Fast Fourier Transform Analysis of DNA Sequences

A Thesis
Presented to
The Division of Mathematics and Natural Sciences
Reed College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts

Russell W. Hanson

May 2003

Approved for the Division
(Physics)

_____

Richard E. Crandall

Acknowledgments

I thank my friends, colleagues, and employers over the last four years for the good times, the good work, and the good pay. I salute the hackers, they know what's going on.

This research was funded in part by a grant from the Howard Hughes Medical Institute.

Table of Contents

List of Figures

Abstract

In this thesis we explain the theory of fast Fourier transform convolution, imaging of nucleotide or string sequences, and some non-standard string-matching algorithms. We also give a brief exposition of applications of signal processing techniques to biosequence analysis. We provide examples of several test cases and implementations of the algorithms with code in *Mathematica* and C. We do this with the ultimate goal of improving biosequence analysis and the prevailing BLAST schemes.

Dedication

This thesis is dedicated to Alfred Olaf Hanson.

Chapter 1

Introduction

His nature grudged thinking, for it crippled his speed in action: the labour of it shrivelled his features into swift lines of pain.

—T.E. Lawrence on Sherif Feisal

1.1 Fourier series

Physics often uses series expansions to simplify calculations or to arrive at usable approximations. The Fourier series allows us to express periodic functions in an intuitive way as a composition of the sine and cosine series. Expressing $\sin nx$ and $\cos nx$ in terms of complex exponentials, yields:

$$\begin{aligned} \sin nx &= \frac{e^{inx} - e^{-inx}}{2i}, \\ \cos nx &= \frac{e^{inx} + e^{-inx}}{2}. \end{aligned} \tag{1.1}$$

Then we can compose our function as

$$f(x) = c_0 + c_1 e^{ix} + c_{-1} e^{-ix} + c_2 e^{2ix} + c_{-2} e^{-2ix} + \ldots = \sum_{n=-\infty}^{n=\infty} c_n e^{inx}. \tag{1.2}$$

The sufficient conditions for the Fourier series to converge are the Dirichlet conditions:

1. $f(x)$ may only have a finite number of finite discontinuities, and

2. $f(x)$ may only have a finite number of extreme values.

Satisfying the Dirichlet conditions on a computer is one of the easier tasks we shall encounter.

A periodic function $f(x)$ is defined as a function for which

$$f(x + T) = f(x), \tag{1.3}$$

thus T is understood to be the *period* of the function.

Just note briefly that the real Fourier series that Fourier (the man) studied, and the complex Fourier series have no real differences. Let

$$a_n = c_n + c_{-n}, \qquad b_n = i(c_n - c_{-n}),$$

and, for $n = 0$,

$$c_0 = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-i0x}dx = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)dx = \frac{a_0}{2}.$$

Then from Equation (1.2),

$$\begin{aligned} f(x) &\sim \sum_{-\infty}^{\infty} c_n e^{inx} = \sum_{n=1}^{\infty}\left[c_n e^{inx} + c_{-n}e^{-inx}\right] \\ &= c_0 + \sum_{n=1}^{\infty}\left[c_n(\cos nx + i\sin nx) + c_{-n}(\cos nx - i\sin nx)\right] \\ &= c_0 + \sum_{n=1}^{\infty}\left[(c_n + c_{-n})\cos nx + i(c_n + c_{-n})\sin nx\right] \\ &= \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n\cos nx + b_n\sin nx) \qquad \text{see (1.4).} \end{aligned}$$

A historical note: Fourier famously showed that any periodic function can be composed of sine and cosine terms in his book *Théorie analytique de la chaleur* (Fourier, 1820). He was not the first, as the essential formulas were known to Gauss, Bernoulli, and Euler. Fourier was interested in a theory of heat and developed methods of integration with trigonometric series to calculate the propagation of heat in a rectangular prism, the movement of heat in a solid cube, or the free movement

of heat on an infinite line. Bernoulli on the other hand was trying to resolve the difficulties of calculating the movement of vibrating strings using similar trigonometric series. These mathematicians arrived at the conclusion that a nicely periodic function f on an interval 2π can be represented as the sum of the trigonometric series

$$\frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n \cos nx + b_n \sin nx). \tag{1.4}$$

The solution posed by the geometry of vibrating strings supposes that an arbitrary function can always be developed as a series of arc lengths of cosines and sines. To Fourier the most complete proof of this proposition consisted of resolving a given function in a series for which one has determined the coefficients.

Fourier described it in this way: if in the series

$$a + b\cos x + c\cos 2x + d\cos 3x + e\cos 4x + \ldots$$

the value of x is inverted, the series stays the same. The sum of the series also remains the same if one adds to x any multiple of the circumference 2π. Thus in the equation

$$\frac{\pi}{2}\phi(x) = \frac{1}{2}\int \phi(x)dx + \cos x \int \phi(x)\cos x dx + \cos 2x \int \phi(x)\cos 2x dx + \cos 3x \int \phi(x)\cos 3x dx + \ldots, \tag{1.5}$$

the function ϕ is periodic and is expressed as a curve composed of arcs of equal length; each of the arcs comprises two symmetric branches which correspond to the two halves of the interval 2π. A similar expansion to (1.5) can be made using sines.

Next we will describe Figure (1.1) and outline how Fourier proves, using an arc length argument, how a function F can be represented by functions ϕ and ψ of a trigonometric series. In the figure, ψ is symmetric around 0 on the interval $-\pi$ to $+\pi$ such that

$$\psi(x) = -\psi(-x);$$

similarly, for ϕ

$$\phi(x) = \phi(-x). \tag{1.6}$$

Thus any function $F(x)$ when traced on a symmetric interval can be divided into two functions ϕ and ψ. The third function in the figure, $f(x)$, is equal to $F(-x)$ by the lengths of the arc measures $F'F'mFF$ and $f'f'mff$. So,

$$F(x) = \phi(x) + \psi(x) \qquad \text{and} \qquad f(x) = \phi(x) - \psi(x) = F(-x),$$

and solving the two equations for $F(x)$ and $f(x)$ for $\phi(x)$ and $\psi(x)$ we get equations (1.6) after noting:

$$\phi(x) = \frac{1}{2}F(x) + \frac{1}{2}F(-x) \qquad \text{and} \qquad \psi(x) = \frac{1}{2}F(x) - \frac{1}{2}F(-x).$$

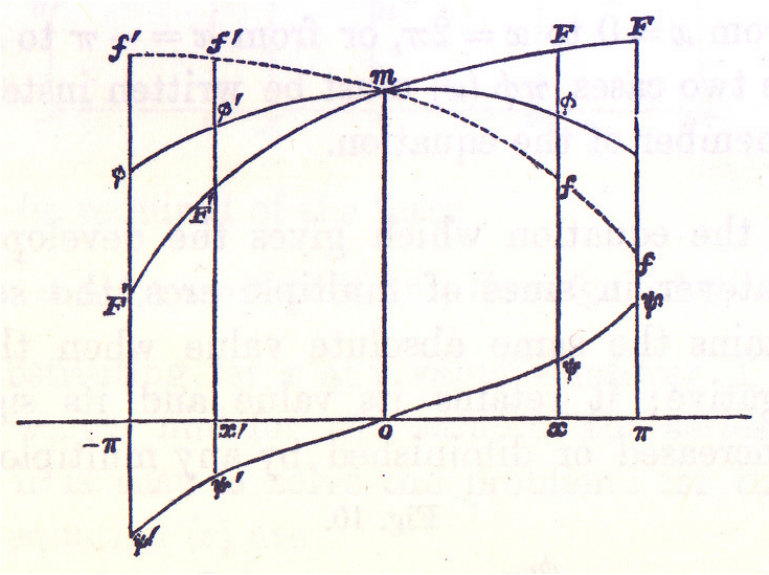

Figure 1.1: **Fourier's picture proof.** Fourier proves with this picture that an arbitrary function $F(x)$ can be expressed as the sum of two other functions. $F(x) = \phi(x) + \psi(x)$, where ϕ and ψ are cosines and sines of multiple arcs.

1.2 Fourier transform

The Fourier transform of a function $f(x)$ is the new function $F(x)$:

$$F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(u) e^{ixu} du. \tag{1.7}$$

The general form of an **integral transform** consists of a kernel $K(\alpha, t)$ and the function we're transforming $f(x)$:

$$g(\alpha) = \int_a^b f(t)K(\alpha, t)dt. \tag{1.8}$$

The following are the kernels of several other named transforms:

- $e^{-\alpha t}$ Laplace transform;
- $J_n(\alpha t)$ Hankel transform;
- $t^{\alpha-1}$ Mellin transform.

In the integral transform below (1.9), the period of the function is $2l$ and the frequency of each of the terms c_n is $n/(2l)$,

$$\begin{aligned} f(x) &= \sum_{-\infty}^{\infty} c_n e^{in\pi x/l}, \\ c_n &= \frac{1}{2l}\int_{-l}^{l} f(x)e^{in\pi x/l}dx. \end{aligned} \tag{1.9}$$

In engineering, one often contrasts finite and infinite domains, or functions that are discrete or continuous in time. In our case, we will consider the finite duration discrete time Fourier transform (DTFT or DFT). The infinite duration continuous time Fourier transform is given by

$$X(j) = \int_{-\infty}^{\infty} x(t)e^{-2\pi ijt}dt \qquad j \in (-\infty, \infty).$$

The reader can easily permute the different limits and domains to obtain the other categories of transforms.

1.3 Discrete Fourier transform

The **discrete Fourier transform** (DFT) takes an input basis vector of complex numbers

$$x = \{x_0, \ldots, x_{N-1}\},$$

where the length N of the vector is a fixed parameter. The k-th element X_k of the transformed complex vector $X_0, \ldots, X_{N-1}$ is:

$$X_k = \sum_{j=0}^{N-1} x_j \, e^{-2\pi i j k/N}. \tag{1.10}$$

The inverse Fourier transform reverses the process; it maps N complex numbers (the X_j's) into N complex numbers (the x_j's), i.e. $X \mapsto x$:

$$x_j = \frac{1}{N} \sum_{k=0}^{N-1} X_j \, e^{+2\pi i j k/N}. \tag{1.11}$$

The only differences between the inverse and the forward transforms are the sign of the exponential and dividing by N. The straightforward implementation of this is a matrix-vector multiplication

$$X_k = x_j F_{jk},$$

which clearly requires $O(n^2)$ operations (plus a twiddle for the exponents). $F_{jk} = exp(-2\pi i/n)^{jk} = w_n^{jk}$ is the primitive n-th root of unity for a field modulo a prime p, e.g.

$$w_n \equiv \left(e^{-2\pi i/n}\right)^n \equiv 1.$$

1.4 Fast Fourier transform

The modern origin of the **fast Fourier transform** (FFT) is an article by Danielson and Lanczos in 1942 (Danielson & Lanczos, 1942), though methods for computing the DFT can be traced to the time of Gauss. They showed that the DFT for a vector of length N can be broken down into two parts with half the running time. Consequently, the so-called *Danielson-Lanczos lemma* allows the recursive application of these divisions. A consequence of this is that in implementations one should only use vectors of length $N = 2^n$ with $n \in \mathbb{Z}$.

For even N, $N = 2m$:

$$\begin{aligned} X_k &= \sum_{j=0,\ \text{even}}^{N-1} \omega_n^{kj} x_j + \sum_{j=0,\ \text{odd}}^{N-1} \omega_n^{kj} x_j \\ &= \sum_{j'=0}^{m-1} \omega_n^{k(2j')} x_{2j'} + \sum_{j'=0}^{m-1} \omega_n^{k(2j'+1)} x_{2j'+1} \\ &= \sum_{j'=0}^{m-1} \omega_m^{kj'} x_{j'}^{\text{even}} + \omega_n^k \sum_{j'=0}^{m-1} \omega_m^{kj'} x_{j'}^{\text{odd}} \end{aligned}$$

since $\omega_n^{2k} = \omega_{n/2}^k$, and writing $x_k^{\text{even}} = x_{2k}$ and $x_k^{\text{odd}} = x_{2k+1}$.

There are many other FFT algorithms optimized for different uses. A standard reference like *Numerical Recipes* (Press et al., 1992) provides detailed descriptions of several of these variants.

Why is it that the FFT has a running time of $O(n \log n)$? Well, running through the array once takes $O(n)$ time, then the recursive function call on each half results in:[1]

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \log n). \end{aligned}$$

1.5 Translates and characters

What is the reason for considering expansions $\cos ax$ and $\sin bx$, or the equivalent complex exponentials e^{iax}? The explanation of this requires the introduction of the **translation operators** T_a, where a is a group element, with the properties:

1. $T_a f(x) = f(x - a)$
2. $T_0 = I$ (identity automorphism of $\mathbb{C}$)
3. $T_{a+b} = T_a T_b$
4. $T_{-a} = T_a^{-1}$.

[1] Strictly speaking $O(n)$ is an asymptotic upper bound and $\Theta(n)$ is an asymptotic tight bound. That is when neglecting constant multiplicative factors c_n, $f = c_n O(g)$ means f is bounded above by g in the asymptotic limit. $f = \Theta(g)$ is defined to mean $f = O(g)$ and $g = O(f)$.

We arrive at the definition of a *character* by observing elementary properties of the set of finite linear transformations of translates $T_a f$ of f, $f \in \mathbb{C}$. For $f \neq 0$ to each group element a there corresponds a complex scalar $\chi(-a)$ such that

$$T_a f(x) = \chi(-a) f(x).$$

Then by (1), the first property of translation operators, for all pairs (x, a),

$$f(x - a) = \chi(-a) f(x). \tag{1.12}$$

Note that if $x = 0$, $f(-a) = f(0)\chi(-a)$, which shows that χ is continuous and non-zero. Then by the properties above and (1.12), we get the result:

$$\chi(a + b) = \chi(a)\chi(b). \tag{1.13}$$

By convention, any complex-valued function $\chi \neq 0$ satisfying (1.13) is called the **character** of the group in question. Immediate results are that $\chi(0) = 1$, i.e. it is non-vanishing, and $\chi(-a) = \chi(a)^{-1}$. For the RHS χ, a homomorphism from an additive group to a multiplicative group, has the standard property $\chi(a^{-1}) = \chi(a)^{-1}$, when written additively on the left. If a character χ is bounded, then (1.13) shows that $|\chi(a)| = 1$ for all group elements a. Therefore χ defines a homomorphism of the group into the multiplicative group of complex numbers of absolute value 1. Standard mathematical references on Fourier series like (Edwards, 1979) provide a much more thorough explanation, but we need the formalism of characters for the following section.

1.6 Convolution and correlation

In this section we briefly state the basic convolution equations and the Fourier transform version, prove the convolution theorem from the definitions, and then relate the notation to what is minimally used in engineering and in many articles using applications of convolution/correlation.

The sequence vectors we convolve are

$$\text{data:} \qquad \text{“}d_i, \ldots, d_{N-1}\text{”} \qquad 0 \le i \le N-1$$

$$\text{query:} \qquad \text{“}q_i, \ldots, q_{L-1}\text{”} \qquad 0 \le i \le L-1$$

and the signal sequence c_n

$$\text{signal:} \qquad \text{“}c_i, \ldots, c_{L+N-2}\text{”} \qquad 0 \le i \le L+N-2.$$

The **convolution** is given by

$$c_n = \sum_{k=0}^{N-1} q_{n-k} d_k \qquad n = 0, \ldots, L+N-2, \tag{1.14}$$

where it is understood that $q_{n-k} = 0$ if $i = k$ is less than zero. It takes NL multiplications to compute this directly.

The **correlation** is very closely related to convolution and is given by

$$c_n = \sum_{k=0}^{N-1} q_{n+k} d_k \qquad n = 0, \ldots, L+N-2, \tag{1.15}$$

where $q_{n+k} = 0$ if $n + k \ge L$. The correlation can be computed as a convolution simply by reading one of the two sequences backwards.

Take

$$\alpha_j = \sum_{k=0}^{N-1} \beta_k e^{2\pi i j k/N},$$

where β_k is the coefficient of the inverse Fourier transform. For the complete cycle modulo N, when $N = jk$, $j = N/k$ is a period. Then the correlation value is c_n for data and query vectors, $\mathbf{D}$ and $\mathbf{Q}$ respectively,

$$\begin{aligned} c_n &= \sum_{\mu} q_\mu d^*_{\mu+n} = \frac{1}{N^2} \sum_{\mu} \left(\sum_k Q_k e^{2\pi i k \mu/N} \right) \left(\sum_l D^*_l e^{-2\pi i l(\mu+n)/N} \right) \\ &= \frac{1}{N} \sum_{l \equiv (l \pmod N)} Q_l D^*_l e^{-2\pi i l n/N}. \end{aligned} \tag{1.16}$$

The order of the elements flips around $N/2$. Asterisk '$*$' denotes complex conjugation. The *Mathematica* version of this is in Appendix (B.1).

One can also write the convolution in integral form:

$$g(t) = f(t) * h(t) = \int_{-\infty}^{\infty} f(\tau)h(t-\tau)d\tau. \tag{1.17}$$

The functions $f(t)$ and $h(t)$ can be anything with the sufficient conditions: (1) they both must be absolutely integrable on the two intervals $(-\infty, 0]$ and $[0, \infty)$; (2) $f(t)$ OR $h(t)$ must be absolutely integrable on $(-\infty, \infty)$. For example, let $f(t) = 3$ and $h(t) = \sin(t)$ on the interval $0 < t < \pi$,

$$g(t) = \int_0^{\pi} 3 * \sin(t-\tau)d\tau = -6\cos(t). \tag{1.18}$$

To an engineer this is very useful since if he knows the impulse response of a system, usually in the form of the response to a delta function input, he can use convolution to find the output when the system is fed some input. Stated slightly differently, the output of a system is the convolution of the input and the impulse response.

