Some Techniques of Data Mining in Biology
Part 1

Marco Valtorta
mgv@cs.sc.edu

Department of Computer Science and Engineering
University of South Carolina

March 9, 2000

Overview

A Little Molecular Biology

Challenge Problems in Computational Biology

Pairwise Sequence Alignment Using Dynamic Programming

Parsing Gene Structure Using Dynamic Programming

A Little Molecular Biology [CMMB, p.30]

The human genetic code is captured in long molecules of DNA, which are packaged in the cell nucleus in the chromosomes. The human genome consists of the DNA (and associated protein molecules, which do not matter to us).

See Figure 1 in Primer.

The DNA is composed of a long (5 feet) chain of nucleotides (A, G, C, and T).

The DNA is usually found in the form of two paired chains (or strands: the "double helix"), held together by weak bonds between pairs of nucleotides on each of the two strands (A pairs with T and G pairs with C). These pairs are called base pairs. The total number of base pairs is approximately three billion (3.3×10^9).

See Figure 2 in Primer.

DNA is organized into distinct, physically separate units called chromosomes. Most cells contain two sets of chromosomes, one from each parent. The pairs include one special pair that determines sex ((X,X) or (X,Y)) and 22 pairs of autosomes. So, each one of the two sets contains 22 autosomes and one X or Y chromosome.

See Figure 6 in Primer.

Chromosomes contain genes. The whole human genome (i.e., the human DNA molecule) contains about 100,000 genes. Each gene therefore contains about 30,000 base pairs. Of these, about 10% (3,000) carry information representing proteins. The remaining 90% do not carry information.

Proteins are long, folded sequences of the twenty common amino acids, or peptides. The information-carrying part of a gene codes a peptide by a sequence of three base pairs, called codons. Since there are 64 (4^3) codons for twenty amino acids, the code is degenerate. For example, the base sequence ATG (or the corresponding base-pair sequence AT-TA-GC) is a codon for the amino acid methionine. Since the information-carrying size of the average gene is 3,000 base pairs, the protein coded by the average gene is 1,000 amino acids long.

See Figure 5 in Primer.

Finding Genes in Bacterial Genome [Kasif99]

Prokaryotic DNA (the DNA of bacteria and archea) has a simple high-level structure that can be captured by the following context-free grammar:
Genome $\rightarrow$ Intergenic Gene Genome
Genome $\rightarrow$ Intergenic Gene Intergenic

In other words, a genome is a sequence of genes separated by intergenic material, and a genome starts and ends with intergenic material.

This leads to the following approach for finding genes:

- Learn a model for genes (M_g) that can assign to any DNA sequence the probability that it is a gene (i.e., a scoring model for genes).

- Learn a model for intergenic regions (M_i) that can assign to any DNA sequence the probability that it is intergenic material (i.e., a scoring model for intergenic material).

- Build a generative model for a genome by using the rules and the models.

- Develop a parsing algorithm that interprets any given DNA sequence by producing the most likely decomposition of the sequence into genes and intergenic material, by maximizing the sum of the individual local regions' scores.

Assuming that we have the scoring models and that there are only a few possible places where regions may start, we may use dynamic programming to find the best decomposition of a DNA sequence into genes and intergenic materials.

Glimmer and GeneMark are examples of systems that "score the likelihood that a given DNA sequence is a gene."

Finding Genes in Eukaryotic Genome

Eukaryotic genome has a more complicated structure. Genes are separated by intergenic material, as before, but now the genes themselves may be divided into regions:

Genome $\rightarrow$ Intergenic Gene Genome

Genome $\rightarrow$ Intergenic Gene Intergenic

Gene $\rightarrow$ Exon SpliceSite Intron SpliceSite Gene

Gene $\rightarrow$ Exon

Some Challenge Problems in Molecular Biology

Protein Structure Prediction

Homology Search

Multiple Alignment and Phylogeny Construction

Genomic Sequence Analysis and Gene-Finding

Pairwise Sequence Alignment Using Dynamic Programming

This is a special case of string matching. Dynamic programming algorithms similar to those used in minimum-distance editing (Wagner and Fisher, 1974) can be used.

Assume that we want to match two strings, A (of M characters) and B (of N characters). Let the reward for aligning two characters, c_1 and c_2, be $S(c_1, c_2)$, the penalty for introducing a one-character gap or extending a gap by one character be b, and assume that the value of an alignment is given by the sum of rewards minus the sum of penalties. Let $D[i, j]$ be the value of the best alignment of the first i characters of A with the first j characters of B.

