.8 shows a comparison of recovery rates for the base model and substitution modifier models based on single and multiple prediction (described in Section 6.2.2) under discrepancy error criterion (described in Section 2.2.3). In the table each training set $\overline{y100}$ for $y \in \{\texttt{B}, \texttt{H}, \texttt{P}, \texttt{S}\}$ is a complement of a training set $y\texttt{100}$ in the universe set U100, and the training set and the corresponding test set are disjoint; the parameter values learned from partial examples in each training set are not trained for the corresponding test set, and hence the table shows the cross validation of learned parameter values.

Note that both of substitution modifier models always outperform the base model and the substitution modifier model with multiple prediction gives improvement in recovery over the model with single prediction in general. The best im-

Table 7.9: Comparison of recovery for all gap context models.

Degree measure	Gap context (%)		
	deletion	insertion	mixed
binary-single	75.63	75.59	76.51
binary-multiple	75.86	75.90	76.43
continuous-multiple	75.96	76.22	76.63

provement from the base model is 8.7% obtained from `B100` and `P100` under the *substitution modifier model with multiple prediction* on the secondary structure.

Gap context models

Table 7.9 shows a comparison of recovery rates from all the gap context models combined with all the possible degree measure of disruption given in Section 6.2.3 under discrepancy error criterion. In the table, we used `U100`, which is the union of `B100`, `H100`, `P100`, and `S100`, as both a training set and a test set, and 6 and 7 for the size of windows for every gap context and the number of discrete levels of disruption respectively. The number of internal gap parameters for each gap context model is 14 whereas that for the base model is 2.

Note that mixed context is better than the other contexts under every degree measure category. The improvement in recovery from the base model is around 1.8% for the *mixed context with continuous-multiple approach* to measuring the degree of disruption.

Combined models of substitution modifiers and gap contexts

Table 7.10 shows a comparison of recovery rates from the base model, the substitution modifier model with multiple prediction (which is the best among substitution modifier models), and the mixed context model with continuous-multiple approach measuring gap disruption (which is the best among all gap contexts). In the table each training set $\overline{y100}$ for $y \in \{\mathtt{B}, \mathtt{H}, \mathtt{P}, \mathtt{S}\}$ is a complement of a training set $y\mathtt{100}$ in

Table 7.10: Comparison of substitution and gap models.

Training set	Test set	Model (%)		
		no modifiers, no context	with modifiers, no context	with modifiers, mixed context
$\overline{\texttt{B100}}$	B100	73.51	81.57	82.88
$\overline{\texttt{H100}}$	H100	79.92	83.58	84.62
$\overline{\texttt{P100}}$	P100	67.87	76.60	77.32
$\overline{\texttt{S100}}$	S100	68.76	72.55	75.52

the universe set U100 like Table 7.8 and hence the table shows the cross validation of learned parameter values. For mixed context model, we used 6 and 7 for the size of windows for every gap context and the number of discrete levels of disruption respectively.

Note that for a given pair of training and test sets, the most accuracy improvement comes from the substitution modifiers, which accounts for the average 6.1% point increment in recovery across all test sets. The final accuracy improvement by the model in the last column is 7.6% point average across all test sets and the maximum 9.5% point for P100, which are obtained from PALI. A conclusion of this series of experiments is that the best model based on the protein secondary structure prediction among those we considered is the one featuring *modifiers with multiple prediction* and *mixed context with continuous-multiple measure* for disruption, which gives a respectable improvement on alignment accuracy.

Finally, Figure 7.4 shows the relative accuracy improvement of two secondary structure based cost models considered in Table 7.10 over the base model. For the plot, we learned parameter values from the examples in U100 for each model. We then sorted the examples with their respective percent identity in a nondecreasing order. For a given number x, we considered the first x examples only in the sorted list. The horizontal axis is the number of examples considered and the vertical axis is the average percent identity and relative improvement in recovery over the base model as fractions. The recovery of the base model is represented as a horizontal