Similarly, one can always find the inverse transform of a transfer function, for example, by looking at a table of Laplace transforms, or by built-in *Mathematica* functions. The impulse response is often the result of some transfer function stimulus. E.g.,

```
In[7]:= LaplaceTransform[Cosh[c*t], t, a]
```

Out[7]= $\frac{a}{a^2-c^2}$

```
In[8]:= InverseLaplaceTransform[%, a, t]
```

Out[8]= $\frac{1}{2}(e^{-ct} + e^{ct})$.

We have given an electrical-engineering definition of convolution. Some algebraists prefer a different formalization, which I give here because it is so different, or at least when written uses such different notation. See for example (Fiqueroa & Gracia-Bondia, 2000). One can convolve elements as well as functions.

Given a unital algebra (A, m, u) with a unit map $u \circ \epsilon$ and a counital coalgebra (C, Δ, ϵ) over a field $\mathbb{F}$, the convolution of two *elements* f, g of the vector space of $\mathbb{F}$-linear maps $\mathrm{Hom}(C, A)$ is defined as the map $f * g \in \mathrm{Hom}(C, A)$ given by the

composition

$$C \xrightarrow{\Delta} C \otimes C \xrightarrow{f \otimes g} A \otimes A \xrightarrow{m} A.$$

This product turns $\mathrm{Hom}(C, A)$ into a unital algebra, where $\mathrm{Hom}(C, A)$ signifies the space of homomorphisms of C and A. Fourier transform convolution and the easily-identifiable delta function match locations also exist in the world of quantum computers. Shor's algorithm uses the quantum Fourier transform for factoring and other algorithms use Fourier transforms for signal processing. While some transforms can give good indications of periodicity, for example the Walsh transform (Walsh, 1923), which has also seen applications in computational biology, the relation of the FFT to cyclic cross-correlation seems unique. In Appendix (A) we will give a better explanation of quantum search algorithms, and hint at quantum convolution. Such quantum search algorithms can perform searches in $O(\sqrt{N})$-time which is a great deal better performance than $O(N \log N)$.

Definition 1.6.1. $\mathbb{Z}_n$ are the integers mod n, $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$. f are functions from $\mathbb{Z}_n \mapsto \mathbb{C}$, which we often write in the form $\{f(0), f(1), \ldots, f(n-1)\}$. With an inner product

$$\langle f, g \rangle = \sum f(j) g^*(j),$$

and let $\chi_p(q) = e^{2\pi i p q / N}$. The operator R is the reverse of f such that $Rf \in \mathbb{C}(\mathbb{Z}_n)$:

$$Rf(p) = f(-p).$$

The two orthonormal bases for $\mathbb{C}(\mathbb{Z}_n)$ are:

$$\left\{ \frac{\chi_p}{\sqrt{N}} \right\}_{0 \le p \le n-1} \qquad \text{and} \qquad \{\delta_p\}_{0 \le p \le n-1}.$$

Proof that the one is an orthonormal basis:

Proof.

$$\begin{aligned}
\left\langle \frac{\chi_p}{\sqrt{N}}, \frac{\chi_q}{\sqrt{N}} \right\rangle &= \sum \chi_p(r)\chi_q^*(r) \\
&= \frac{1}{N}\sum_r e^{2\pi ipr/N} e^{-2\pi iqr/N} \\
&= \frac{1}{N}\sum_r^{N-1} e^{2\pi ipr(p-q)/N} \\
&= \frac{1}{N}\left[\frac{1-\left(e^{2\pi i(p-q)/N}\right)^N}{1-e^{2\pi i(p-q)/N}}\right] \qquad e^{2\pi i(n\in\mathbb{Z})} = 1 \\
&= 0.
\end{aligned}$$

□

The other is the delta function, $\delta_q(r) = 1$ when $q = r$ and obviously $\delta_{pq} = 0$. Also, $\chi_q \cdot \chi_p = N$, hence the normalization conditions above.

The Fourier transform of a character is a delta function:

$$\hat{\chi}_q(p) = \langle \chi_q, \chi_p \rangle = n\delta_{qp} = \delta_q(p).$$

The Fourier transform of a delta function is a character:

$$\hat{\delta}_q(p) = \langle \delta_q, \chi_p \rangle = \sum \delta_q(r)\chi_p^*(r) = \chi_p^*(q) = \chi_q(-p) = (R\chi_q)(p).$$

So,

$$\boxed{\hat{\chi}_q = n\delta_q} \qquad \text{and} \qquad \boxed{\hat{\delta}_q = R\chi_q.}$$

The Fourier transform is not entirely linear, e.g. here F is a Fourier transform:

$$\begin{aligned}
(\widehat{Rf}) &= R\hat{f} \\
FRf &= RFf.
\end{aligned}$$

Proof. If $f \in \mathbb{C}(\mathbb{Z}_n)$

$$\begin{aligned}(Rf)\hat{}(p) &= \langle Rf, \chi_p \rangle = \sum_{k=0}^{n-1} Rf(k)\chi_p^*(k) \\ &= \sum_{k=0}^{n-1} f(-k)\chi_{-p}^*(-k) = \sum_{k=0}^{n-1} f(k)\chi_{-p}^*(k) \\ &= \langle f, \chi_{-p} \rangle = \hat{f}(-p) = (R\hat{f})(p).\end{aligned}$$

□

So,

$$\boxed{R(\hat{f}) = (Rf)\hat{}.}$$

Two applications of the Fourier transform equal itself times a normalization factor and a reversal, for all $q \in \mathbb{Z}_n$

$$\begin{aligned}\hat{\hat{\delta}}_q &= (\widehat{R\chi_q}) = (R\hat{\chi}_q) \\ &= R(n\delta_q) = nR\delta_q \\ \hat{\hat{\delta}}_q(p) &= nR\delta_q(p)\end{aligned}$$

$$\boxed{\hat{\hat{f}} = nRf} \quad \text{or} \quad \boxed{\frac{1}{n}R\hat{\hat{f}} = f.}$$

Hence the Fourier transform is inevitable and the inverse Fourier transform is the mapping $f \mapsto \check{f}$ where $\check{f} = \frac{1}{n}R\hat{f}$,

$$(Rf)\check{} = \frac{1}{n}R\widehat{Rf} = \frac{1}{n}RR\hat{f} = \frac{1}{n}\hat{f}.$$

There is no reverse-free, R-free version of the transform since:

$$\begin{aligned}\hat{\delta}_p &= \chi_p \\ \hat{\chi} &= \delta_{-p}.\end{aligned}$$

Now we prove the **convolution theorem**, that is we show that the Fourier transform of a convolution is the same as the product of Fourier transforms and that the convolution is equal to the inverse transform on the product of two forward

transforms,

$$\boxed{(f * g)\hat{} = \hat{f}\hat{g}}$$

$$\boxed{(f * g) = (\hat{f}\hat{g})\check{}\,.}$$

Proof. For all $p \in \mathbb{Z}_n$

$$\begin{aligned}
(\hat{f}\hat{g})(p) &= \hat{f}(p)\hat{g}(p) \\
&= \sum_{j=0}^{n-1} f(i)\chi_p^*(j) \cdot \sum_{k=0}^{n-1} g(k)\chi_p^*(k) \\
&= \sum_{j=0}^{n-1}\sum_{k=0}^{n-1} f(j)g(k)\chi_p^*(j+k) \qquad [\text{let } k = l - j] \\
&= \sum_{j=0}^{n-1}\sum_{l=0}^{n-1} f(j)g(l-j)\chi_p^*(l) \\
&= \sum_{l=0}^{n-1}\sum_{j=0}^{n-1} f(j)g(l-j)\chi_l^*(p) \\
\text{so} \qquad \hat{f}\hat{g} &= \sum_{l=0}^{n-1}\sum_{j=0}^{n-1} f(j)g(l-j)R\chi_l(p) \qquad [\text{by } \chi_p^*(q) = \chi_{-p}(q) = \chi_q(-p) = R\chi_q(p)].
\end{aligned}$$

Observe for example $\chi_p^*(q) = \chi_{-p}(q)$, since $e^{-2\pi ipq/n} = e^{2\pi i(-p)q/n}$. □

And the second relation:

Proof.

$$\begin{aligned}
(f * g)(p) &= (\hat{f}\hat{g})(p)\check{} \\
&= \sum_{l=0}^{n-1}\sum_{j=0}^{n-1} f(j)g(l-j)R\chi_l(p)\check{} \qquad [\text{note } R\chi_l(p)\check{} = \frac{1}{n}R(\widehat{R\chi_l}) = \frac{1}{n}RR\hat{\chi}_l] \\
&= \sum_{l=0}^{n-1}\sum_{j=0}^{n-1} f(j)g(l-j)\frac{1}{n}\hat{\chi}_l(p) \\
&= \sum_{l=0}^{n-1}\left(\sum_{j=0}^{n-1} f(j)g(l-j)\right)\delta_l(p) \\
&= \sum_{j=0}^{n-1} f(j)g(p-j)
\end{aligned}$$

which we recognize as the familiar convolution. □

The Fourier transform of a convolution is the product of Fourier transforms, or one could say the Fourier transform of a "product" is a product of Fourier transforms. The use of the term "convolution" is arbitrary, and an entity with its properties could just as easily be called a "product." We re-express the convolution in terms of the cyclic cross-correlation notation used in Rajasekaran (Rajasekaran et al., 2002) as follows. Call the cyclic correlation $\mathbf{z}_{cyc} = (z_0, \ldots, z_{n-1})$, for complex vectors $\mathbf{x}$ and $\mathbf{y}$, with a point-wise ("dyadic") product $\mathbf{x} \odot \mathbf{y} = (x_0 y_0, x_1 y_1, \ldots, x_{n-1} y_{n-1})$. Following the custom of denoting the FFT of a vector with the corresponding capital letter, the FFT is $\mathbf{Z}_{cyc} = \mathbf{X} \odot \mathbf{Y}_R$,

$$\begin{aligned} \mathbf{z}(j) &= \sum_{k=0}^{n-1} x_{j+k} y_k \\ &= \sum_{k=0}^{n-1} Rx(-j-k) y(k) \\ &= (Rx * y)(-j) \\ &= R(Rx * y)(j). \end{aligned}$$

Returning to Rajasekaran's notation $\mathbf{z}_{cyc} = R(Rx * y)$ and the inverse we have:

$$\begin{aligned} \mathbf{z}_{cyc}\hat{} &= R(Rx * y)\hat{} \\ &= R[(Rx)\hat{} \odot y\hat{}] \\ &= [R(Rx)\hat{}] \odot (R\hat{y}) \\ &= (RR\hat{x})(\widehat{Ry}) \\ &= \hat{x}\widehat{Ry}. \end{aligned}$$

This concludes our elaboration and discussion of Rajasekaran's five-line theory section.[2]

[2]Two excellent books giving an exposition of the DFT and convolution are (Bracewell, 1986) and (Heideman, 1988).

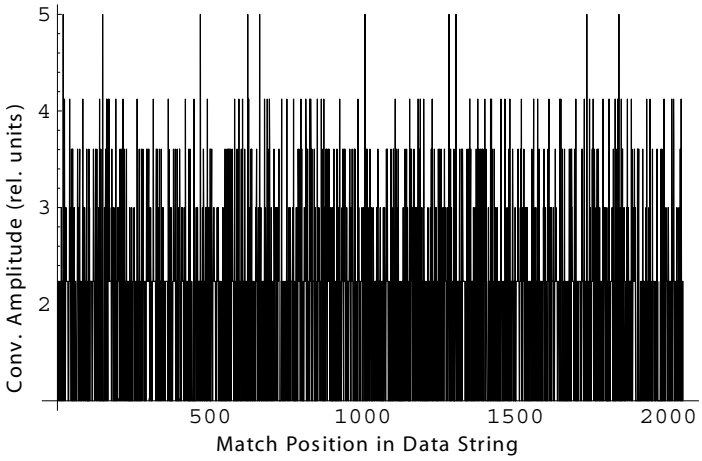

Figure 1.2: **Convolution theorem example.** The height of the peaks is equal to the length of the query string, in this case 5 units; this example used randomly-generated data.

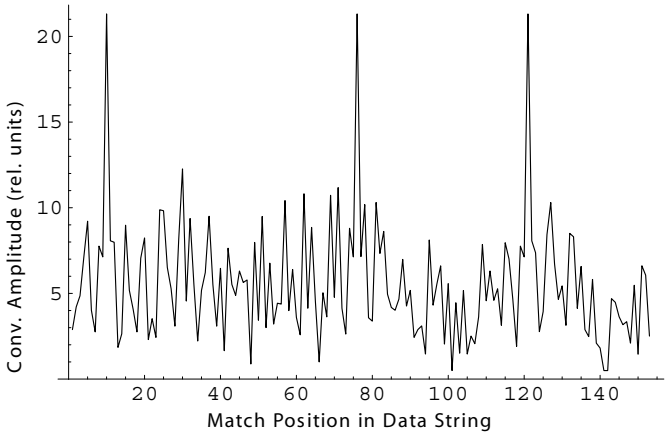

Figure 1.3: **Convolution string matching example.** Uses a 22-nucleotide long DNA query string and a very short data string.

1.7 Preliminary results

The mathematics above provides some of the theory for convolution and Fourier transforms. Our interest is in performing FFT convolution on a set of DNA sequences to find match positions. Briefly, it is necessary to convert nucleic acids to defined mathematical quantities. We do this by requiring for a nucleic acid sequence

x that $x_j \in \{1, i, -1, -i\}$, and by the mappings: Adenine $\rightarrow 1$, Cytosine $\rightarrow -i$, Guanine $\rightarrow i$, and Thymine $\rightarrow -1$. This allows us to perform a discrete Fourier transform as in Equation (1.10):

$$X_k = \sum_{j=0}^{N-1} x_j \, e^{-2\pi i j k / N},$$

and a correlation operation as in Equations (1.16) and (3.9),

$$x \times y = DFT^{-1}(DFT(x) \odot DFT(y^*)).$$

Where sequence x is a query, e.g. a primer product, and sequence y is a database, e.g. Human chromosome 1. Then the complex vector $c \equiv x \times y$, and any occurrence of $|c_n|^2 = \dim(x)$ indicates a match of length $\dim(x)$ at location y_{N-1-n}. A good portion of Chapter 3 is dedicated to explaining these steps in greater detail.

Figure (1.4) shows the first "real" DNA sequences we correlated. The result is similar to figures published in (Cheever et al., 1991).

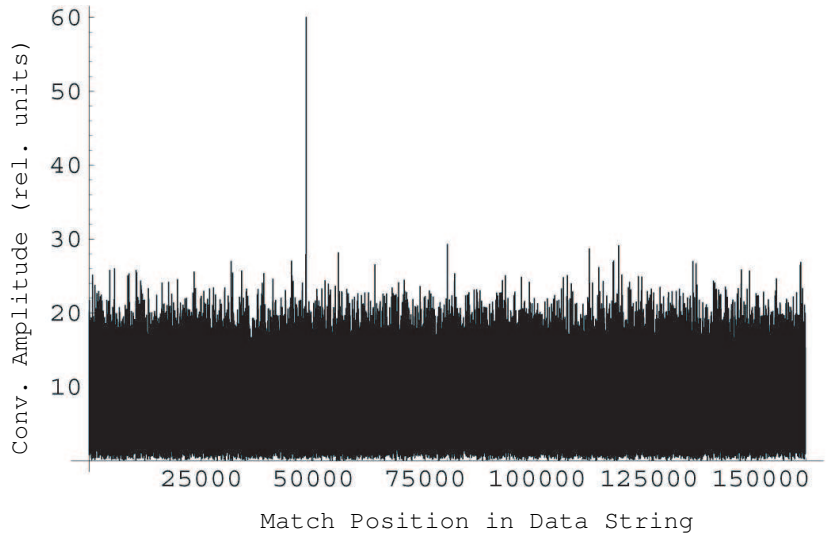

Figure 1.4: ***Homo sapiens* chromosome 1 and primer product correlation.** The delta function at ˜50,000 on the match position axis indicates a match of length equal to the convolution amplitude at base pair position $150,000 - 50,000 \approx 100,000$. The database used was the *Homo sapiens genomic contig sequences database* which contains 1,395 sequences with 2,826,392,627 total letters. A sequence between primers including primers at both ends is called a "product." The primers were selected using http://www-genome.wi.mit.edu/cgi-bin/primer/primer3_www.cgi. A positive control was performed using `BLASTN 2.2.4 [Aug-26-2002]` at http://www.ncbi.nlm.nih.gov/blast/Blast.cgi.

Chapter 2

Introduction to sequence analysis

學而不厭，誨人不倦

—Confucius[1]

2.1 BLAST

Basic Local Alignment Search Tool (BLAST) is the dominant tool for bio-sequence analysis. The National Center for Biotechnology Information (NCBI) recognized the need for tools to do sequence comparisons and database queries and published BLAST in 1990 (Altschul et al., 1990).

There are three main algorithmic steps to BLAST's method for finding **maximal segment pairs** (MSP's):

1. compiling a list of high-scoring words;
2. scanning the database for hits;
3. extending the hits found.

For DNA, it uses a word list with a distribution of contiguous w-mers in the query sequence. Often the length of $w = 12$. Thus a query of length n yields a list of $n - w + 1$ words, which commonly represent a few thousand words in the list.