Then, the recurrence that expresses (with appropriate initial conditions and bounds on the indices) the value of an alignment is:

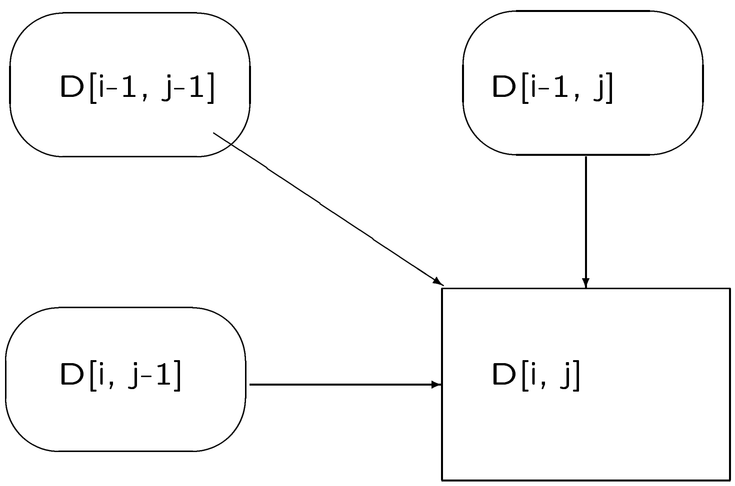
$$D[i,j] = max \begin{cases} D[i-1,j] - b \\ D[i,j-1] - b \\ D[i-1,j-1] + S(A_i, B_j) \end{cases}$$

The first case corresponds to the situation in which the match of (sub)string $A[1:i]$ with (sub)string $B[1:j]$ is obtained by introducing or extending a gap in the first string. The second case is similar, but the gap is introduced or extended in the second string. In the third case, the match is obtained simply by adding characters A_i and B_j, without introducing or extending gaps.

The proof of the recurrence is by contradiction. Informally, if there were a better alignment of $A[1:i]$ with $B[1:j]$ than the one indicated by the recurrence, then this alignment would *contain* a better way of aligning $A[1:i-1]$ with $B[1:j]$, or $A[1:i]$ with $B[1:j-1]$, or $A[1:i-1]$ with $B[1:j-1]$.

Therefore, the optimal alignment is an *extension* of the three alignments considered in the recurrence, and Bellman's principle of optimality holds for this problem.

The recurrence indicates a pattern of data dependency that can be spatially described as follows:



The data dependency graph suggests a tabular realization of a dynamic programming algorithm in which each A string character corresponds to a row and each B string character corresponds to a column. The entries in the first row and column are filled as a special case, since they each involve the match of a single character against a string. (Alternatively, one can fill a dummy "zeroth" row and column with entries $D[0, i] = D[i, 0] = i \times -b$, and avoid the special case.) The other entries are filled in one row at a time. The only three entries needed to fill in a cell are the one on the immediate left, the one immediately above, and the one immediately diagonally above and to the left.

Since the algorithm does a fixed amount of work per entry in an $M \times N$ array, its complexity is $O(NM)$.

For example, let $b = 2$, $S(.,.)$ be 5 if the arguments are equal characters, 2 if they are different vowels, and 1 otherwise, and consider the strings `ITMA` (A) and `THI` (B). By local search, we fill out the first row and column as follows:

$$\begin{array}{cccc} & T & H & I \\ I & 1 & -1 & 1 \\ T & 3 & & \\ M & 1 & & \\ A & -1 & & \end{array}$$

Then, we fill the other entries using the data dependency diagram and the recurrence. We start with the second row (corresponding to the character T). For example, $D[2,2] = \max((3-2), (-1-2), (1+1)) = \max(1, -3, 2) = 2$.

Eventually, we obtain:

	T	H	I
I	1	-1	1
T	3	2	0
M	1	4	3
A	-1	2	6

To recover a best match, we start from the bottom right corner and search locally for one of the three adjacent positions that could have resulted in the bottom right value. (There may be more than one.) We then iterate up to the first row or column. In the example, we move from $D[4,3] = 6$ to $D[3,2] = 4$, and finally to $D[2,1]$. This results in the following match:

$$\begin{array}{cccc} I & T & M & A \\ & T & H & I \end{array}$$

Here are two another examples, using the same parameter values but different sequences:

$$\begin{array}{cccc} & I & H & I \\ I & 5 & 3 & 1 \\ T & 3 & 6 & 4 \\ M & 1 & 4 & 7 \\ A & -1 & 2 & 6 \end{array} \qquad \begin{array}{cccc} I & T & M & A \\ I & & H & I \end{array}$$

$$\begin{array}{cccc} & I & H & I \\ I & 5 & 3 & 1 \\ T & 3 & 6 & 4 \\ I & 1 & 4 & 7 \\ M & -1 & 2 & 6 \end{array} \qquad \begin{array}{cccc} I & T & I & A \\ I & H & I & \end{array}$$