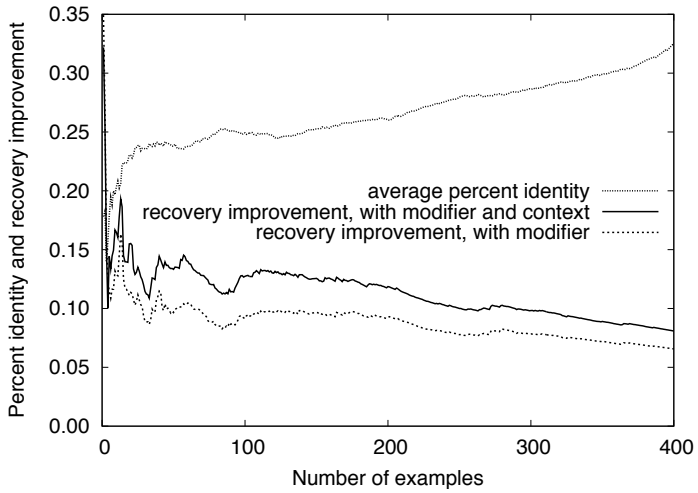

Figure 7.4: Accuracy improvement of secondary-structure-based models.

axis and that of the other two models are two bottom curves in the plot. As more examples are considered, the accumulated average percent identity tends to increase (the top curve) and the accumulated relative improvement in recovery (the bottom curves) tends to decrease. Our secondary structure based models performs very well on examples with low percent identity and the model featuring modifiers with multiple prediction and mixed context with continuous-multiple measure improves the accuracy of the base model as much as 15% for examples with less than 25% identity.

7.3.3 Effect of the size of training set on recovery

Table 7.11 shows the experiments on how the size of a training set affects the generalization of recovery on a test set under the best model obtained from Table 7.10, which is the model featuring modifiers with multiple prediction and mixed context with continuous-multiple measure for gap disruption. For these experiments, we varied the size of a training set by adding more examples to the training set while a test set is fixed. For each entry in the table showing a recovery rate under the size

Table 7.11: Effect of number of examples on recovery.

Training set	Test set	Size x		
		25	50	100
$\overline{\texttt{B}x}$	B100	79.36	80.67	82.88
$\overline{\texttt{H}x}$	H100	81.83	83.48	84.62
$\overline{\texttt{P}x}$	P100	75.26	75.10	77.32
$\overline{\texttt{S}x}$	S100	72.81	74.65	75.52

Table 7.12: Running time and number of cutting planes.

Training set	Test set	Size x					
		25		50		100	
		time	planes	time	planes	time	planes
$\overline{\texttt{B}x}$	B100	39:09	15,448	40:37	14,234	45:03	13,838
$\overline{\texttt{H}x}$	H100	46:47	17,841	37:29	14,864	49:58	14,789
$\overline{\texttt{P}x}$	P100	54:55	18,619	52:01	16,008	38:22	14,440
$\overline{\texttt{S}x}$	S100	54:50	17,831	48:44	15,313	52:53	15,356

column $x \in \{\texttt{25}, \texttt{50}, \texttt{100}\}$, the parameter values learned from a training set $\overline{yx}$ were applied to a test set $y\texttt{100}$, where $y \in \{\texttt{B}, \texttt{H}, \texttt{P}, \texttt{S}\}$. Therefore for a fixed row, the number of examples increases across the columns, and a larger training set includes all the examples in a smaller training set.

Note that as the size of a training set becomes larger, the recovery on a fixed test set becomes better. Recall that as the size of a training set increases, it covers more families in the SCOP classification hierarchy. A key conclusion of these experiments is that a training set with a *larger coverage* on the protein classification yields a *better generalization* of recovery on a test set even though the test set is never seen. Also note that this implies a training set with a larger coverage results in a better recovery on *cross validation* of parameter values.

Finally, Table 7.12 gives running times and the numbers of cutting planes for six iterations of the iterative scheme defined in Algorithm 5.1 on the experiments of Table 7.11. Running times are wall-clock times to solve all six linear programs,

each of which were generated in the reduction of Inverse Alignment under Discrepancy Error. In the worst case, solving an instance of discrepancy error criterion finishes in less an hour. Recall that under the discrepancy error model, the cutting plane algorithm completes partial examples in a training set under a fixed feasible solution to the linear program as well as solving the linear program. Hence each time in the table is the total time spent by both the separation and the completion algorithms. The running times were measured on a 3.2 GHz Pentium 4 with 2 GB of RAM, using `GLPK` [40] to solve the linear programs.

Summary of the chapter We gave experimental results from three different pieces of work on Inverse Alignment. The first set of experiments (Section 7.1) demonstrated that under the relative error criterion, parameter values learned for all *free* alignment parameters yield a better relative error in cost. It also demonstrated that the best convex combination of two extreme parameter choices is the *central choice* for our training sets. The second set of experiments (Section 7.2) demonstrated that the *absolute error criterion* outperforms the relative error criterion, and is *not overfitting* the parameters to our training sets. The last set of experiments (Section 7.3) demonstrated that among all the models incorporating the protein secondary structure prediction, the best is the model featuring substitution modifier with multiple prediction and mixed context with continuous-multiple approach, and a training set with a *larger coverage* on the `SCOP` protein classification hierarchy yields a *better generalization* of recovery.