[1] *"Xúe ér bú yàn,*
hùi rén bú juàn
To learn without tiring,
To teach without growing weary"

High-scoring segment pairs (HSPs) are related to MSPs in the sense that MSPs $\subseteq$ HSPs. The idea behind MSPs is that given two sequences A and B, we are looking for all pairs (a, b) for which a and b are subsequences of A and B, respectively. Both a and b are of the same length, and the software empirically sets a similarity score S. If the two subsequences can be neither shrunk nor expanded to increase the score S, then they form a MSP.

MSP's are calculated which maximize the score and, thus, the degree of sequence homology. They are functionally defined as a highest scoring pair of *identical length* segments chosen from two sequences. An MSP score for two sequences may be calculated using dynamic programming (DP) in computer time proportional to the product of their lengths. The optimization problem thus addressed by dynamic programming is to find the longest common subsequence.

BLAST for DNA is a typical exclusion-type algorithm. In the first step it locates hits or initial survivors of the fixed word length w by looking for *exact* matches. In the second step the hit is examined, and the MSP extension scheme is used. This is to be contrasted to either typical exclusion-type algorithms, like the BYP method, or the partition method for approximate matching. Furthermore, since the original BLAST disallows gaps, a dynamic programming algorithm for finding an edit path between sequences allowing the introduction of gaps addresses an entirely different problem.

Position-Specific Iterated BLAST (PSI-BLAST) introduced a refinement on the MSP idea used in the original BLAST. This method uses a so-called two-hit approach. The database is scanned for query words which when paired with the database have a score at least T. If the alignment passes this criterion, it is reported. Any such hits are termed *hits*.

After finding the initial hits, the second step is to examine the hits and see if they lie within an alignment with a score above S. Determining the hit locations is done by a look-up table hashing method. Every w-mer in the database is converted, i.e. hashed,[2] into an integer. Using this index a search may be made between the query and the database. Corresponding matching w-mers are listed. A finite state machine can also perform the functionality of this second step.

[2] A "hash" is a binary relation or associative array which assigns to each array element a positive integer via a deterministic "hashing" algorithm.

During the second "w-mer" step two filters are applied. The filters act to compensate for locally-biased distributions of bases and highly-repeated sections. The first filter works then as a means of limiting less statistically-significant words. It removes an uninformative word when an 8-tuple occurs more often than predicted by chance. For the second filter words from the query which are known to be from highly repetitive sections are removed from the search in the database. Thus for full database searches, while matches from the query to the known sublibrary of repetitive sequences are reported, matches of such words between the query and the database are not reported.

The third step expands each hit to a maximal segment pair. If the score, i.e. the number of aligned elements, is higher than S, it is registered as a highest-scoring segment pair. To find maximal segment pairs from high-scoring segment pairs requires extending the hits until the running alignment score drops below the score S, the maximum score yet attained. This extension step, producing alignments which are maximal segment pairs, accounts for upwards of 90% of BLAST's running time. Consequently, many efforts have been made to increase the speed of this step. One way to accomplish this is by reducing the number of extensions that need to be performed. This can be done by increasing the values of T and S, but would reduce the sensitivity to weak alignments.

The hashing step which performs the first "scan" step is done by what we consider a brute-force method. The four letters A, C, G, T are each encoded as two bits (A is 00, C is 01, G is 10, and T is 11). Then an array of all possible w-letter words is generated. The number corresponding to each particular word location in the database is recorded: this makes an associative array of words with a list of their locations. This is brute-force in the sense that as a w-length frame moves through the query and database sequences, as many as $w - 1$ elements of the last frame position have already been examined. Possible optimizations could take advantage of the redundancy of so many letters as the frame moves along a sequence. While reports from NCBI suggest that this step is not where the majority of the computing time is being spent (most time is spent in the extension step), it is particularly this step which is comparable, if not much faster, to FFT convolution methods. Brute-forcing all 8-mers with the FFT associated with their delta-function locations would

```
Words (2^16 bits)       Location

AAAAAAAA                23, 254, 30158, 30166
AAAAAAAC                55, 232
AAAAAAAG

(...)

TTTTTTTT
```

Figure 2.1: **BLAST brute-force table lookup**

be a comparable strategy.

This particular design means that the running time for locating w-mers doesn't have clear bounds. The running time may range from $O(n)$ to $O(n^2)$. For instance if one BLASTs the human genome's $3 * 10^9$ base pairs with itself, setting w equal to the length of the database, it will never finish.

One can also think of every w-mer as an integer in the range from $[1, 4^w]$ for DNA sequences, or $[1, 20^w]$ for protein sequences. This results in an array of size $4^8 = 65536$ for the 8-mers we enumerate in Figure (2.1).

With the help of mathematicians and programmers, BLAST has grown into a small enterprise, with dedicated staff, funding, and FAQs. The codebase is stable, mature, and well-commented. One notable aspect of the code is that it cross compiles on the following systems: OS_DOS, OS_MAC, OS_NT, OS_OS2, OS_UNIX, OS_UNIX_BSD, OS_UNIX_SUN, OS_UNIX_SYSV, OS_UNIX_IRIX, OS_UNIX_ULTRIX, OS_UNIX_OSF1, and OS_VMS. Some of these systems are no longer supported or shipping owing to the quick obsolescence of entire commercial operating systems. A consequence of this generic design is that all the basic IO functions and datatypes are wrapped to "NCBI corelib" versions: it may require some work for a programmer to adapt to the `corelib`. Companies that specialize solely in building computer systems, aka BioClusters, that run BLAST fast are profitable circa 2002.

What does BLAST *do*?

```
5003 r@hale ~/blastexec/db> blastall

blastall 2.2.5   arguments:

  -p  Program Name [String]
  -d  Database [String]
    default = nr
  -i  Query File [File In]
    default = stdin
  -e  Expectation value (E) [Real]
    default = 10.0
  -m  alignment view options:
(...)
```

So `blastall`, which is a front end for the different BLAST algorithms takes as command-line arguments (1) one of the BLAST variants {blastn, blastp, blastx, tblastn, tblastx}, (2) a database name, and (3) a query file. The common database files can be downloaded from the NCBI servers at ftp://ftp.ncbi.nih.gov/blast/db/. The nucleotide and protein sequence databases are updated by the sequencing and annotation projects daily, and most installations update their local images often.

Truly, how are these carbons, hydrogens, and nucleic acid sequences represented in computer memory and mass storage? They are mapped to ASCII characters, following a set of conventions known as the **FASTA** format description. This has nothing to do with the FastA (Fast Approximation) algorithm other than the name. FastA is an implementation of the Smith-Waterman string match algorithm with two heuristic steps to avoid excessive calls to the full algorithm at the cost of sensitivity. Section (2.4) contains the Smith-Waterman procedure citation.

For the 23 amino acid residues, which are run using BLASTP and TBLASTN, the supported codes are:

```
A  alanine                        P  proline
B  aspartate or asparagine        Q  glutamine
C  cystine                        R  arginine
D  aspartate                      S  serine
E  glutamate                      T  threonine
F  phenylalanine                  U  selenocysteine
G  glycine                        V  valine
H  histidine                      W  tryptophan
I  isoleucine                     Y  tyrosine
K  lysine                         Z  glutamate or glutamine
L  leucine                        X  any
```

```
M  methionine                    *  translation stop
N  asparagine                    -  gap of indeterminate length
```

with two problematic codes for the presence of gaps. 'X' means any single amino acid, and '-' means a gap of indeterminate length (more on gaps below).

The nucleic acid codes are

```
A --> adenosine           M --> A C (amino)
C --> cytidine            S --> G C (strong)
G --> guanine             W --> A T (weak)
T --> thymidine           B --> G T C
U --> uridine             D --> G A T
R --> G A (purine)        H --> A C T
Y --> T C (pyrimidine)    V --> G C A
K --> G T (keto)          N --> A G C T (any)
- --> gap of indeterminate length
```

with similar codes for gaps and for a single unknown nucleotide. 'N' represents any of {A,G,C,T}; and '-' means a gap of indeterminate length.

And despite the desire to avoid duplicating information that anyone could find in an introductory biology textbook, we duplicate here the translation table between the amino acid codes and the nucleic acids codes. We do so for the simple reason that the entries in the BLAST databases are often of proteins and are submitted as amino acids. For a program that uses these databases, it is necessary to re-encode the amino acids as nucleic acids to take advantage of the complex-plane encoding which is at the heart of our scheme, and the most recent schemes by Rajasekaran et al. (2001) and Katoh et al. (2002). This poses the very simple problem of choosing which of the several 3-letter DNA codes to use when re-encoding. For example, serine and leucine have as many as six equivalent and different DNA codes which could result in problematic results when re-encoded and run through a computer alignment scheme. There is no ambivalence going from DNA codes to amino acid codes however. If you're lost, nucleic acids make up RNA and DNA and amino acids make up proteins.

A very useful and often-searched database, the 'est_human' database uses only nucleic acids, and thus our translation problem is alleviated in that circumstance.

```
Amino Acid        FASTA          DNA Codes for each Amino Acid
                  Abbreviation
```

Alanine	A	GCA,GCC,GCG,GCT
Cysteine	C	TGC,TGT
Aspartic Acid	D	GAC,GAT
Glutamic Acid	E	GAA,GAG
Phenylalanine	F	TTC,TTT
Glycine	G	GGA,GGC,GGG,GGT
Histidine	H	CAC,CAT
Isoleucine	I	ATA,ATC,ATT
Lysine	K	AAA,AAG
Leucine	L	TTA,TTG,CTA,CTC,CTG,CTT
Methionine	M	ATG
Asparagine	N	AAC,AAT
Proline	P	CCA,CCC,CCG,CCT
Glutamine	Q	CAA,CAG
Arginine	R	CGA,CGC,CGG,CGT
Serine	S	TCA,TCC,TCG,TCT,AGC,AGT
Threonine	T	ACA,ACC,ACG,ACT
Valine	V	GTA,GTC,GTG,GTT
Tryptophan	W	TGG
Tyrosine	Y	TAC,TAT

Researchers who use sequence comparison schemes with highest scoring maximal pairs to do sequence homology comparisons can run into problems. Often it is necessary to have a thorough grasp of database searching and/or statistics to analyze the resulting scores definitively. For instance, when new proteins are sequenced often the best clues to their function come from comparing the proteins to other known protein sequences. One of the main motivations for the development of BLAST3 is the observation that with the HSPs and high scoring local alignments, false homologies occur with surprising frequency.

Altschul's article on protein database searches for multiple alignments demonstrates that high scoring alignments with false homologies occur with a surprising abundance. BLAST3 was designed to minimize the number of so-called false homologies by not just looking for alignment similarities, but for statistically-significant ones. There are many circumstances where high-scoring similarities are indicated for unrelated sequences: these could be minimized by incorporating a more powerful statistical framework with traditional techniques.

A researcher who uses only pairwise comparisons may miss biologically-significant relationships. For instance, when looking at a family of cytochromes, a researcher

Variant	Function
blastn	nucleotide query to a nucleotide database
blastp	protein query to a protein database
blastx	nucleotide (translated) query to a protein database
tblastn	protein query to a nucleotide (translated) database
tblastx	nucleotide (translated) query to a nucleotide (translated) database

Figure 2.2: **BLAST variants**

DB Name	Description	Location	Size
NT	Non-redundant GenBank+EMBL+DDBJ+PDB (no EST, STS, GSS, or TAGS sequences)	ftp.ncbi.nih.gov/blast/db/nt.Z	2.14 GB
NR	Non-redundant GenBank CDS translations+PDB+SwissProt+PIR	ftp.ncbi.nih.gov/blast/db/nr.Z	364 MB
est_human	Non-redundant GenBank+EMBL+DDBJ EST human sequences	ftp.ncbi.nih.gov/blast/db/est_human.Z	855 MB
SWISS-PROT	SwissProt extracted from the SP_NR	ftp.ncbi.nih.gov/blast/db/swissprot.Z	33.1 MB
PDB	Sequences from the 3-dimensional structure at Brookhaven Protein Data Bank	ftp.ncbi.nih.gov/blast/db/pdbaa.Z	3.6 MB

Figure 2.3: **Several BLAST-able databases**

may notice distantly related cytochromes among high-scoring sequences. He might then investigate to determine whether those regions show substantial overlap. It is this process that BLAST3 aims to automate (Altschul & Lipman, 1990, page 5513).

2.1.1 BLAST variants

There are many variants of BLAST that have been developed over the years. Some of the ones we know about are: NCBI BLAST, and Apple/Genentech BLAST on AltiVEC.

In May 2002, Apple Computer released a single rack space unit server computer product named Xserve. Clustered server computers and an operating system that is entirely GUI-based, as all of Apple's operating systems prior to OS X were, do not work in synergy. So, in a way the release of Xserve signalled Apple's foray into the world of computing clusters and increased its relevance with the scientific computing crowd. Before these two developments, many computer professionals had held Apple in disdain, barring their great astonishment at the friendly face of the Macintosh 128k in 1984.

```
5004 r@hale ~/blastexec/db> blastall -p blastn -d ./est_human
-i ../jahansprimer.txt
BLASTN 2.2.5 [Nov-16-2002]


Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer,
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997),
"Gapped BLAST and PSI-BLAST: a new generation of protein database search
programs",  Nucleic Acids Res. 25:3389-3402.

Query=
         (20 letters)

Database: est_human
           5,040,861 sequences; 2,613,540,672 total letters

Searching..................................................done

                                                                    Score  E
Sequences producing significant alignments:                         (bits) Value

gb|CA487488.1|CA487488 AGENCOURT_10808671 MAPcL Homo sapiens cDN... 40    0.008
gb|BU623275.1|BU623275 UI-H-FL1-bgc-g-01-0-UI.s1 NCI_CGAP_FL1 Ho... 40    0.008

>gb|CA487488.1|CA487488 AGENCOURT_10808671 MAPcL Homo sapiens cDNA clone
IMAGE:6718976 5'
          Length = 1049

 Score = 40.1 bits (20), Expect = 0.008
 Identities = 20/20 (100%)
 Strand = Plus / Plus


Query: 1   agaaggctggcactgtacga 20
           ||||||||||||||||||||
Sbjct: 483 agaaggctggcactgtacga 502


>gb|BU623275.1|BU623275 UI-H-FL1-bgc-g-01-0-UI.s1 NCI_CGAP_FL1
Homo sapiens cDNA clone
           UI-H-FL1-bgc-g-01-0-UI 3'
          Length = 723

 Score = 40.1 bits (20), Expect = 0.008
 Identities = 20/20 (100%)
 Strand = Plus / Minus


Query: 1   agaaggctggcactgtacga 20
           ||||||||||||||||||||
Sbjct: 621 agaaggctggcactgtacga 602
```

Figure 2.4: **Sample BLAST output**

```
5109 r@hale ~/blastexec/db> time blastall -p blastn -d ./est_human
-i ../jahansprimer.txt -o ../jahansprimeroutput.txt

9.212u 1.576s 3:22.24 5.3%      0+0k 0+0io 178629pf+0w
```

Figure 2.5: **Whole-genome BLAST timing.** In this trial we ran BLAST on the est_human database to correlate two primers. When searching the est database for primers one is trying to verify primer specificity. A pair of primers are used to amplify a certain gene or region of nucleic acid sequence. If the primer pair amplifies more than one region this can result in two regions of different lengths. The desired region might be separated by length using gel electrophoresis. Successfully BLASTing such a primer pair will help ensure that only the desired region is amplified, or will help correct possible problems later on. This sample run of a 20-base primer and a 3.0 GB database took 3 min 22 sec.

It became important to Apple to increase market share in hardware sales in compute-intensive organizations by releasing rack mount servers. The Biotech space has significant purchasing power, and one of its chief applications is BLAST. This lead to the development of the Apple/Genentech on AltiVEC version of BLAST. What this amounted to in many cases was replacing multiply and add operations by the vectorized AltiVEC'd versions of the same calls. AltiVEC-optimized applications can be straightforward to implement and often involve unrolling loops, migrating function calls, and the repeated running of profilers. Apple's profiler `CHUD` is particularly well-suited for marking sections of code that are taking up a lot of CPU time and that need optimizing.

The largest Xserve biocluster in the world was installed by The BioTeam of Boston, MA, at the biotech firm TLL/Singapore in Singapore. It is an 85-node system running Platform LSF (a grid clustering program) and is used as part of a high-thruput genome annotation pipeline.

2.1.2 PSI-BLAST and applications

PSI-BLAST is an extension to the original BLAST which allows gaps. It extends the original BLAST by using a two-hit method. The two-hit method determines statistically-significant alignments with a position-specific scoring matrix. By using

a dynamic programming algorithm it extends in both directions the central pair of aligned residues resulting from the initial hit scan. They set a window-length A and extend a segment only when two non-overlapping hits are found less than a distance A from one another.

Alignment-free sequence analysis is a topic that has considerable interest to researchers interested in phylogeny and inheritance, as well as to disinterested mathematicians and programmers. Molecular biology has been reduced to linear sequences of discrete units, much like the abstraction used by linguists for natural language. By using a linear string-based model and removing the physicality of these molecules and molecular networks, we lose the ability to analyze the undeniable 3D structure and complex functional relationships. Well established pair-wise sequence alignment tools query a database with a sequence pattern and return a similarity score and database locations. Protein folding is an example of this problem. Finding the similarity in tertiary structure (folding onto themselves) between two RNAs is in the class of NP-hard problems. NP is the class of problems which have solutions which can be quickly *checked* on a classical computer (e.g., checking a factorization) but for which the solutions cannot be quickly *determined* on a classical computer (e.g., performing the actual factoring). P is the class of computational problems that can be solved quickly on a classical computer (sorting a list, say). These classes are not so easily defined on quantum computers since, for instance, factoring can be performed as multiplication with a similar evaluation cost.