The Smith-Waterman Algorithm

The recurrence on which the algorithms is based is:

$$D[i,j] = max \begin{cases} D[i-1,j] - W_k \\ D[i,j-1] - W_k \\ D[i-1,j-1] + S(A_i, B_j) \\ 0 \end{cases}$$

The data dependency graph is the same.

The penalty to introduce a gap is greater than the penalty to extend a gap. Smith and Waterman [1981] propose:

$$W_k = 1 + \frac{1}{3}k$$

Parsing Gene Structure Using Dynamic Programming

"The discovery in 1977 of split genes in eukariotic genes [the genes of most non-bacterial organisms, including humans] must rank as one of the most surprising scientific findings of this century" [Burge, CMMB, p.129]. A eukariotic gene is a sequence of exons and introns, which begins and ends with an exons. Recall part of the eukariotic DNA grammar:

Gene $\rightarrow$ Exon SpliceSite Intron SpliceSite Gene
Gene $\rightarrow$ Exon
or, more simply,
Gene $\rightarrow$ Exon Intron Gene
Gene $\rightarrow$ Exon

The problem of parsing (eukariotic) genes is to segment the gene into exons and introns, or, as Wu puts is, "to predict the structure of a gene from its . . . DNA sequence. Gene structure is specified by a set of coding regions called exons that alternate with non-coding regions called introns. Predicting gene structure is complicated by the vast space of possible gene structures, a space that explodes combinatorially with the length of the DNA sequence."

There are several constraints on the form of a gene, in addition to the basic grammar. (Maybe this could be handled formally by using attribute grammars.)

Recall that genes are composed of triples of nucleotides, called codons, each of which codes a peptide. Exons are always terminated by a special codon, called stop codon. (Possible stop codons are TAA, TAG, and TGA.) Stop codons never appear in the interior of an exon. Moreover, when two exon are joined (by removing the intron that separates them), the resulting gene segment cannot contain any new stop codons.

We therefore have the following three constraints:

- *Frame constraint*: the sum of all exon lengths must be a multiple of three.

- *Internal stop constraint*: exons may not contain internal stop codons in their reading frame.

- *Merged stop constraint*: Exon-exon junctions may not form stop codons in their reading frame.

The input to the algorithm consists of signal positions and evaluation functions.

There are four types of signal positions. Two of them mark the beginning and the end of a whole gene, respectively:

- Initiation sites (denoted by B for Beginning positions): these are the positions at the beginning of a gene. Possible initiation sites are typically the codons ATG or GTG.

- Termination sites (denoted by S for Stop positions): these are the positions at the end of a gene. They are located at codons TAA, TAG, or TGA.

The other two types of signal positions mark the boundaries of an exon, where it meets an intron:

- Donor splice sites (D): these are the positions at the beginning of an intron. They are typically located at the dinucleotide GT. Schematically: `----exon----|GT--intron----`.

- Acceptor splice sites (A): these are the positions at the end of an intron. They are typically located at the dinucleotide AG. Schematically: `----intron--AG|----exon----`.

There are two evaluation functions, V_E and V_I, which score potential exons and introns, respectively. "Typically, they estimate the log-likelihood that a segment will be an exon or an intron, but this interpretation is not required by the algorithm" [Wu, p.6].

The two goals of the algorithm are:

1. to compute the maximal score for the exon and intron values over the set of gene structures that are compatible with the constraints ("consistent"):

$$\max_G \left[\sum_{E \in G} V_E(E) + \sum_{I \in G} V_I(I) \right]$$

2. to produce the gene structure that generates the maximal score.

Some notation and definitions:

- A *junction* divides two nucleotides.

- A *site* or *position* p is the junction immediately to the left of the $p-th$ nucleotide. Splice sites (junctions between introns and exons) are indicated by a vertical bar (|). Note that the position of the (donor) junction marked by GT is the index of G, but the position of the (acceptor) junction marked by AG is $p+2$, where p is the index of A. Refer to the following diagram: `--exon----|GT----intron----AG|----exon--`.