CHAPTER 8

CONCLUSION

We have presented a new approach to inverse parametric sequence alignment. Our approach is 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 cost model for protein alignment from hundreds of pairwise alignments in a few minutes of computation under the relative error criterion.

We have explored a new approach to inverse parametric sequence alignment that for the first time carefully treats *partial examples*. This new approach minimizes the error of alignment costs under the given criterion and iterates over completions of partial examples. We also studied for the first time the performance of parameter values learned from partial examples when used for multiple sequence alignment. A key conclusion from these experiments is that the *absolute error criterion* substantially outperforms the relative error criterion, in terms of their recovery rates on benchmarks alignments. Furthermore, our results indicate that a substantial improvement in alignment accuracy can be achieved on individual protein families, and that parameters learned across a sampling of protein families generalize well to other families.

We have extended the standard alignment cost model to incorporate predicted *secondary structure* on protein sequences to improve the accuracy of protein sequence alignment. We extensively investigated for the first time the effect of differ-

ent structure disruption measures and gap contexts on the trained parameter values. A key conclusion from these experiments is that the model featuring modifier parameters with multiple prediction and the mixed context with continuous-multiple gap disruption measure gives the most improvement in accuracy over the standard model. Furthermore, the results indicate that a training set with a larger coverage on the protein families yields a better generalization in terms of recovery rates on test sets.

Some topics for future research include the following.

- Can parameter values be learned from both *positive and negative examples*? In other words, is it possible to learn parameter values that make the positive examples score as close to optimal as possible while the negative examples score as far from optimal as possible at the same time?

- Can cutting planes be efficiently found for inverse alignment that are *facet-defining inequalities*? In other words, when the separation algorithm returns a violated inequality, is it possible to find a violated inequality that defines a face of the polytope at the vertex that is an optimal solution of inverse alignment, instead of simply being a most violated inequality?

- Can the best *convex combination* of the normalized cost error and recovery error under discrepancy error criterion be learned? In other words, after scaling the absolute error term in the discrepancy error measure into approximately $[0, 1]$, we have both cost and recovery errors in the same scale. Then what convex combination of these errors yields the parameter values that generalize well in terms of recovery rates?

- Is parameter generalization benefitted by including a *regularization term* in the objective function, for instance by penalizing parameter overfitting through the L_1 norm to retain a linear programming formulation?

Clearly many lines of investigation remain open.

REFERENCES

[1] S.F. Altschul and B.W. Ericson, "Optimal sequence alignment using affine gap costs," *Bulletin of Mathematical Biology* 48, pp. 603–616, 1986.

[2] S.F. Altschul, T.L. Madden, A.A. Schäffer, J. Zhang, Z .Zhang, W. Miller, and D.J. Lipman, "Gapped `BLAST` and `PSI-BLAST`: a new generation of protein database search programs," *Nucleic Acids Research* 25:17, pp. 3389–3402, 1997.

[3] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf,*Computational Geometry: Algorithms and Applications*, Springer-Verlag, Berlin, 2000.

[4] A. Bahr, J.D. Thompson, J.C. Thierry, and O. Poch, "`BAliBASE` (Benchmark Alignment dataBASE): enhancements for repeats, transmembrane sequences and circular permutations," *Nucleic Acids Research* 29:1, pp. 323–326, 2001.

[5] S. Balaji, 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, pp. 61–65, 2001.

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

[7] H. Carrillo and D. Lipman, "The multiple sequence alignment problem in biology," *SIAM Journal on Applied Mathematics* 48, pp. 1073–1082, 1988.

[8] C. Chothia and A.M. Lesk, "The relation between the divergence of sequence and structure in proteins," *EMBO Journal* 5, pp. 823–826, 1986.

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

[10] M.O. Dayhoff, 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, pp. 345–352, 1978.

[11] G.B. Dantzig, "Linear programming and extensions," Princeton, N.J. Princeton University Press, 1963.

[12] C. Do, `CONTRAlign`: CONditional TRAining for protein sequence alignment, version 1.04, 2005.
`http://contra.stanford.edu/contralign/`

[13] C. Do, S. Gross, and S. Batzoglou, "`CONTRAlign`: discriminative training for protein sequence alignment," Proc. 10th ACM *Conference on Research in Computational Molecular Biology* (RECOMB), Springer-Verlag LNBI 3909, pp. 160–174, 2006.