The researcher Iddo Friedberg has published several papers analyzing PSI-BLAST on such criteria as how sequence analysis can indicate protein structure (Friedberg et al., 2000; Friedberg & Margalit, 2002b,a). It is widely observed that structurally-similar proteins have very dissimilar sequence. This is partially due to the fact that sequence space is larger than structure space. In studying the "sequence-structure" relationship, Friedberg concentrated on mutually persistently conserved (MPC) positions in a stringently chosen set of protein samples exhibiting structural similarity and sequence dissimilarity. One of the chief results is that "residue" conservation despite sequence dissimilarity is essential to protein function. A "residue" in this context means an area of the protein exhibiting hydrophobicity, charge, or another well-defined structural property.

Conclusions from studies Friedberg references indicate that there is a small number of protein sequences which produce conserved structural residues across organisms and protein classes.

Genetic shuffling and recombination occurs which may not change the expressed phenotype and which may in fact cause phenotypic improvement. For example, Zhang et al. (2002) published results indicating that "genome shuffling leads to rapid improvement in bacteria." Alternatively, intron movement and recombination shows evidence for being an evolutionary process in population genetics (Lynch, 2002). There has also been work on sequence analysis tools which do not require pair-wise alignments. One recent example of such a program uses a "consensus database" and sequence tagging to do its comparisons (Christoffels et al., 2001).

Homologous recombinates can be created from a genotypic library of background strains. This alleviates the problem of creating varied genomic libraries by random mutagensis. If one has a probe, random mutagensis is not a very efficient way of targeting genes or gene products. Zambrowicz & Sands (2003) conducted a study of 5,000 "druggable" genes via mouse knockout. Of these 5,000 genes from the human genome, they determined approximately 100–150 high quality targets. Their study examined the role mouse knockout genetics might have on foreseeable drug target discoveries and on validation.

Gene knockout (KO) consists of selectively removing a single gene and observing the change in phenotype. "Knocking-out" the gene is not accomplished by finding primers which cut in the correct location, as one would do in preparation for PCR. By adding a mutagen to a colony of cells a body of genetic mutants result. Nuclear transfer entails injecting the nuclei of the mutant embryonic stem cells into a host organism's egg cell.

By taking DNA which is complementary to the coding strand one wants to knock out, one can produce a short specific probe. In practice this requires complete knowledge of the gene one wishes to modify. A mutagen is linked to each probe which allows the localization of the mutation. The specificity of an engineered probe to a mutagen can be determined if it has an observable chemical tag (fluorescent, radiative, etc.).

Before the completion of major sequencing projects, libraries and clones were

necessary for conducting gene-specific experiments. With a fuller library of expression sequence tags (ESTs), which annotate a full genomic sequence, one can in theory produce a knockout library of animals reflecting the phenotypes associated with each gene. Human knockout is not possible, because we can't experiment on live humans, but we can knock out human-like genes in other organisms. Scientists therefore conduct experiments on homologous genes in humans and mice. These homologs are found via comparative genomics, which is often accomplished via BLAST searches. Over evolutionary time, the differences in the genetic makeup of an organism and the location of the genes in a single organism change. Heredity can be studied by such applications of comparative genetics.

2.2 MAFFT

As we saw in Chapter 1, it is pretty easy to compute the location of matching sequence pairs using 1D FFT convolution. The problem with real-world sequence data is that there exist gaps, often of indeterminate length and content. By the FASTA conventions given above these gaps have specific codes. Gaps are inevitable because of the biological methodology used in determining sequences when high or complete coverage is necessary.

In addition to gaps, a scoring algorithm which gives an error estimate between the query-database pair is often essential for biologically-motivated analysis (see above for the PSI-BLAST scoring technique). A method for doing gapped multiple sequence alignments with the FFT called MAFFT (Multiple Alignment FFT) was proposed by Katoh et al. (2002). Theirs is a substantial effort, with a 100-fold reduction in time over previous multiple alignment schemes such as T-COFFEE and comparable accuracy to standard accuracy benchmarks like **ClustalW** (Thompson et al., 1994). Parts of MAFFT were in development for over five years, looking at the dates in their code. In this section we analyze some aspects of their paper's solution to the problems of scored gapped FFT alignment.

The main motivation for using FFT correlation in this context is to optimize the **Needleman and Wunsch** (NW) algorithm since it does not scale practically for hundreds or even dozens of sequences. Needleman and Wunsch scales very poorly

in CPU time, approaching $O(N^K)$ for K sequences of length N.

```
tbfast (aa) Version 3.82
generating 200PAM scoring matrix for amino acids ... done.
pre in align = pre
Constructing dendrogram ... done.
STEP 1 /35
group1 =  34
group2 =  35
FFT ... 4 segments found
```

Figure 2.6: **MAFFT group-to-group alignments.** MAFFT uses the FFT to find multiple segment pairings between groups of sequences.

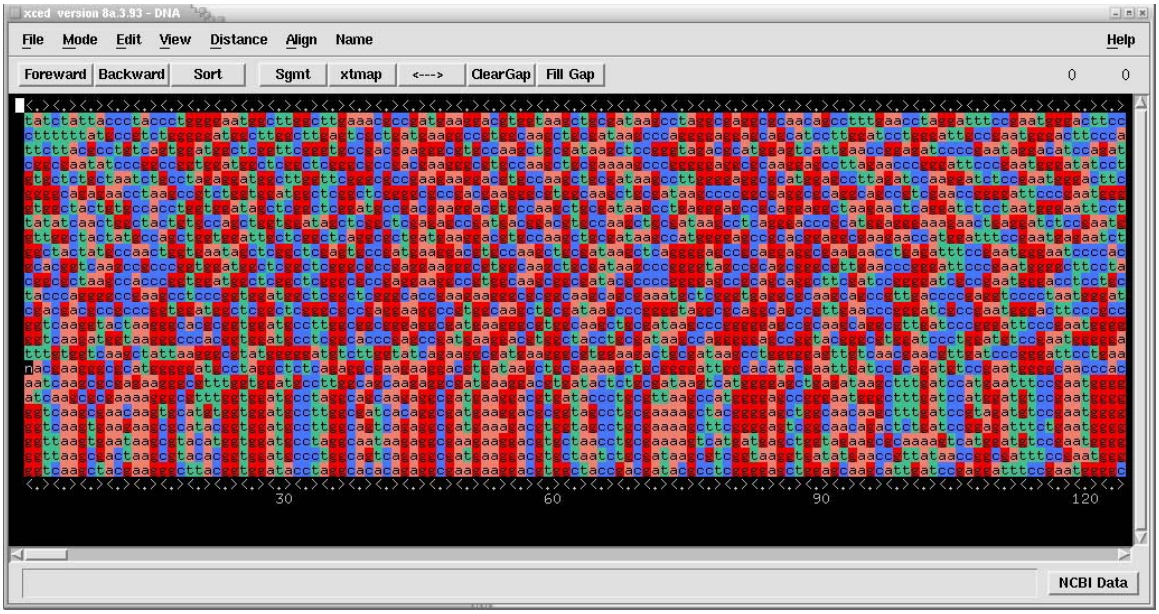

Figure 2.7: **MAFFT 25 sequences unaligned.** MAFFT provides an assortment of algorithms to perform multiple sequence alignment. Two heuristic FFT methods are developed: FFT-NS-2 is a progressive method; FFT-NS-i is an iterative refinement method.

2.2.1 Dynamic programming

MAFFT uses FFT convolution as inputs to a dynamic programming algorithm. The general idea behind dynamic programming is breaking up a large computational

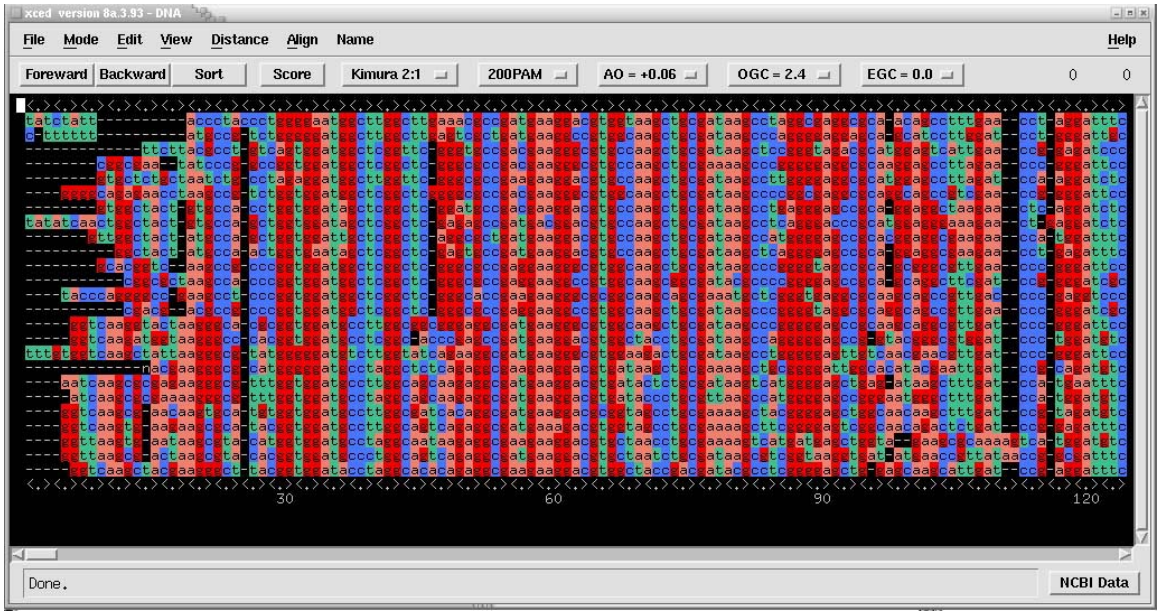

Figure 2.8: **MAFFT 25 sequences aligned.** MAFFT has inserted optimal gaps in these now-aligned nucleotide sequences. Letters are color-coded in their aligned columns.

problem into smaller problems then storing the answers to the smaller subproblems. The stored answers are then used to solve the larger problem. The Needleman and Wunsch algorithm is not the DP algorithm that is most often used in bioinformatics applications, as the Smith-Waterman procedure, for one, supersedes it in efficiency.

There are two aspects which characterize problems for which a dynamic programming algorithm may produce an optimal solution. One, the problem must have optimal substructure. That is, optimal solutions to subproblems must exist. Thus the dynamic programming algorithm iterates on subproblem instances. Two, the subproblems must be overlapping. The recursive algorithm revisits the same problem over and over again. This is very different from a divide-and-conquer scheme. The overlapping subproblem stipulation obviously has implications for many sequence alignment problems. Overlapping high-quality sequence is the most useful data for genome sequence assembly programs such as GigAssembler.

When the algorithm solves the subproblems, it stores them in an array. So let a 2D array $subSolution(i, j)$ maintain the solutions to the problems that have been solved. When the recursive function computes another solution, it fills in the appropriate entry in the table. After proceeding through several iterations on

subproblems however, it begins to re-visit problems it has already solved, and so it returns the solution immediately. As the array $subSolution(i, j)$ fills up, the larger problem nears its solution. Since by definition the optimal solutions to the subproblems must exist, if the algorithm succeeds minimally, it succeeds in the large. The direct analogy to sequence algorithms is the subproblem of pair-wise matches and the larger problem is global alignment with optimal gaps.

Dynamic programming is very good at finding optimal alignments between two sequences. However, it fails when comparing more than two sequences simply enough because the computational time is proportional to $O(N^K)$, where N is the length of the string or sequence fragment, and K is the number of sequences. It is apparent this is prohibitively costly for even as few as three sequences, when compared to FFT convolution, which requires only $O(N \log N)$.

Dynamic programming has a very close relation to *edit distance*, the number of iterative steps needed for a particular algorithm to "replace" one string by another. This is only interesting of course if the `replace`(*string* X by *string* Y) operation is significantly more costly than an individual iterative step.

2.2.2 FFT application to group-to-group alignment

Katoh et al. (2002) observed that group-to-group alignments are also possible using more or less generic FFT correlation.

For reasons relating to "protein residue substitution frequencies," see (Grantham, 1974), Katoh et al. (2002) formulate the FFT correlation for an amino acid a in terms of (1) the volume value $v(a)$ and (2) the polarity value $p(a)$. As a result, the correlation of the volume component $c_v(k)$ is

$$c_v(k) = \sum_{1 \le n \le N, 1 \le n+k \le M} \hat{v}_1(n)\hat{v}_2(n+k). \tag{2.1}$$

And the correlation of the polarity component $c_p(k)$ is

$$c_p(k) = \sum_{1 \le n \le N, 1 \le n+k \le M} \hat{p}_1(n)\hat{p}_2(n+k). \tag{2.2}$$

These equations are functionally equivalent to Equation (1.15).

One can consider these two equations as special cases with one sequence in each group. So the extension from sequence-to-sequence to group-to-group alignment is done by replacing $\hat{v}_1(n)$ by $\hat{v}_{group1}(n)$. This is a linear combination of volume components belonging to $group1$. Thus Equations (2.1) and (2.2) now become:

$$\hat{v}_{group1}(n) = \sum_{i \in group1} w_i \cdot \hat{v}_i(n) \tag{2.3}$$

and

$$\hat{p}_{group1}(n) = \sum_{i \in group1} w_i \cdot \hat{p}_i(n). \tag{2.4}$$

w_i is the weighting factor for sequence i calculated via the ClustalW method.

This group alignment idea would be useful for converting a sequence to a sequence of four-dimensional vectors whose components are frequencies of nucleotides $\{A, C, G, T\}$, instead of volume and polarity values.

2.2.3 FFT scoring and gap penalty

A similarity matrix $H(i, j)$ between two amino acid sequences $A(i)$ and $B(j)$ is constructed by setting $H(i, j) = \hat{M}_{A(i)B(j)}$. i and j are positions in the two sequences.

The normalized similarity matrix $\hat{M}_{ab}$ for amino acids a and b is

$$\hat{M}_{ab} = [(M_{ab} - average2)/(average1 - average2)] + S^a. \tag{2.5}$$

The two average values are defined as: $average1 = \sum_a f_a M_{aa}$ and $average1 = \sum_b f_b M_{bb}$. f_a is the frequency of the occurrence of amino acid a and S^a is a parameter that functions as a gap penalty. A value for the parameter S^a is 0.06.

Then leveraging the group-to-group alignment discussion above, when two groups of sequences are aligned, the homology matrix between group1 and group2 is calculated as:

$$H(i, j) = \sum_{n \in \text{group1}} \sum_{m \in \text{group2}} w_n w_m \hat{M}_{A(n,i)B(m,j)}. \tag{2.6}$$

$A(n, i)$ is the i-th site of the n-th sequence in group1; similarly, $B(m, j)$ is the j-th site of the m-th sequence in group2; w_n is defined as above.

The NW algorithm has factors for gaps, which we state here, these will go into

the value for the accumulated score for the optimal path:

$$G_1(i,x) = S^{op} \cdot \{1 - [g_1^{start}(x) + g_1^{end}(i)]/2\}, \tag{2.7}$$

and likewise for $G_2(j,y)$. Here $g_1^{start}(x)$ is the number of gaps that start at the x-th site and $g_1^{end}(i)$ is the number of gaps starting at the i-th site:

$$g_1^{start}(x) = \sum_{m \in \text{group1}} w_m \cdot a_m(x) \cdot z_m(x+1), \tag{2.8}$$

$$g_1^{end}(i) = \sum_{m \in \text{group1}} w_m \cdot z_m(i-1) \cdot a_m(i). \tag{2.9}$$

The two parameters to these gap penalties, z_m and a_m, are: if the i-th site of sequence m is a gap, then $z_m(i) = 1$ and $a_m(i) = 0$; if it is not a gap, then $z_m(i) = 0$ and $a_m(i) = 1$. Two competing methods for calculating the gapped score are given in (Thompson et al., 1994) and (Gotoh, 1993).

The recursive equation for NW for the optimal alignment is:

$$P(i,j) = H(i,j) + max \begin{cases} P(i-1,j-1) \\ P(x,j-1) - G_1(i,x) \quad (1 \le x < i-1) \\ P(i-1,y) - G_2(j,y) \quad (1 \le y < j-1). \end{cases} \tag{2.10}$$

This is the accumulated score for the optimal path from $(1,1)$ to (i,j). So in fact it appears as though Katoh et al. have not developed any novel gap scoring technique. They are using the NW score with some simplifications. Specifically one simplification is only using the w_m measures for the weights from ClustalW. The complexity is of course in scoring the gap penalty: the recursion in Equation (2.10) is determined entirely by the gap penalties G_1 and G_1. And from Equations (2.8) and (2.9), with the much simplified z_m and a_m, a great deal of complexity is spurned, presumably in the name of speed.

2.3 Alignment, substitution matrices, and sequencing

As formal and hopefully useful definitions for some standard concepts like gaps and local alignment haven't been given yet, we state here some definitions.

Definition 2.3.1. A **gap** is any *maximal, consecutive run* of spaces in a *single* string of a given alignment.