- A segment $||p, r||$ is the sequence or string of nucleotides starting with the $p-th$ one and ending with the $(r-1)st$. The length of the sequence $||p, r||$ is $r-p$. Note that segments $||p, r||$ and $||r, q||$ are contiguous.

- The absolute frame of position p is $F(p) = p \bmod 3$.

- The reading frame $f_G(p)$ of position p in exon segment E of gene structure G is the absolute frame of the codon boundaries for E. (Note that all positions in an exon have the same reading frame.) Note that we need to distinguish the reading frame, from the absolute frame, because (1) the first codon in a gene is preceded by intergenic material, whose length is not necessarily a multiple of three, and (2) we do not know where a gene starts.

- The phase $\phi(p)$ of a position p in exon segment E is the position of p relative to the reading frame of E: $\phi(p) = [p - f(p)] \bmod 3$. It follows that the reading frame can be derived from the phase: $f(p) = [p - \phi(p)] \bmod 3$.

For example, consider the following string (with the position of each character indicated below it). (We use A,B, and C for nucleotides, instead of A,C,G, and T.)

$$\begin{array}{ccccccccc} A & B & C & A & B & C & A & B & C \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array}$$

The absolute frame of B at index 5 is $5 \bmod 3 = 2$. Suppose that the gene structure G starts at index 4. Then the first codon is ABC, and the reading frame is $4 \bmod 3 = 1$.

Now, consider the following string, where we show the end of an intron and the beginning of an exon, i.e. the vertical bar marks a junction at an acceptor site.

$$\begin{array}{cccccccccc} A & B & C & A & B & C & | & A & B & C \\ 101 & 102 & 103 & 104 & 105 & 106 & & 107 & 108 & 109 \end{array}$$

The junction has absolute frame $F(107) = 107 \bmod 3 = 2$, has reading frame $f(107) = 2$ (and that is the reading frame of all nucleotides in the exon that starts at position 107), and has phase $\phi(107) = [107 - f(107)] \bmod 3 = [107 - 2] \bmod 3 = 105 \bmod 3 = 0$. Note that all first bases in a codon triplet have phase 0. Also check: $2 = f(107) = [107 - \phi(107)] \bmod 3 = [107 - 0] \bmod 3 = 2$.

- (Conservation of phase) The phase at the end of E_1 must equal the phase at the beginning of E_2, and therefore if d is a donor site at the end of E_1 and a is an acceptor site at the beginning of E_2, then $\phi(a) = \phi(d)$. For example, suppose that the donor splice site is marked by GT and the acceptor splice site is marked bt AG. Then, the phase of the splice sites marked with the vertical bar in the diagram below must be the same.
  `--exon E1---|GT--intron--AG|X--exon E2--`.
  Equivalently, by definition of phase of a site or junction (with a slight abuse of notation), the phases of the first G and of X must be the same.

Here are some segment compatibility equations. They describe which intron and exon segments may extend (or: are compatible with) partial gene structures. Also refer to Figure 2 in Wu's paper.

- (Intron Compatibility). Let d_n be the donor junction ending exon E_{n-1}. Consider an intron that extends from d_n to a_n. Let the reading frame at position a_n be $f(a_n)$. Then the phase at position d_n must be

  $$\phi(d_n) = [a_n - f(a_n)] \bmod 3 \qquad (1)$$

  Illustration:

  `En-1----W|X---In---A|BC---`

  The phase of X must equal the phase of B. ($\phi(d_n) = \phi(X)$ must equal $\phi(B) = \phi(a_n)$, and, by definition of phase, $\phi(a_n) = [a_n - f(a_n)] \bmod 3$.)

  Equation (1) indicates whether a partial gene structure is compatible with a given intron and reading frame in terms of a prerequisite phase.

- (Internal Exon Compatibility)

$$f(a_n) = [d_{n+1} - \phi(d_{n+1})] \bmod 3 \qquad (2)$$

`In---AB|C---Ex---D|EF---`

The reading frame of C must equal the reading frame of E.

Equation (2) indicates whether a partial gene structure is compatible with a given exon and phase, in terms of a prerequisite reading frame.

- (Terminal Exon Compatibility)

$$f(a_n) = F(s) \tag{3}$$

`In-----|A---Terminal exon|S`

In order for the stop codon to be read in frame, the phase of S must be zero: $\phi(s) = [s - f(s)] \bmod 3 = [s - f(a_n)] \bmod 3 = 0$.