[14] R.C. Edgar, "`MUSCLE`: multiple sequence alignment with high accuracy and high throughput," *Nucleic Acids Research* 32, pp. 1792–1797, 2004.

[15] D. Eppstein. "Finding the k shortest paths." *SIAM Journal on Computing* 28:2, pp. 652–673, 1998.

[16] D. Eppstein, "Setting parameters by example," *SIAM Journal on Computing* 32:3, pp. 643–653, 2003.

[17] M.L. Fredman, "Algorithms for computing evolutionary similarity measures with length independent gap penalties," *Bulletin of Mathematical Biology* 46, pp. 553–566, 1984.

[18] Z. Galil and R. Giancarlo, "Speeding up dynamic programming with applications to molecular biology," *Theoretical Computer Science* 64, pp. 107–118, 1989.

[19] O. Gotoh, "An improved algorithm for matching biological sequences," *Journal of Molecular Biology* 162, pp. 705–708, 1982.

[20] V.S. Gowri, S.B. Pandit, B. Anand, N. Srinivasan, and S. Balaji, `PALI`: Phylogeny and ALIgnment of homologous protein structures, release 2.3, 2005.
`http://pauling.mbu.iisc.ernet.in/~pali`

[21] J.R. Griggs, P. Hanlon, A.M. Odlyzko, and M.S. Waterman, "On the number of alignments of k sequences," *Graphs and Combinatorics* 6, pp. 133–146, 1990.

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

[23] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, 1988.

[24] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge University Press, New York, 1997.

[25] D. Gusfield, K. Balasubramanian, and D. Naor, "Parametric optimization of sequence alignment," *Algorithmica* 12, pp. 312–326, 1994.

[26] D. Gusfield and P. Stelling, "Parametric and inverse-parametric sequence alignment with `XPARAL`," *Methods in Enzymology* 266, pp. 481–494, 1996.

[27] S. Henikoff and J.G. Henikoff, "Amino acid substitution matrices from protein blocks," *Proc. National Academy of Sciences USA* 89, pp. 10915–10919, 1992.

[28] D.T. Jones, "Protein secondary structure prediction based on position-specific scoring matrices," *Journal of Molecular Biology* 292, pp. 195–202, 1999.

[29] N. Karmarkar, "A new polynomial time algorithm for linear programming," *Combinatorica* 4:4, pp. 373–395, 1984

[30] R.M. Karp and C.H. Papadimitriou, "On linear characterization of combinatorial optimization problems," *SIAM Journal on Computing* 11, pp. 620–632, 1982.

[31] L.G. Khachiyan, "A polynomial algorithm in linear programming," *Doklady Akademii Nauk SSSR* 244:S, pp. 1093–1096, 1979. Translated in Soviet Mathematics Doklady 20:1, pp. 191–194. 1979,

[32] J. Kececioglu and E. Kim, "Simple and fast inverse alignment," Proc. 10th ACM *Conference on Research in Computational Molecular Biology* (RECOMB), Springer-Verlag LNBI 3909, pp. 441–455, 2006.

[33] J. Kececioglu and D. Starrett. "Aligning alignments exactly." Proc. 8th ACM *Conference on Research in Computational Molecular Biology* (RECOMB), pp. 85–96, 2004.

[34] E. Kim and J. Kececioglu, "Inverse sequence alignment from partial examples," Proc. 7th EATCS/ISCB *Workshop on Algorithms in Bioinformatics* (WABI), Springer-Verlag LNBI 4645, pp. 359–370, 2007.

[35] E. Kim and J. Kececioglu, `IPA`: software for inverse parametric alignment, version 0.577, 2008
`http://ipa.cs.arizona.edu`

[36] E. Kim and J. Kececioglu, "Learning scoring schemes for sequence alignment from partial examples," will appear in *IEEE/ACM Transactions on Computational Biology and Bioinformatics*.

[37] V. Klee and G.J. Minty, "How good is the simplex algorithm?" Inequalities III, ed. O. Shisha New York: Academic Press, Inc., pp. 159–175, 1972

[38] Y. Lu and S.-H. Sze, "Multiple sequence alignment based on profile alignment of intermediate sequences," Proc. 11th *Conference on Research in Computational Molecular Biology* (RECOMB), Springer-Verlag LNBI 4453, pp. 283–295, 2007.

[39] R. Lüthy, A.D. McLachlan, and D. Eisenberg, "Secondary structure-based profiles: use of structure-conserving scoring tables in searching protein sequence databases for structural similarities," *Proteins* 10, pp. 229–239, 1991.