Definition 2.3.2. We define an **alignment** of two or more sequences intuitively. An *alignment* with *offset* n of a pair of alphabetic sequences S_1 and S_2 is a pairing of letters of the sequences in which the i-th letter of S_1 is paired with the $(i + n)$-th of S_2. For a particular alignment, a *match* occurs if and only if corresponding letters are identical.

Definition 2.3.3. The **local alignment** of two strings S_1 and S_2, is given by substrings α and β of S_1 and S_2, whose similarity, i.e. optimal global alignment value, is maximal for all pairs of substrings from S_1 and S_2.

Definition 2.3.4. Local similarity is a measure of relatively conserved subsequences.

Definition 2.3.5. Global alignment determines the overall alignment of two sequences, and may contain large stretches of low similarity.

MAFFT (Section (2.2)) is a global alignment program. It finds alignments between a set of protein sequences of similar length.

Definition 2.3.6. Global similarity determines the overall similarity of two sequences, and may contain large stretches of low similarity.

Definition 2.3.7. Homology means exhibiting similarity in characteristics resulting from common ancestry. However researchers doing sequence alignment may sometimes use "sequence homology" interchangeably with "sequence similarity." Similarly, two *homologous proteins* are proteins related to each other by a common ancestor in their evolutionary histories.

2.3.1 PAM matrices

The acronym PAM has two meanings in the literature: "point accepted mutation" and "percent accepted mutations."

Definition 2.3.8 (PAM unit). Two sequences S_1 and S_2 are defined to be one **PAM unit** divergent if a series of accepted point mutations, without insertions or deletions, have converted sequence S_1 to S_2. For one PAM unit of divergence, there can be no more than one mutation per one-hundred amino acids.

An accepted mutation is one which when incorporated into the protein doesn't result in changed phenotype or functional structure. If it does result in changed phenotype, the change must be at least nonlethal.

Several remarks should be made on interpreting PAM units. Firstly, one PAM unit divergence between two sequences does not mean that they differ by one percent. For two aligned sequences—recall that there are no insertions or deletions—that have a certain score, they could well be different by *less* than their score indicates. The point mutation at a certain location can mutate back to its original state. For sequences with very large PAM scores, 200–300 PAM units, one still expects the sequences to be identical in most positions.

Secondly, the actual calculation of the PAM number is subject to problematic biological constraints. Ideally the sequence of an extant protein is compared with a genetic ancestor and the number of point mutations counted. Acquiring ancestor protein and sequence is often difficult, since in a sense all we have currently are the extant organisms. One also can't be sure that only accepted mutations have occurred: in protein evolution, insertions and deletions occur but are not distinguishable and the correct correspondence between sequence positions may be impossible to determine. Finding true historical gaps is very difficult especially in circumstances with large numbers of PAM units.

A technique that leverages the molecular clock theory can help to make calculations between ancestral proteins. Suppose two sequences S_i and S_j can be shown to have diverged from a common ancestor S_{ij}. Following the molecular clock theory, the expected PAM unit distance between S_{ij} and S_i equals the expected PAM unit distance between S_{ij} and S_j. Thus, the difference in the PAM scores between

S_{ij} and its descendants S_i and S_j can be taken to be half the number of differences between S_i and S_j. However this is mostly a theoretical guideline, since one can seldom assume that amino acid mutations are (1) reversible and (2) uniformly bi-directional—equally likely to occur in both inherited and derived directions.

The funny thing about PAM matrices is that they group sequences with *the same* PAM unit score into a single matrix. To form an ideal PAM matrix one collects a set of pairs of homologous sequences which have the same PAM score n. Next, each pair of sequences is aligned and for each amino acid pair A_i, A_j, the number of times A_i matches opposite A_j is counted. That number is then divided by the total number of pairs in all the aligned data. This quotient gives the frequency $f(i, j)$, and f_i and f_j are the frequencies that the amino acids occur in the sequences. So f_i is the number of times A_i appears in all the sequences divided by the total lengths of the sequences. For the ideal PAM matrix values, one divides the true replacement history $f(i, j)$ by the replacement history due to chance, $f(i)f(j)$. The log-odds ratio for the (i, j)-th PAM entry is therefore:

$$\text{PAM}(i, j) \equiv \log \frac{f(i, j)}{f(i)f(j)}. \tag{2.11}$$

The logarithm is taken because when PAM matrices were first being used in 1978 by Dayhoff et al. (1978), it was important that addition is easier than multiplication.

PAM matrices have proven very effective for finding alignments between biologically-significant protein sequences. In fact, the PAM250 matrix is often considered the "canonical" protein scoring matrix. The PAM250 matrix is a matrix which has homologs which diverge by 250 PAM units, even after an scheme to introduce optimal gaps has been applied. There is no analytic way of characterizing the effectiveness of PAM matrix scoring, in part because it is so difficult to quantify biological significance beyond simply knowing a lot about a certain set of proteins. Time and research experience have proven PAM's usefulness, though another substitution matrix, BLOSUM62, is being increasingly used (Henikoff & Henikoff, 1992). The BLOSUM62 matrix scheme emphasizes scoring conserved substrings and a database of known proteins. Such a database cataloguing protein structure and functional domains is PROSITE http://www.expasy.ch/prosite/. Position-specific scoring matrices (PSSMs) which we describe in Section (3.1.1) have much in common with

PAM scoring matrices.

2.3.2 Sequencing and gap distribution

As we saw in Section (2.1), the online nucleotide and protein sequence databases use gap characters to represent gaps of indeterminate length. Some of these gaps result from errors accrued during the process of sequencing and assembly. For example, when using the shotgun sequencing technique on a genome like the human genome, with huge areas of repeated sequence, it becomes impossible to use bacterial or yeast artificial chromosomes to make the clone libraries. The level of repeats exceeds the threshold for either structural stability or the capacity of the translation and transcription machines to translate and transcribe sequences with very long repeat regions.

Lander & the International Human Genome Sequencing Consortium (2001) details the steps of hierarchical shotgun sequencing:

$$\begin{aligned}&\text{Genomic DNA} \rightarrow \text{BAC library} \rightarrow \text{Organized mapped large clone contigs}\\ &\rightarrow \text{BAC to be sequenced} \rightarrow \text{Shotgun clones} \rightarrow \text{Shotgun sequence} \rightarrow \text{Assembly.}\end{aligned} \tag{2.12}$$

So the BAC library is constructed by fragmenting the original genome and cloning it into large-fragment cloning vectors. The genomic DNA fragments in the BAC clones are then organized into a physical map (often with the aid of fingerprint scaffolding). Individual BAC clones are then selected and sequenced by an automated process using the random shotgun method. After the BACs are sequenced, the sequences are assembled reconstructing the sequence of the genome.

This last assembly step was accomplished for the HGP by a program written by Jim Kent at the University of California at Santa Cruz called GigAssembler (Kent & Haussler, 2001). GigAssembler was run on the first "freeze" of the HGP sequence data on May 24, 2000. For political reasons having to do with the simultaneous publication of the private Celera sequence data and the public Human Genome Project data, Kent had little more than one month to write his program and set it to work on several years worth of data. The program uses the full assortment of

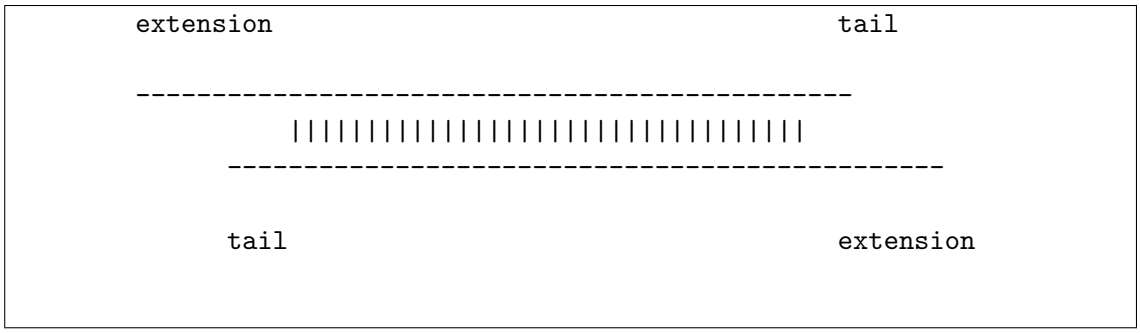

Figure 2.9: **Sequence with low-quality tails.**

collected data: the sequence contig, fingerprint map, mRNA, EST, and BAC end data.

The design of GigAssembler is interesting in the sense that it incorporates a wide range of biological data, must take care of problems in this data, and that it incorporates a variety of bioinformatics computer techniques. The fact that it works, i.e. can assemble sequence with low error, from HGP data is also an unmistakable indicator that the hierarchical human genome sequencing paradigm developed between Washington University, MIT Whitehead, and the Sanger Centre is sound. That it is sound, via its basis on hierarchical shotgunning, and not whole-genome shotgunning, is important for many reasons. For one, Celera Genomics used the HGP's fingerprint contigs to assemble their own WGS-derived sequence. To give an exposition of the full assembly method would be unnecessarily tedious, but a short description may prove enlightening.

There are eight algorithmic steps to the program:

1. Overlapping initial sequence contigs are merged into "rafts." These sequence contigs come from one of two places—they are either constructed from finished sequence or are from accessions to the draft clone library. This merge may be as simple as aligning the ends of overlapping contigs, but for many initial sequence contigs the ends have data of very low quality. These low-quality ends are discarded and regarded as non-aligned "tails."

2. Sequenced clone contigs called "barges" are built from overlapping clones.[3]

[3] "Barges are called barges because they are similar to rafts, but are built from larger pieces" (Kent & Haussler, 2001, page 1544).

3. Default coordinates are assigned to the initial sequence contigs. The coordinates are based on the barge offset the contig is in plus the start position of the accession number for the clone.

4. Build the directed raft-ordering graph. This is a graph with two node types: the rafts and the sequenced clone end points.

5. Connect the rafts in the ordering graph by the addition of mRNA, EST, BAC end pairs, and additional ordering information. A "bridge" is built out of alignments of initial sequence contigs with the other additional data. There are two scoring parameters for bridges: the type of information, for example mRNA has a very high score, and the strength of the underlying raft, based on length of the alignment minus tails and extensions. See Figure (2.9).

6. Order the rafts by walking the bridges in order of the default coordinates.

7. After walking the bridges, a sequence path along the rafts is built. The following steps are very important: (1) Find the longest most finished initial sequence contig passing through each section of the raft. (2) Put the best initial sequence contig from the first part of the raft from step (1) into the sequence path. (3) Find an alignment between the best sequence contig from the first and second parts of the raft. (4) Repeatedly extend the sequence path until the end of the raft is reached.

8. Build the final sequence for the fingerprint clone contigs by inserting a number of Ns between raft sequence paths. 100 Ns are inserted between rafts from the same barge, $50,000$ Ns between bridged barges, and $100,000$ Ns between unbridged barges.

Interestingly, gaps are present in introns and sequence for expressed genes. As an example of this scenario, we tried to determine the ratio of 'gap' characters to coding characters in the est_human and nr databases. This would give a sense of the relative numbers of gaps for some protein sequence. However, after removing the FASTA header from the nr database, there are *no* gap characters whatsoever. In est_human, there also appear to be no gap characters, but there is a difficulty when manipulating files larger than the 2GB file size limit.

Gaps in genomic DNA occur for various reasons. Insertion or deletion of an entire subsection of DNA may occur as a result of a single mutational event. The fact that there are biological applications which require gaps is due to extant phenomena in biology which cause "gaps." Retroviruses insert their own genetic code into their hosts, thereby inserting a gap into the original contiguity of the host genome. Translocations occur between chromosomes. During meiosis, genes may cross-over unequally, causing insertion in one strand of a chromosome, and a deletion in the other. Transposable elements, or "jumping genes," may be found in some organisms. DNA slippage occurs if during replication the machinery loses its place on the template strand, slipping backwards causing sections to be repeated and insertion/deletion gaps to form.

The role of gaps in alignments has a natural analogy in the case of complementary DNA (cDNA). cDNA is complementary to mRNA and contains only concatenations of exons. Thus the cDNA alignment and matching problem consists of trying to match cDNA with the DNA it came from, while the DNA is full of introns, which are the gaps we must account for.

The procedure known as "finishing" attempts to remedy the errors caused by the shotgunning and reassembly of large genomes by systematically comparing the output to fingerprint contigs and BAC clones (the scaffolding). Special consideration is often necessary to produce BAC clones with overlaps on both ends, called BAC-end clones, or BAC-end sequences. Scaffolding is essential to sequencing genomes with areas of repeated sequence and helps prevent the misassembly of fragments even when using the best available computer assembly tools. The two contrasting methods are often called the *hierarchical shotgun (HS) assembly* and the *whole-genome shotgun (WGS) assembly*. These two approaches were used by two big-gun sequencing operations: the Human Genome Project used HS, and Celera Genomics used WGS. In the HS assembly, the entire genome is first broken down into a set of intermediate clones called bacterial artificial chromosomes (BACs).

The article by Waterston et al. (2002) presents one of the most concise treatments of the problems of WGS versus HS. Sir John Sulston was one of the co-authors, and as former head of the Sanger Center, he had a vested interest in refuting Celera's claims that the WGS approach would work for genomes sizes of order 10^9 base pairs.

Lander and Waterson's article (Lander & Waterman, 1988) presents a mathematical analysis of the expected number of gaps as a function of coverage. This is fairly straightforward and assumes that fingerprint clone contigs are present in a geometric distribution.

Let us begin by defining four main variables used in this model for predicting the number of gaps in a sequencing operation for which coverage is not total:

G haploid genome length in base pairs (bp);

L clone or read length in bp;

N number of clones fingerprinted; [4]

T amount of overlap need for detection of overlap.

Several values are based on these empirically determined numbers:

$c \equiv \frac{LN}{G}$ the coverage;
$\alpha \equiv \frac{N}{G}$ the probability of starting a new clone, per bp;
$\theta \equiv \frac{T}{L}$ fraction of clone overlap needed for detections.
$\sigma \equiv 1 - \theta$ "useful" read length less overlap

To find the expected number of clones in a contig, recall the geometric distribution $P(k) = p(1 - p)^k$ or given a random variable X, with probability function

$$f(X) = (1 - \pi)\pi^X \qquad \text{for } X = 0, 1, 2, \ldots$$

is referred to as the geometric distribution with parameter π. It follows that the expectation value for a contig k with parameter p is:

$$\sum_{k=o}^{\infty} kp(1-p)^k = EX(k) = \frac{1-p}{p}. \tag{2.13}$$

This simplifies since

$$\sum_{k=o}^{\infty} p(1-p)^k = 1$$

[4] After making restriction digests, overlapping areas can be inferred. Fingerprint clone contigs are produced by joining these inferred overlapping clones. Small fingerprint clone contigs are useful in sequencing projects and have been used to identify non-redundant clones and to seed sequencing in new regions. A "contig" is the term for one or more overlapping clones.

to $EX(k) = 1 + \frac{1-e^{-c\sigma}}{e^{-c\sigma}}$, which means the expected number of clones fingerprinted is $N = e^{c\sigma}$, where c and σ are defined above.

The expected length of the contig is

$$L[(e^{c\sigma} - 1)/c + 1] = L + \frac{L}{c}(e^{c\sigma} - 1).$$

Before genetic sequencing operations expanded to the capacity and level of automation necessary for projects like the HGP, scientists considered physical mapping, that is genomic mapping, an important part of genetic analysis. The physical mapping of several organisms, *E. coli* and *Saccharomyces cerevisiae*, brought to the attention of Lander & Waterman (1988) the mathematical problems of genetic mapping by fingerprint clones. To find overlaps between clones, a large number of clones were chosen at random from a library of recombinant clones and those clones with sufficiently similar fingerprints were inferred to have overlapping sequence.

2.4 Classical string search and alignment algorithms

There are three widely-used and often-cited string matching and alignment algorithms both in the computer science community and in the biology community. Two of the papers explain methods for string-string and multiple alignment: Needleman & Wunsch (1970) is rarely used and runs in cubic time; Smith & Waterman (1981) runs in quadratic time. The term "Needleman-Wunsch" is often used to state *the problem* of global alignment, not a solution. The third gives an optimal string search algorithm: Knuth et al. (1977).

We will describe briefly the "find-the-location-of-a-substring functionality" which is neither new, nor necessarily complex. In fact it is very easy. There is a C library function `strstr(char *s1, char *s2)` which does this, and a simple implementation is given here. It does exactly what you'd imagine a naïve method would: it checks to see if the search string is of zero length; then it looks through each letter of the strings for matches until either the end of the search string or a mismatch occurs, returning the location of the search string in the database string on success.

However, the naïve method is just that, naïve, and computer scientists have put considerable effort into finding optimal string search algorithms.

```
#undef strstr
/*
 * find first occurrence of s2[] in s1[]
 */
char *(strstr) (const char *s1, const char *s2) {
    if ( *s2 == '\0' )
        return ( (char *)s1 );
    for (; ( s1 = strchr( s1, *s2 )) != NULL; ++s1 ){
        const char *sc1, *sc2;
        for ( sc1 = s1, sc2 = s2; ; )
            if ( *++sc2 == '\0' )
                return ( (char *)s1 );
            else if  ( *++sc1 != *sc2 )
                break;
    }
    return (NULL);
}
```

As Gusfield points out, "The day may well come when sequence database search will just involve the repeated application of precise two-string alignments. But that day is not yet here" (Gusfield, 1997, page 376). This emphasizes a core concept behind any BLAST-like system—BLAST does not perform precise two-string alignments. It scores alignments across a database using a look-up table, precisely because it is not feasible to repeatedly apply an alignment algorithm. Very different methodologies are used when solving global alignment and local alignment. Global alignments can be optimally computed via dynamic programming, for instance when performing alignments between more than two strings. There is no doubt a simple "string-substring" method works for a single application of local alignment between two sequences, but the day when it can execute a broad sequence database search is not yet here.