- (Initiation Site Compatibility)

$$F(b) = [d_1 - \phi(d_1)] \bmod 3 \qquad (4)$$

```
|B---Initial Exon---|D
```

$F(b) = F(B) = D(d) = F(D)$. But $F(D) = [D_1 - \phi(D_1)] \bmod 3 = [d_1 - \phi(d_1)] \bmod 3$.

Equations (1,2,3,4) are used by Wu's algorithm to satisfy the frame constraints. Note that equations (1) suggests that introns be classified by their (final) reading frame, equation (2) suggests that exons be classified according to their (final) phase, and equation (4) suggests that initiation sites be classified according to their absolute frame.

Internal stop constraints

Consider an exon with reading frame f and ending at d. Let S_F be the set of all stop positions such that the absolute frame of s is F. The *stop boundary* is $s^*(f,d) = \max s \in S_F | s < d$

- Internal stop constraint

  `|A---Exon---|D`

  Let $f(d)$ be f. Then, the exon contains an in-frame stop codon iff $a \leq s^*(f,d)$.

- Weak stop boundary

  Wu's algorithm does not use this.

- Weak stop constraint

  Wu's algorithm does not use this.

- Merged stop constraint

  This constraint expresses the requirement that two exon-exon junctions may not form a codon in their reading frame.

  `Exon1--XYZ|D--Intron--|ABC--Exon2`
  XYZABC may not contain a stop codon: $\neg\text{stop}(||d-\phi, d|| \oplus ||a, a+3-\phi||)$.

  Note that the two halves of the interrupted codon are $||d-\phi||$ and $||a, a+3-\phi||$. The first half is added to the rest of the second exon when applying the exon value function.

Wu's Algorithm

We will introduce the major data structures, give an informal description of the algorithm, and examine the example in Figure 3 of Wu's paper before presenting the algorithm in detail.

The major data structures are four lists:

- B_F, for initiation sites with absolute frame F. (Recall that an initiation site is the position of the nucleotide that begins the gene structure.)

- I_f, for intron segments ending with reading frame f. This list contains tuples (one per intron segment) of the form $< a, p, i, c >$, where $a \in A$ is an acceptor site (which comes at the end of the intron), p is the beginning of the segment, i is the maximal cumulative score for the partial structure terminating with this intron segment, and c is the codon fragment from the preceding exon.

- E_ϕ, for internal exon segments ending with phase ϕ. This list contains tuples (one per internal exon segment) of the form $< d, p, e >$, where e is the maximal cumulative score for partial structures terminating with this exon segment.

- T, for terminal exon segments (all of which must end with phase 0). This list contains tuples (one per terminal exon segment), of the form $< s, p, t >$, where t is the maximal cumulative score for structures terminating with this exon segment.

The algorithm is a dynamic programming algorithm that exploits the following instance of Bellman's principle of optimality: A (partial) gene structure has highest value if and only if it is a compatible extension of some partial structure of highest value. (Recall that the value of a gene structure is the sum of the scores of the introns and exons that compose it.) Initially, the lists are all empty. The algorithms processes the signals in the input lists (B, of initiation sites, D, of donor splice sites, A, of acceptor splice sites, and T of termination sites) from left to right. When a new site is processed, the compatibility constraints are checked and an entry to the appropriate work list (B_F, I_f, E_ϕ, or T) is normally made. (The constraints may prevent an extension to a partial gene structure, in which case no entry is made.) At the end, the algorithm returns the maximum score among all complete gene structures: $\max_{<s,p,t>\in T} t$. The best gene structure is found by tracing backwards, with local search.

Example: See paper.

Detailed Description: See paper.

Extensions (near-optimal solutions, cardinality constraints): See paper.

Analysis (time complexity, space complexity): See paper.

Discussion: See paper.

Availability of the code: See paper.

References

The first ten pages of the (U.S.) DOE Human Genome Project "Primer on Molecular Genetics," http://www.ornl.gov/hgmis/publicat/primer/intro.html.

Two papers from the November/December 1999 issue of *IEEE Intelligent Systems* (available by following the appropriate links from the Digital Article Database Service, http//dads.aub.auc.dk): "Datascope: Mining Biological Sequences," by Simon Kasif (pp.38-43), and "Gene Discovery in DNA Sequences," by Steven Salzberg (pp.44-48).

The following paper by Thomas D. Wu (in preprint form): "A Segment-based Dynamic Programming Algorithm for Parsing Gene Structure," available at
http://cmgm.stanford.edu/(tilde)brutlag/
Papers/wu96b.pdf. (The final version was published in *J. of Comp. Biol.*, 2 (1996?), 3, 375-395.)