[40] A. Makhorin, `GLPK`: `GNU` Linear Programming Kit, release 4.8, 2005. `http://www.gnu.org/software/glpk/`

[41] W. Miller and E.W. Myers, "Sequence comparison with concave weighting functions," *Bulletin of Mathematical Biology* 50, pp. 97–120, 1988.

[42] K. Mizuguchi, C.M. Deane, T.L. Blundell, and J.P. Overington, "`HOMSTRAD`: a database of protein structure alignments for homologous families," *Protein Science* 7, pp. 2469–2471, 1998.

[43] A.G. Murzin, S.E. Brenner, T. Hubbard, and C. Chothia, "`SCOP`: a structural classification of proteins database for the investigation of sequences and structures," *Journal of Molecular Biology* 247, pp. 536–540, 1995.

[44] A. Murzin, J.-M. Chandonia, A. Andreeva, D. Howorth, L. Lo Conte, B. Ailey, S. Brenner, T. Hubbard, and C. Chothia, `SCOP`: Structural Classification of Proteins, release 1.69, 2005. `http://scop.berkeley.edu`

[45] S.B. Needleman and C.D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology* 48, pp. 443–453, 1970.

[46] L. Pachter and B. Sturmfels, "Parametric inference for biological sequence analysis," *Proc. National Academy of Sciences USA* 101:46, pp. 16138–16143, 2004.

[47] M.W. Padberg 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.

[48] J. Pei and N.V. Grishin, "`PROMALS`: towards accurate multiple sequence alignments of distantly related proteins," *Bioinformatics* 23:7, pp. 802–808, 2007.

[49] G.A. Petsko and D. Ringe, *Protein Structure and Function,* New Science Press, London, 2004.

[50] B. Rost, "Twilight zone of protein sequence alignments," *Protein Engineering Design and Selection* 12, pp. 85–94, 1999.

[51] C. Sander and R. Schneider, "Database of homology-derived protein structures and the structural meaning of sequence alignment," *Proteins* 9, pp. 56–68, 1991.

[52] V.A. Simossis and J. Heringa, "The influence of gapped positions in multiple sequence alignments on secondary structure prediction methods," *Compututational Biology and Chemistry* 28, pp. 351–366, 2004.

[53] V.A. Simossis and J. Heringa, "`PRALINE`: a multiple sequence alignment toolbox that integrates homology-extended and secondary structure information," *Nucleic Acids Research* 33, pp. W289–W294, 2005.

[54] D. Starrett, T.J. Wheeler, and J.D. Kececioglu, `AlignAlign`: software for optimally aligning alignments, version 0.9.7, 2005.
`http://alignalign.cs.arizona.edu`

[55] F. Sun, D. Fernández-Baca, and W. Yu, "Inverse parametric sequence alignment," *Journal of Algorithms* 53, pp. 36–54, 2004.

[56] J.D. Thompson, D.G. Higgins, and T.J. Gibson, "`CLUSTAL W`: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," *Nucleic Acids Research* 22, pp. 4673–4680, 1994.

[57] I. Van Walle, I. Lasters, and L. Wyns, "`SABmark`: a benchmark for sequence alignment that covers the entire known fold space," *Bioinformatics* 21:7, pp. 1267–1268, 2005.

[58] M.S. Waterman, "Efficient sequence alignment algorithms," *Journal of Theoretical Biology* 108, pp. 333–337, 1984.

[59] M.S. Waterman, M. Eggert, and E. Lander, "Parametric sequence comparisons," *Proc. National Academy of Sciences USA* 89, pp. 6090–6093, 1992.

[60] T.J. Wheeler and J.D. Kececioglu, "Multiple alignment by aligning alignments," Proc. 15th ISCB *Conference on Intelligent Systems for Molecular Biology* (ISMB), *Bioinformatics* 23, pp. i559–i568, 2007.

[61] T.J. Wheeler and J.D. Kececioglu, `Opal`: software for aligning multiple biological sequences, version 0.3.7, 2007.
`http://opal.cs.arizona.edu`

[62] C.-N. Yu, T. Joachims, R. Elber, and J. Pillardy, "Support vector training of protein alignment models," Proc. 11th *Conference on Research in Computational Molecular Biology* (RECOMB), Springer-Verlag LNBI 4453, pp. 253–267, 2007.

[63] H. Zhou and Y. Zhou, "`SPEM`: improving multiple sequence alignment with sequence profiles and predicted secondary structures," *Bioinformatics* 21:18, pp. 3615–3621, 2005.