2.5 Mathematical models for genomics

2.5.1 Hidden Markov models

Definition 2.5.1. A **stochastic process** is a process in which (1) each element of a set is classified in a distinct state among several fixed states and (2) that these

elements can switch among states over time.

Definition 2.5.2. A **hidden Markov model** (HMM) is a doubly-stochastic process that is not observable (it is hidden), and that can only be observed through another set of stochastic processes that produce the sequence of observed symbols. An HMM consists of a model λ and three parameters:

1. A = state transition probability distribution,

2. B = observation symbol probability distribution in state j,

3. π = the initial state distribution.

Thus,

$$\lambda = (A, B, \pi) \text{ and is termed an HMM.}$$

The outcome of the model is a set of observable symbols O—in our case it is convenient to consider them characters from a finite alphabet $\Sigma = \{A, G, C, T, N\}$. It is also useful to stipulate some signal processing considerations, as this is signal processing. Observables can be either time-varying or time-invariant; deterministic or stochastic. Consider temporal variation in speech processing since it is in general a simple linear time-invariant system. There is a given sequence of symbols, or sounds, and each symbol is a linear system representing a short segment of the process. In the analysis therefore one develops a common short time model for each steady part of the signal. This re-iterates some of the ideas from Section (1.2).

HMMs have been developed to address three distinct problems: (1) How to identify steady or distinct periods in the signal (2) How to characterize the sequentially-evolving nature of these periods (3) What common short time model should be chosen for the distinct periods. In designing a suitable model one must designate a size T as the number of states in the model. Also one must choose model parameters, i.e. state transition probabilities, to optimize the model so that it best explains the observed outcome sequence.

To formalize these three problems, as they are addressed in real-world applications of HMMs, we identify the following three problems for a model $\lambda = (A, B, \pi)$:

Problem 1 Given a set of observations $O = O_1, O_2, \ldots, O_T$, how do we compute $\Pr(\mathrm{O}|\lambda)$, the probability of the observation sequence—that is the *score* produced by the model.

Problem 2 Given a set of observations $O = O_1, O_2, \ldots, O_T$, at hidden level 1 of the doubly-stochastic model, how do we choose a state sequence $I = i_1, i_2, \ldots, i_T$ which is optimal for some chosen criterion.

Problem 3 How do we adjust the model parameters A, B, π in the model to maximize $\Pr(\mathrm{O}|\lambda)$.

For **Problem 1** we have the following two equations:

$$\Pr(\mathrm{O}|I, \lambda) = b_{i_1}(O_1)b_{i_2}(O_2)\cdots b_{i_T}(O_T)$$

is the probability of the observation sequence O for each fixed state sequence $I = i_1, i_2, \ldots, i_T$. And

$$\Pr(I|\lambda) = \pi_{i_1} a_{i_1 i_2} a_{i_2 i_3} \cdots a_{i_{T-1} i_T}$$

is the probability of the state sequence $I = i_1, i_2, \ldots, i_T$. Thus the product of these two is simply the probability that O and I occur simultaneously, the joint probability.

$$\begin{aligned} \Pr(\mathrm{O}|\lambda) &= \sum_{\text{all } i} \Pr(\mathrm{O}|I, \lambda)\Pr(I|\lambda) \\ &= \sum_{i_1, i_2, \ldots, i_T} \pi_{i_1} b_{i_1}(O_1) a_{i_1 i_2} b_{i_2}(O_2) \cdots a_{i_{T-1} i_T} b_{i_T}(O_T). \end{aligned} \tag{2.14}$$

Equation (2.14) gives the probability of observation O over all states $i_1, i_2, \ldots, i_T$. The transition to state i_2 occurs with probability $a_{i_1 i_2}$. A $b_{i_1}(O_1)$ term is the probability of generating a symbol O_i. Each step in the summation calculation represents the probability of a transition; so products under the summand continue until the last transition from state i_{T-1} to i_T, generating symbol O_T with probability $b_{i_T}(O_T)$.

Even the casual reader will remark that this is computationally infeasible. For each time $i_t\big|_{1 \le t \le T}$ there are N possible states, and the products in the summation

require at least $2T$ operations. This gives a rough asymptotic upper bound of $O(2T \cdot N^T)$ operations. Plugging in some small values, say, $N = 5$ and $T = 100$, yields $2 \cdot 100 \cdot 5^{100} \approx 10^{72}$ computations. However, there are only 10^{87} elementary particles in the universe, and those are mostly photons, so in practice $\Pr(\mathrm{O}|\lambda)$ is calculated using the forward-backward procedure which requires only $O(N^2T)$ calculations.

Briefly, the **forward-backward procedure** can be explained as follows. There are two variables α, β, the forward and backward variables, respectively.

$$\alpha_t(i) = \Pr(O_1, O_2, \ldots, O_t, i_t = q_i|\lambda)$$

is the probability until time t of the partial observation sequence O and state q_i at time t. And

$$\beta_t(i) = \Pr(O_{t+1}, O_{t+2}, \ldots, O_T, |i_t = q_i, \lambda)$$

is the probability of the partial observation sequence from $t+1$ to T given state q_i at time t. And we relate it to (2.14) as:

Step 1 $\alpha_1(i) = \pi_i b_i(O_1), \qquad 1 \leq i \leq N$

Step 2 $\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i)a_{ij}\right] b_j(O_{t+1}), \qquad t = 1, 2, \ldots, T-1 \qquad 1 \leq j \leq N$

Step 3 $\Pr(\mathrm{O}|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$.

Description: Step 1 initializes forward probabilities. In Step 2 the product $\alpha_t(i)a_{ij}$ is the probability of the joint event of $O_1, O_2, \ldots, O_t$ observed and state q_j reached at time $t+1$ via q_i at time t. Summing the product over N states of q_i gives us the probability of q_j at the time $t+1$ with the previous partial observations, α_t. Then, $\alpha_{t+1}(j)$ is multiplied by the probability $b_j(O_{t+1})$.

Finally, since in Step 2 the range is from $t = 1, 2, \ldots, T-1$ there is a *terminal forward variable* $\alpha_T(i)$ which gives the resulting probability $\Pr(\mathrm{O}|\lambda)$, for $i_T = q_i$ and model λ.

Determining the asymptotically-tight upper bound on the time complexity indicates the necessary range covered, $1 \leq t \leq T$, $1 \leq j \leq N$ in determining the $\alpha_t(j)$ multiplied with another N elements for the summation in Step 3 yields $O(N^2T)$.

The backward variable $\beta_t(i)$ is calculated via a similar 2-step procedure replacing the equations in steps 1–2 above by the equation

$$\beta_t(i) = \sum_{i=1}^{N} a_{ij} b_j(O_{t+1}),$$

where

Step 1 $\beta_T(i) = 1, \qquad 1 \leq i \leq N$

Step 2 $t = T-1, T-2, \ldots, 1, \qquad 1 \leq i \leq N.$

Thus, $\beta_T(i)$ is set to 1 for all i. And Step 2 is best illustrated by Figure (2.10). In order to be in state q_i at time t you have to have been in q_j at time $t+1$, with probability $\beta_{t+1}(j)$, and having made a transition with probability a_{ij} from state q_i to q_j. End forward-backward procedure, onwards.

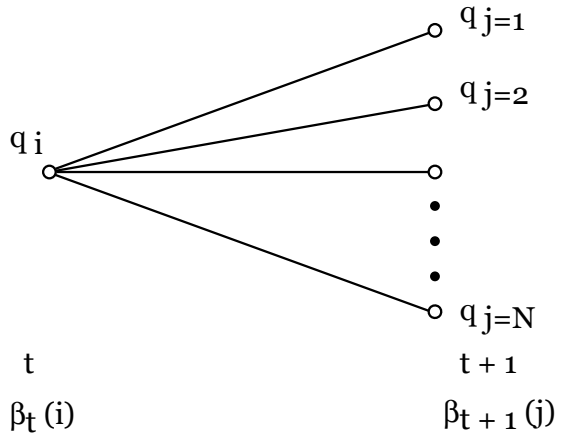

Figure 2.10: **HMM backward variable.**

Problem 2 At the hidden level 1 observation state sequence, $O_1, O_2, \ldots, O_T$, how do you choose a state sequence $I = i_1, i_2, \ldots, i_T$ which is optimal in a meaningful sense? That is, what are meaningful optimality criteria? In this example we want to choose the states i_t which are individually most likely.

Let,

$$\gamma_t(i) = \Pr(i_t = q_i | O, \lambda)$$

be the probability of state q_i at time t, for an observation sequence O and model λ. Recalling the forward-backward procedure, $\gamma_t(i)$ can be re-written as

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\Pr(\mathrm{O}|\lambda)}.$$

Since $\alpha_t(i)$ expresses all the partial observation probabilities of states $O_1, O_2, \ldots, O_t$ and $\beta_t(i)$ of observed states $O_{t+1}, O_{t+2}, \ldots, O_T$. The normalization factor $\Pr(\mathrm{O}|\lambda)$ simply assures that for

$$\sum_{i=1}^{N} \frac{\Pr(O_1, \ldots, O_t|\lambda)\Pr(O_{t+1}, \ldots, O_T|\lambda)}{\Pr(\mathrm{O}|\lambda)} = \sum_{i=1}^{N} \gamma_t(i) = 1, \tag{2.15}$$

since the sum of the probabilities of the states divided by the probability of the observable O must be 1.

To find the individually most likely state, i_t, take

$$i_t = \mathrm{Max}(\gamma_t(i)), \qquad 1 \leq i \leq N \qquad 1 \leq t \leq T \tag{2.16}$$

the single maximum probability of state q_i at time t. This trivial technique for finding the optimal state sequence is superseded in practice by the Viterbi algorithm (Forney, 1974). The Viterbi algorithm is similar to a Kalman filter in that they both track stochastic processes with pseudo-optimum recursive methods, the one of Markov models, the other of Gaussians.

Problem 3 How can one adjust the model parameters $\lambda = (A, B, \pi)$ to maximize $\Pr(\mathrm{O}|\lambda)$? This problem is essentially how to optimize model parameters to best match a training sequence. This involves a re-estimation step on the model λ. Call $\bar{\lambda}$ the re-estimation model. By iteratively replacing λ by $\bar{\lambda}$ until a limiting factor has been reached, we increase the probability for observing outcome O.

One way to do this is by defining a re-estimation parameter $\xi_t(i, j)$:

$$\xi_t(i, j) = \Pr(i_t = q_i, i_{t+1} = q_j|O, \lambda), \tag{2.17}$$

the probability of being in state q_i at time t making a transition to q_j at time $t + 1$.

Returning once again to the *forward-backward procedure* parameters α, β, con-

sider q_i, q_j as intermediate states between $t-1$ and $t+2$, which they are. Then if the joint event $\alpha_t(i)$ and $\beta_{t+1}(j)$ are equal we have probability 1, otherwise they are separated by a parameter $a_{ij}(O_{t+1}) < 1$ which is the transition state to q_j at time t with symbol O_{t+1}. $\alpha_t(i)$, as before, signifies t accumulated observations from $t = 1, 2, \ldots, t$ and $\beta_{t+1}(j)$ represents all occurrences from $t+1$ to T, the end of the observation sequence.

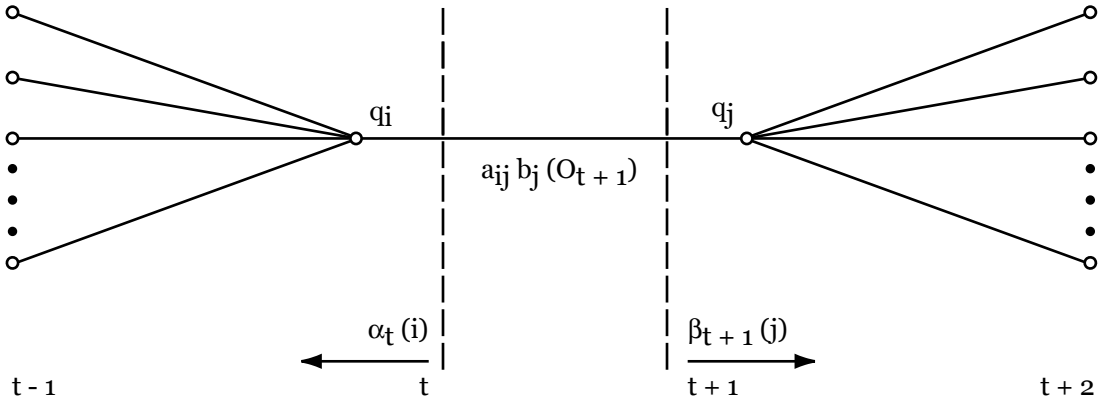

Figure 2.11: **HMM joint event.** Joint event that the system is in state q_i at time t, and state q_j at time $t+1$.

This scenario is expressed by the equation:

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}(O_{t+1})\beta_{t+1}(j)}{\Pr(\mathrm{O}|\lambda)} \tag{2.18}$$

and the Figure (2.11). λ is still the model represented by the HMM (A, B, π). $\bar{\lambda}$ is re-estimated using $\xi_t(i,j)$ in Equation (2.18) and the conglomeration of the Baum-Welch re-estimation formulas:

- $\bar{\pi} = \gamma_1(i) \qquad 1 \le j \le N,$
- $\bar{a}_{ij} = \sum_{t=1}^{T-1} \xi_t(i,j) \Big/ \sum_{t=1}^{T-1} \gamma_t(i),$
- $\bar{b}_j(k) = \sum_{\substack{t=1 \\ O_t=k}}^{T-1} \gamma_t(j) \Big/ \sum_{t=1}^{T} \gamma_t(j).$

L.R. Rabiner has written extensively on applications of HMMs and I have drawn heavily on his work. There are several extant implementations using HMMs for multiple sequence alignment: HMMER http://hmmer.wustl.edu/; SAM http://www.soe.ucsc.edu/research/compbio/sam.html; and HMMpro http://www.netid.com/html/hmmpro.html.

2.5.2 Identification of genes in human genomic DNA

Burge's methodology, upon which he based his program GENSCAN, is essentially the same as the framework described above for hidden Markov models (Burge, 1997; Burge & Karlin, 1997).

While the abstract theory of HMMs is given above, actually choosing a model and the state transitions for the biological "system" of a genome like the human genome, complete with genes, exons, introns, promoters, regulation regions, etc., is complex.

Burge determined his model parameters from non-redundant sets of human single and multi-exon genes. These sets were compiled from the GenBank, and were processed by the GeneParser test sets and the PROSET program. The model parameters consist of, e.g., state transition and initial probabilities and splice site models.

The following list and Figure (2.12) provide a more explicit description of the GENSCAN HMM model:

1. intergenic region;
2. 5' untranslated region (from transcription start to translation initiation);
3. single-exon, intronless gene (translation start to stop codon);
4. initial exon (translation start to donor splice site);
5. phase k internal exon (acceptor splice site to donor splice site);
6. 3' untranslated region (from after stop codon to polyadenylation signal);
7. polyadenylation signal;
8. phase k intron.

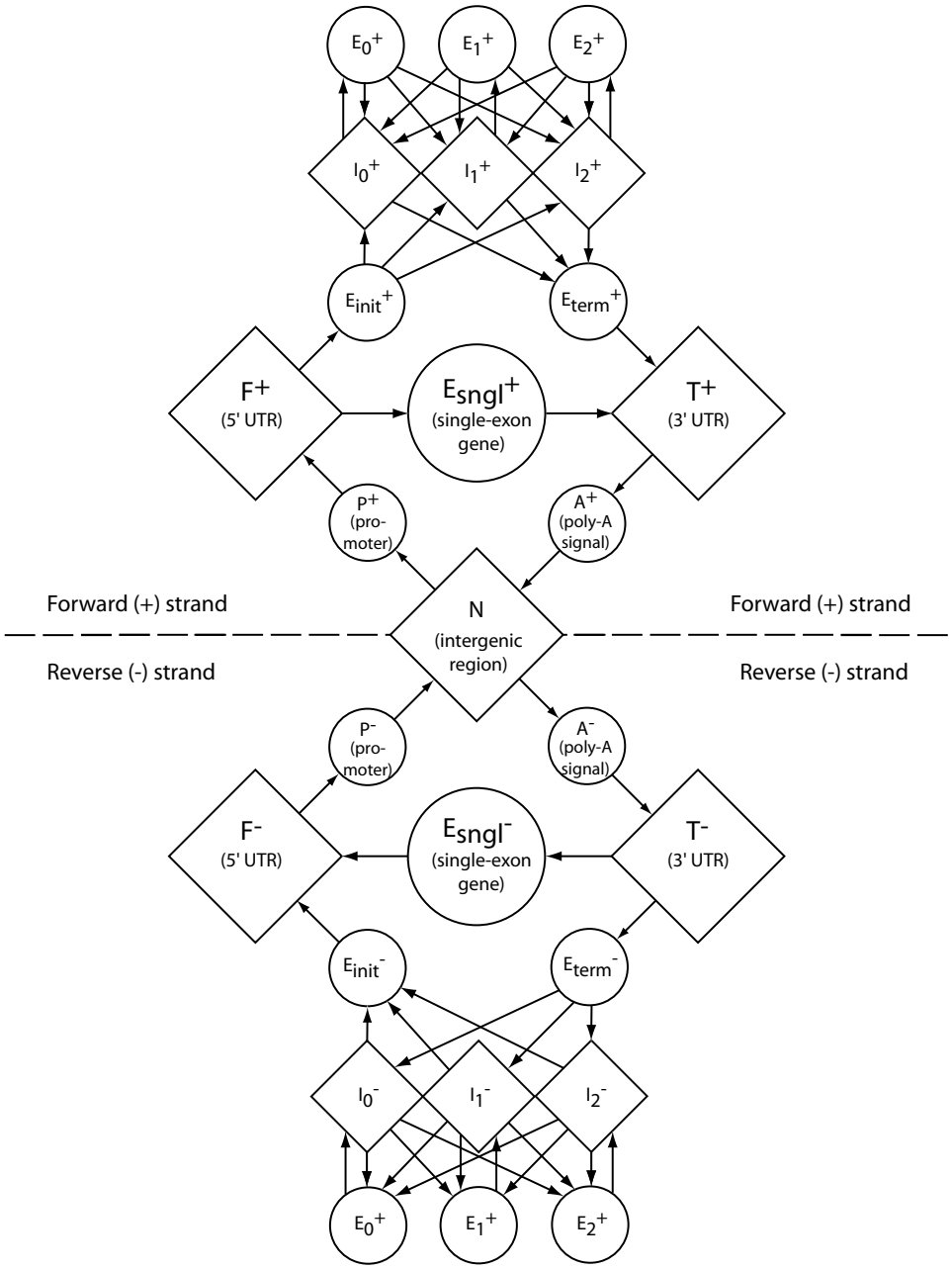

Figure 2.12: **GENSCAN HMM gene functional units.** In this diagram circles and diamonds represent functional units, HMM states, of genes or genomic regions: N, intergenic region; P, promoter; F, $5'$ untranslated region; E_{sngl}, single-exon, intronless gene; E_{init}, initial exon; E_k, phase k internal exon; E_{term}, terminal exon; T, $3'$ untranslated region; A, polyadenylation signal; I_k, phase k intron. The upper half of the figure corresponds to the forward strand (+), and the lower half to the reverse, complementary strand (-).

2.5.3 Statistical & probabilistic sequence analysis

The probabilistic model for match scoring is very well-developed. See for example Dembo et al. (1994) in the Annals of Probability. We would like to give an example of one equation from this foundational probabilistic paper on critical phenomena for sequence match scoring. It is worth remarking that even highly technical statistics, for example the recent work of Siegmund (Storey & Siegmund, 2001), still can't take into account the presence of gaps. And as we have repeatedly demonstrated any viable model *must* allow for gaps.

Here is Dembo's Equation (1):

$$\gamma^*(\mu_X, \mu_Y) \equiv \sup_{\nu \in M_1(\Sigma)} J(\nu) \equiv \sup_{\nu \in M_F(\Sigma)} J(\nu). \tag{2.19}$$

Definitions:

Σ_X and Σ_X are the alphabetic sequences.

μ_X and μ_Y refer to the probabilities on Σ_X and Σ_X.

γ^* is a constant expressed in terms of relative entropy, as in Equation (2.19).

$M_1(\Sigma)$ is the set of all probability measures on Σ.

$M_F(\Sigma)$ is the subset of probability measures ν satisfying $E_\nu(F) \geq 0$.

E is the *expected score* for some parameters.

$F(x, y)$ is the score for the letter pair (x, y).

ν and μ are the parameters of the relative entropy denoted by,

$$H(\nu|\mu) = \sum_{i=1}^{N} \nu(b_i) \log \frac{\nu(b_i)}{\mu(b_i)}.$$

$J : M_1(\Sigma) \to [-\infty, \infty)$

$$J(\nu) = \frac{E_\nu(F)}{H^*(v|\mu_X, \mu_Y)}.$$

Chapter 3

Experimental methods

Ohne Wein und ohne Weiber,
hol' der Teufel uns're Leiber!

—Johann Wolfgang von Goethe[1]

3.1 Algorithms

Rajasekaran presents a classical version of FFT alignment with a **position-specific scoring matrix** (PSSM).

Definition 3.1.1 (Position-specific scoring matrix). Let $\mathbf{A} = A_0 A_1 \dots A_{l-1}$ be a sequence in alphabet Λ, and $S_0, S_1, \dots, S_{m-1}$ be a sequence of scoring functions, then

$$S_k : A \xrightarrow{k} range(S_k); \qquad S_k(a) \in \mathbb{R}$$

$$\begin{pmatrix} \cdots & S_0 & \cdots \\ & S_1 & \\ & \vdots & \\ & S_{m-1} & \end{pmatrix}.$$

And there exists a position-specific match score $\mathbf{z} = (z_0, z_1, \dots, z_{n-1})$ which is calculated by summing over the j-th column of the PSSM and the correlated vector,

[1] "Without wine and without women,
The devil has taken our loves!"

as in Equation (3.1) below. In effect the PSSM keeps a history of match scores at different positions.

Scoring matrices improve the sensitivity of nucleotide similarity searches by calculating a score which indicates the degree of similarity between to sequences. The scoring scheme is fairly arbitrary, but traditional BLAST uses $+5$ for a match and -4 for a mismatch. PSSM's compare to the PAM matrices, Section (2.3.1), in that they both score alignments and store the scores in a matrices. Scoring matrices are a type substitution matrix.

The difficulty with developing an effective model for score values in scoring matrix algorithms is that, "one must be able to vary the gap and gap size penalties independently and in a query-dependent fashion in order to obtain the maximal sensitivity of the search" (Gusfield, 1997, page 279). In practice this is almost impossible, because you can't re-test your PSSM on every input data set. What is done instead is to do training runs on a set of input data to determine some values for the PSSM weights which give an acceptable sensitivity. The NCBI scientists did this to find the weights used above, and they are hard-wired as the defaults for the BLAST program. Given the wide range of applications for which BLAST is used, individual users are given the option of redefining the match/mis-match values for particular runs.

Like in Figure (3.1), for every set of sequences $\mathbf{A}$ and $\mathbf{B}$ there is a PSSM $\mathbf{S}$. Formally, $\mathbf{S} = S_0, S_1, \ldots, S_{m-1}$ is a sequence of column vectors, with a j-th linear alignment of $\mathbf{A}$ and $\mathbf{S}$.

The position-specific match score for $\mathbf{z} = (z_0, z_1, \ldots, z_{n-1})$ is

$$z_j = \sum_k S_{k'}(A_k) \tag{3.1}$$

and expresses formally the calculation we ran through in the caption to Figure (3.1). What it represents are the values $S_k(a)$ $(a \in \Lambda)$, the k-th column of the PSSM, summed over all aligned pairs $(A_k, S_{k'})$ where A_k is the k-th element of the sequence $\mathbf{A}$ and $S_{k'}$ is the column of the PSSM evaluated on A_k, $S_{k'}(A_k)$.

The matrix S is k-columns by m-rows. The dimension of the sequence $\mathbf{A}$ is $l-1$, dimension of the sequence $\mathbf{B}$ is $m-1$, and the dimension of the match score $\mathbf{z}$ (the

A	A	T	G	C	G

B		T	G	T	G	A

S	A	-4	-4	-4	-4	5
	C	-4	-4	-4	-4	-4
	G	-4	5	-4	5	-4
	T	5	-4	5	-4	-4

Figure 3.1: **Position-specific scoring matrices.** The figure represents the gapless global alignment between string sequences $\mathbf{A} = ATGCG$ and $\mathbf{B} = TGTGA$. For a pairwise scoring, the old BLAST scoring method defaults to $+5$ for a match and -4 for a mismatch. Thus, the global alignment **A** and **B** shown has four matching letters and a score of $5+5-4+5 = 11$. The PSSM **S** represents the pairwise scoring when a sequence is aligned with **B**. **S** can also be aligned with sequence **A** and the result $5 + 5 - 4 + 5 = 11$ is necessarily the same as the pairwise score between **A** and **B**. An example of local alignment, as opposed to global alignment, generated by the same two sequences is: take subsequences GCG in **A** and GTG in **B**. Local alignments ignore any other relationships, like PSSM scores or other letters outside the subsequences. Gapped local alignment appears impossible with the FFT. See the MAFFT discussion in Section (2.2) for details on global alignment using FFT. This figure was adapted from (Rajasekaran et al., 2002).

correlation value) is $n-1$. The index j has a range $0 \leq j \leq n-1$ corresponding to **z**. Therefore, the sum over the free index k (not the dimension of the columns) runs from MAX(0,$j-(m-1)$), covering each of the rows for each j (unless $j-(m-1) = 0$), to MIN($l-1$,j), the smaller dimension of **A** or **z**. k' is defined as $k' - k \equiv m-1-j$.

Fix a letter a in alphabet Λ. Let $\mathbf{x}_a = (I(a, A_0), I(a, A_1), \ldots, I(a, A_{l-1}))$ be the indicator vector of letter a in sequence vector **A**. Here, $I(a, b)$ is a pairwise scoring matrix for exact matches. This is the regular pairwise alignment measure, so for two letters a, b, if $a = b$, $I(a, b) = 1$, else $I(a, b) = 0$. The matrix I is symmetric,

that is $I(a,b) = I(b,a)$.

Let $\mathbf{y}_a = \mathbf{S}_a = (S_0(a), S_1(a), \ldots, S_{m-1}(a))$ be the row vectors $0, \ldots, m-1$ of the PSSM $\mathbf{S}$. The cross-correlation $\mathbf{z}_a = (z_{a,0}, z_{a,1}, \ldots, z_{a,n-1})$ of the query and database vectors $\mathbf{x}_a$ and $\mathbf{y}_a$ is the position-specific match score for letter a.

The j-th coordinate of the position-specific match score for the letter a is therefore,

$$z_{aj} = \sum_k I(a, A_k) S_{k'}(a). \tag{3.2}$$

Algorithm 3.1.2 (Rajasekaran's Algorithm 1). Vector $\mathbf{x}_a$ indicates exact matches between a fixed letter a and $\mathbf{A}_n$. So $\mathbf{x}_a = (I(a, A_0), I(a, A_1), \ldots, I(a, A_{l-1}))$ is the indicator vector of letter a in sequence vector $\mathbf{A}$. Vector $\mathbf{y}_a$ corresponds to row a of PSSM $\mathbf{S}$, so $\mathbf{y}_a = \mathbf{S}_a$. Every position in $\mathbf{S}_{\{0 \le n \le m-1\}}$ is a function of letter a, thus $\mathbf{S}_a = (S_0(a), S_1(a), \ldots, S_{m-1}(a))$. The only computation then is to compute the cyclic cross-correlation of $\mathbf{x}_a$ and $\mathbf{y}_a$,

$$\mathbf{Z}_{cyc} = \mathbf{X} \odot \mathbf{Y}_R. \tag{3.3}$$

This results in the cross-correlation vector $\mathbf{z}_a = (z_{a,0}, z_{a,1}, \ldots, z_{a,n-1})$. The j-th coordinate of the position-specific match score for the letter a is therefore,

$$z_{aj} = \sum_k I(a, A_k) S_{k'}(a). \tag{3.4}$$

Algorithm 3.1.3 (Rajasekaran's Algorithm 2). While Algorithm 3.1.2 doesn't take advantage of the complex-plane base encoding from Cheever et al. (1991) this algorithm uses the 2-vector complex plane encoding from Figure (3.4). Suppose letters $a, b \in \Lambda$. Then there exists an indicator vector for the two letters, $\mathbf{x}_{ab}$ in a sequence $\mathbf{A}$. Then

$$x_{ab,j} = I(a, A_j) + iI(b, A_j) \tag{3.5}$$

expresses the coordinates of the indicator vector for the j-th element of $\mathbf{A}$. With $\mathbf{S}_a$ and $\mathbf{S}_b$ defined as above, let $\mathbf{y}_{ab} = \mathbf{S}_a - i\mathbf{S}_b$. Then the cross-correlation $\mathbf{z}_{ab} =$

$(z_{ab,0}, z_{ab,1}, \ldots, z_{ab,n-1})$ of $\mathbf{x}_{ab}$ and $\mathbf{y}_{ab}$ has the coordinates

$$z_{ab,j} = \sum_k \{I(a, A_k)S_{k'}(a) + I(b, A_k)S_{k'}(b)\} - i\sum_k \{I(a, A_k)S_{k'}(b) + I(b, A_k)S_{k'}(a)\} \tag{3.6}$$

Comparing this with Equation (3.4) shows that $\mathrm{Re}(\mathbf{z}_{ab,j})$, the real part, equals $\mathbf{z}_a + \mathbf{z}_b$ from Algorithm 3.1.2. The CPU time is again $O(Ln \log n)$.

FFT convolution can be seen as a divide-and-conquer optimization of the "shift-and" algorithm to count string matches. Shift-and is in fact a very easy way to solve the match-count problem. The match-count problem is as follows. Consider two sequences $\mathbf{A} = A_0A_1 \ldots A_{l-1}$ and $\mathbf{B} = B_0B_1 \ldots B_{l-1}$. For each linear alignment between the two sequences, compute a match count (a, b), this amounts to counting the number of times the pair (a, b) occurs for an offset i. Most generally, we find the cyclic auto-correlation, with indices modulo $n + m$ by,

$$V_a(\alpha, \beta, i) = \sum_{j=0}^{j=n+m-1} \bar{\alpha}(j) \times \bar{\beta}(i + j). \tag{3.7}$$

And

$$\sum_{j=0}^{z-1} X(j) \times Y(i + j) = W(i), \tag{3.8}$$

the cyclic correlation of X & Y for a z-length real vector $W(i)$ where indices are all mod z. In fact, computing the vector $V_a(\alpha, \beta)$ is precisely the cyclic correlation of the two padded vectors $\bar{\alpha}_a$ and $\bar{\beta}_a$. Thus, the following relations hold: $W = V_a(\alpha, \beta)$, $X = \bar{\alpha}_a$, $Y = \bar{\beta}_a$, and $z = n + m$.

The equivalent statement in terms of FFT-convolution requires 3 FFTs and runs in time $O(N \log N)$. From Crandall & Pomerance (2001),

Theorem 3.1.4 (Convolution theorem). *Let signals x, y have the same length D. Then the cyclic convolution of x, y satisfies*

$$x \times y = DFT^{-1}(DFT(x) * DFT(y)), \tag{3.9}$$

or written in summation form

$$(x \times y)_n = \frac{1}{D} \sum_{k=0}^{D-1} X_k Y_k g^{kn}.$$

offset i=2	1	0	1	1	0	0	0	1	0	0	0	1	0	
AND			1	0	0	1	0	1	1	0				
Σ			1					1						=2

Figure 3.2: **Shift-and match count algorithm.** Shift-and is an easy way to find string similarity. Note for example that most processors have machine operations for "shift-right(accumulatorA,n)" and "bit-wise-AND(accumulatorA,operand)."

In Figure (3.2) only roughly one fourth of the necessary operations are represented. The first step in shift-and-ing nucleotide sequences is performing a binary encoding, for n,m-tuples α and β:

$$\alpha = \{agc \ldots t\} \longrightarrow 101100,$$

$$\beta = \{acc \ldots g\} \longrightarrow 100100110.$$

Breaking down each of the four component vectors, we get a total correlation value $V(\alpha, \beta, i)$ for offset i:

$$V(\alpha, \beta, i) = V_a(\alpha, \beta, i) + V_c(\alpha, \beta, i) + V_g(\alpha, \beta, i) + V_t(\alpha, \beta, i). \tag{3.10}$$

Equation (3.10) is a measure of local alignment, that is to say a substring α and a search database β. One can similarly have a match count value for an i-independent global alignment. The total correlation for such a global alignment is then, for all $\{a, g, c, t\}$ components, equal to the following sum:

$$V(\alpha, \beta) = V_a(\alpha, \beta) + V_c(\alpha, \beta) + V_g(\alpha, \beta) + V_t(\alpha, \beta). \tag{3.11}$$

3.2 Compression step

Cheever et al. (1991)'s main interest was implementing FFT correlation on specialized hardware, for example on DSP processors with 24-bit word sizes. His article was among the first to demonstrate that by different encodings of the bases, a reduction in the number of FFTs is possible. For instance, let $\alpha = a, c, t, a, t, g, a, t, t$ and $\beta = c, t, g, a, g, c, a, c, g$. Then computing the number of a matches by converting the sequences to binary (all $a \rightarrow 1$, else $x \rightarrow 0$) yields $\alpha = 1, 0, 0, 1, 0, 0, 1, 0, 0$ and $\beta = 0, 0, 0, 1, 0, 0, 1, 0, 0$. Repeating the process for all four bases results in a total of eight vectors/sequences. So it would take eight FFTs and one inverse FFT to compute the alignment.

There are two clever schemes to reduce the number of signal vectors which need to be Fourier-transformed. The so-called two-vector complex-plane base encoding scheme reduces the number of FFT calculations to four (plus one FFT^{-1}). For α and β above, one can encode the bases via the rule, $g \rightarrow i$ and $a \rightarrow 1$, where $i \equiv \sqrt{-1}$. This coding for sequence α results in $\alpha_{\{g,a\}} = 1, 0, 0, 1, 0, i, 1, 0, 0$ and likewise for the other two bases, $c \rightarrow i$ and $t \rightarrow 1$, $\alpha_{\{c,t\}} = 0, i, 1, 0, 1, 0, 0, 1, 1$.

The four FFTs on the input signals and the inverse FFT on the sum of the dyadic products of the corresponding encodings results in the cyclic cross-correlation W. A much-simplified though fully-functional C-format example of this is in Appendix (B.9).

$$W = FFT^{-1}[FFT(\alpha_{\{g,a\}}) \odot FFT(\alpha^R_{\{g,a\}}) + FFT(\beta_{\{c,t\}}) \odot FFT(\beta^R_{\{c,t\}})]. \quad (3.12)$$

The one-vector complex-plane encoding which we use goes one step further and allows for the encoding of a nucleotide sequence with four letters as one signal vector: $a \rightarrow 1$, $c \rightarrow -i$, $g \rightarrow i$, and $t \rightarrow -1$. This requires only $3 * N \log N$ operations. Cheever et al. (1991) noted that this approximation would not be exact, but we have not observed a noticeable difference in the error between the three encoding methods. It would perhaps be interesting to do a more thorough analysis of the numerical error introduced by multiple applications of the FFT to the different n-base vector encodings. Software precision, like that implemented in *Mathematica*, would be a very desirable component of such systems. Software precision in compiled

languages tends to be very difficult—note for example that multiplication of double-precision floating-point numbers doesn't commute on Intel processors (basically the rounding of the least significant digits is inconsistent).

$$\alpha = a, c, t, a, t, g, a, t, t$$

A	100100100
C	010000000
G	000001000
T	001010011

Figure 3.3: **4-Vector complex-plane base encoding**

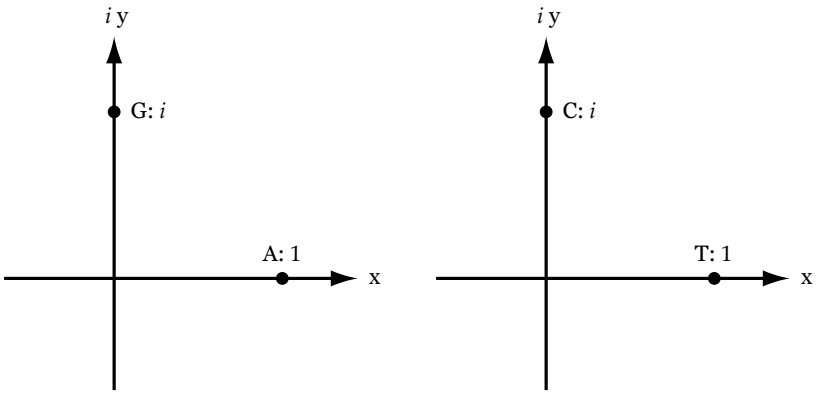

Figure 3.4: **2-Vector complex-plane base encoding**

3.3 Processing & pre-processing

Benson (1990b) made some interesting observations on probabilities of finding specific sequences of letters in GeneBank sequences. He noted that since in a given sequence a particular letter is often much more abundant than dictated by chance, analysis by orthogonal polynomials which assumes a more or less stationary distribution must take into account a non-standard weighting factor. Nucleic acids that are most abundant have an increased matching probability. As one observes a

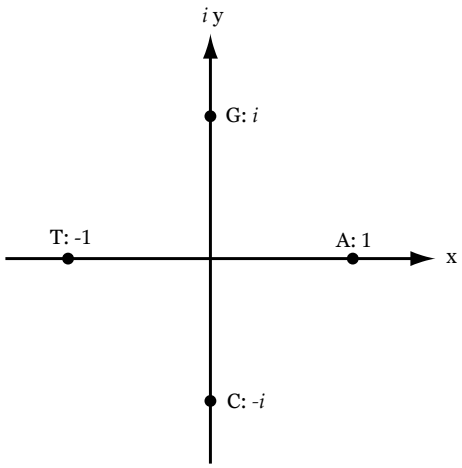

Figure 3.5: **1-Vector complex-plane base encoding**

particular sequence, the alphabetic densities vary over the length of the sequence. These non-stationary densities may cause false "matches." Therefore larger weights are assigned to less probable matches.

To this end, for a match between two letters with probabilities p_1 and p_2, we designate a weighting factor

$$W = 1/p_1 p_2. \tag{3.13}$$

Instead of merely counting the number of matching letters, via this method the total of the weight factors W corresponding to the individual matches is calculated. If a single match is statistically independent, W would have an expected value of 1. Necessarily if the probabilities of letters p_1 and p_2 have been correctly determined, for an alignment of two sequences of length L, the expected total of the weights also equals L. We can consider L a normalization factor for an alignment with unweighted match values. That is, if the nucleotide densities were equally distributed, the probabilities would be the same and before returning the total alignment value, it would be necessary to divide by L. The above modification of the basic method thus correctly adjusts the scores for sequences with non-stationary densities.

Let S_1 and S_2 be alphabetic sequences of length L. Generally we can think of the sequences as being circular. The coordinate system is defined relative to S_1. Switching the coordinate system to S_2 doesn't result in any lost (or gained) information, but it requires switching the reversal when accessing the correlation vector. The

density of the particular sequences or Fourier components can be modelled as mass densities. By using orthogonal polynomials it is also possible to do regressions on the "mass densities" which an alignment represents. Benson states, "Any discrete mass distribution can be approximated by a continuous density function and the goodness of the approximation is what we refer to as the 'resolving power'"(Benson, 1990b, page 6307). The alignment values can be represented as mass distributions in a sinusoidal graph covering an angle θ equal to the dimension of the DFT convolution vector.

The discrete mass distribution $D(x)$ is

$$D(x) = a_1\delta(x - x_1) + a_2\delta(x - x_2) + \ldots + a_m\delta(x - x_m),$$

where a_i are discrete masses, and the x_i are match locations for the alignment on the interval $[c, d]$. The Fourier coefficients F_i are

$$F_i = \int_c^d D(x)\phi_i(x)w(x)dx$$

where $\phi_i(x)$ are the orthogonal functions for a weight distribution function $w(x)$.

From DFT convolution of the i-th alignment we have match locations at integers

$$\{x_{ij}\}_{0 \le x_{ij} < L} \qquad\qquad j = 1, 2, \ldots, m_i.$$

m_i is the i-th component of the convolution. The Fourier coefficients for the mass density regression are:

$$\begin{aligned} P_i &= m_i/L \\ Q_i &= (2/L)\sum_{j=1}^{m_i} \sin(2\pi x_{ij}/L) \\ R_i &= (2/L)\sum_{j=1}^{m_i} \cos(2\pi x_{ij}/L). \end{aligned}$$

These constant coefficients form a total mass density equal to

$$P_i + Q_i \sin(2\pi x/L) + R_i \cos(2\pi x/L). \tag{3.14}$$

Define the indicator sequence $\mathbf{e}(S, \alpha)$ by

$$e_j(S, \alpha) \begin{cases} 1, & \text{if the } j\text{-th letter of } S \text{ is an } \alpha; \\ 0, & \text{otherwise.} \end{cases} \tag{3.15}$$

Define the tally of a sequence S by $\mathbf{a}(S, \mathbf{r}, \alpha)$, where $\mathbf{r}$ is a sequence vector we choose. Then,

$$a_j(S, \mathbf{r}, \alpha) = \begin{cases} r_j, & \text{if the } j\text{-th letter of } S \text{ is an } \alpha; \\ 0, & \text{otherwise.} \end{cases} \tag{3.16}$$

Define real sequences $\mathbf{r}'_{j2}$ and $\mathbf{r}''_j$ by

$$\mathbf{r}'_{j2} = \sin(2\pi(j + 1/2)/L)$$
$$\mathbf{r}''_j = \cos(2\pi(j + 1/2)/L).$$

Then the i-th component of the convolution

$$(2/L)\mathbf{a}(S_1, \mathbf{r}', \alpha) * \mathbf{e}(S_2^R, \alpha)$$

is equal to Q_i, and the i-th component of the convolution

$$(2/L)\mathbf{a}(S_1, \mathbf{r}'', \alpha) * \mathbf{a}(S_2^R, \alpha)$$

is equal to R_i. This is remarkable because one can, with slight modification, calculate as many terms as one likes of the Fourier series via FFT convolution. So a high order regression is not an impossibility.

A sinusoidal approximation using the first three terms of the Fourier series can be used to find the point mass distribution. The approximation is useful to us as it is: (1) optimal in the sense of least squares;[2] and (2) can be computed using an FFT. This is not to be confused with the DFT, here we are explicitly using expansions of Fourier coefficients for weight values.

When running BLAST it is necessary to first run the `formatdb` program on

[2]The least square approximation entails approximating a function $\xi(f)$ with some constants, say for a quadratic, a, b, c, such that $\xi(f) \cong af^2 + bf + c$.

the database one wishes to query. Similarly it would be possible to increase the speed of the FFT analysis by formatting the database in an optimized format. Such pre-processing could also aid in finding relationships between the different locations of the correlations across a very large database, not just the name of the tagged sub-sequence where the correlation was found. A logical design for a sequence comparison and alignment program with the necessary functionality of returning the associated values for the "name, location, and accession number" for an alignment requires pre-formatting in a database structure. This is one step towards making a functional query/lookup structure, and is a functionality we did not implement in our tests. It is much simpler in the case of the global alignment problem that the `MAFFT` designers tackled—no database is necessary since the sequences in question are all treated individually.

Appendix A

Quantum search, quantum Fourier transform, and alignment

A search algorithm is an alignment-independent comparison, that is it determines matches or approximate matches (think of `egrep(1)`, for example), it does not find sequence homology between a set of n-tuples. Quantum Fourier transform convolution can be used to find sub-sequences, or common local alignments between several sequences, with few modifications from the standard Fourier transform. The present author is aware of no quantum algorithms that perform alignment or global alignment. Computer graphicists have been interested in quantum algorithms for several specific applications. Shor's algorithm can calculate radiosity and Z-buffering, and Deutsch's algorithm allows double-speed communication on noisy communication channels. We are interested in quantum algorithms to find a remarkably efficient solution to protein and DNA sequence search problems.

There are essentially three classes of quantum algorithms presently in the literature:

1. Shor's algorithm for quantum Fourier transform, computing the discrete logarithm, and factoring large numbers;

2. Deutsch's algorithm for fast communication via quantum interference;

3. Grover's algorithm for quantum searching, running in time $O(\sqrt{N})$.

Grover's algorithm has mainly theoretical applications to substring search

and BLAST-like alignment problems. A striking reason why it is a theoretical approach is that the largest number of coherent qubits that have been prepared in the laboratory to date is ˜12. A scientist could perform such a small search by observation if it weren't the case that the number of effective computational basis states in an n qubit system is 2^n. So a 12 qubit system has $2^{12} = 4096$ states. Loading the "registers" with the nucleic acids, $x_i \in \{1, i, -1, -i\}$, on the order of several thousand is a current experimental difficulty.

The search algorithm determines the location of an element in a set, with the possible extension to multiple elements. For example, in a system with $n = 2$ qubits, and the set $x = \{0, 1, 2, 3\}$, we are looking for a unique value x_0 such that $f(x_0) = 1$, otherwise $f(x) = 0$. The average number of evaluations needed to find such a value x_0 in a random set on a classical computer is $9/4 = 2.25$. Grover's algorithm on a quantum computer can perform this search in a single evaluation.

There are seven steps to the experimental implementation of Grover's algorithm, for example using nuclear magnetic resonance (NMR), see e.g. (Chuang et al., 1998). But these steps can be separated into the explicit enumeration of the subroutine for the Grover iteration/operator G which has four steps, and the algorithm itself which only has four steps, one of which is applying G.

Just as in the short example above, we are looking for a value x_0 in the range $0 \leq x < 2^n$ for which $f(x_0) = 1$, otherwise $f(x) = 0$. A black box oracle O performs the transformation $O|x\rangle|q\rangle = |x\rangle|q \otimes f(x)\rangle$. Superposition is one of the most basic quantum transformations. The Hadamard transform, or "gate," $H^{\otimes n}$ applied to n qubits initially in state $|0\rangle$ results in the superposition

$$\frac{1}{\sqrt{2^n}} \sum_n |x\rangle. \tag{A.1}$$

The tensor product $\otimes n$ means heuristically "combine the n states." Thus the transform H produces an equal superposition of 2^n computational basis states using only n gates—what remarkable efficiency!

The Grover iteration G used below has four steps:

1. Apply the oracle O.

2. Apply the Hadamard transform $H^{\otimes n}$.

3. Perform the conditional phase shift:

$$|x\rangle \rightarrow -(-1)^{\delta_{x0}}|x\rangle$$
$$|x\rangle \rightarrow -|x\rangle \qquad \text{for } x > 0$$
$$|0\rangle \rightarrow |0\rangle.$$

4. Apply the Hadamard transform $H^{\otimes n}$ again.

The Hadamard transform H prepares the uniform superposition state $|\psi_0\rangle$ by rotating each qubit from the state $|0\rangle$ to $(|0\rangle + |1\rangle)/\sqrt{2}$. It can be written out explicitly for the 4-state example system above:

$$\begin{aligned} |\psi_0\rangle &= H|\psi_{in}\rangle = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \\ &= \frac{1}{2}\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \end{aligned} \tag{A.2}$$

Quantum search algorithm: The input to the algorithm is $n+1$ qubits in the state $|0\rangle$. The runtime for the algorithm is $O(\sqrt{2^n})$, as the range above is $0 \leq x < 2^n$. The four steps are as follows:

1. Initialize the states

$$|0\rangle^{\otimes n}|0\rangle.$$

2. Apply $H^{\otimes n}$ to the first n qubits, and a conditional Hadamard to the last qubit

$$\frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right].$$

3. Apply the Grover iteration $R \approx \pi\sqrt{2^n}/4$ times

$$[(2|\psi\rangle\langle\psi| - I)O]^{\otimes R} \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] \qquad \approx |x_0\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right].$$

4. Measure the first n qubits and return x_0

$$\rightarrow x_0.$$

Grover suggested that while the algorithm in its most general formulation can only search for a unique state satisfying the constraint, it can be easily modified to search for multiple states. One way to achieve this is to repeat the experiment and check the range of degeneracy (degeneracy between states occurs when they have linearly independent eigenvectors with the same eigenvalue). By checking the degeneracy of solutions in the range $(k, k+1, \ldots, 2k)$, one would be able to tell whether there were multiple occurrences of a state. For a system of N states it would take $\log N$ repetitions to perform the degeneracy checking. Another possibility for checking for multiple occurrences of a state is to introduce a random perturbation to remove the degeneracy with a very high probability. Detailing this technique beyond the scope of this thesis.

Appendix B

Implementation of routines

The following are C, Java, and *Mathematica* format implementations of some of the methods documented in Chapters 1–3.

B.1 Correlation algorithm in *Mathematica*

```
len = 2048;
w = 5;
d = Table[Exp[2 Pi I Random[Integer, {0,3}]/4], {len}];
q = Table[If[j <= w, Exp[2 Pi I Random[Integer, {0, 3}]/4], 0], {j, 1, len}];

Take[q, {1, w}]

corr = Sqrt[len]*InverseFourier[Fourier[d]*Conjugate[Fourier[q]]];

ListPlot[Abs[corr], PlotJoined -> True, PlotRange -> All];
```

B.2 cleanDNA.c

```
#include <stdlib.h>
#include <stdio.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

#define TRUE 1
#define FALSE 0
#define DEBUG FALSE

#if DEBUG
#define DEBUGF(a...) do{fprintf(stderr, "%s[%d]", __FILE__, __LINE__); \
```

```
                    fprintf(stderr, ##a);}while(0)
#else
#define DEBUGF(a...)
#endif

int main( int argc, char *argv[] ) {
    FILE *fp_in, *fp_out;
    char ch;

    if( argc != 3 ){
        printf("usage: %s dirty_dna_file clean_dna_file \n"
               "convert a dirty DNA file {agctAGCT} to a clean one\n"
               "(remove line numbers, spaces, line breaks, etc.)", argv[0]);
        exit(1);
    }
    if( (fp_in = fopen( argv[1], "r" )) == NULL ) {
        fprintf( stderr, "fopen \"%s\" failure\n", argv[1]);
        exit(1);
    }
    if( (fp_out = fopen( argv[2], "w" )) == NULL ) {
        fprintf( stderr, "fopen \"%s\" failure\n", argv[2]);
        exit(1);
    }

    while( (ch = fgetc(fp_in)) != EOF ) {
        switch (ch){
        case 'A': case 'a':
            fprintf( fp_out, "A");
            break;
        case 'G': case 'g':
            fprintf( fp_out, "G");
            break;
        case 'C': case 'c':
            fprintf( fp_out, "C");
            break;
        case 'T': case 't':
            fprintf( fp_out, "T");
            break;
        default:
            /* some non-AGCT input in file ignored */
            DEBUGF("some non-AGCTagct in input file ignored, ch = %c \n", ch);
            break;
        }
    }

    fclose( fp_in );
    fclose( fp_out );
    return 0;
```

}

B.3 replaceplain.c

```
#include <stdlib.h>
#include <stdio.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

#define TRUE 1
#define FALSE 0
#define DEBUG FALSE

#if DEBUG
#define DEBUGF(a...) do{fprintf(stderr, "%s[%d]", __FILE__, __LINE__); \
                     fprintf(stderr, ##a);}while(0)
#else
#define DEBUGF(a...)
#endif

int main( int argc, char *argv[] ) {
    FILE *fp_in, *fp_out;
    char ch;

    if( argc != 3 ){
        printf("usage: %s orig_file new_file \n"
               "convert a DNA file to 1-vector complex plane base encoding \n"
               "Mathematica format file: \n"
               "C := -i := \\(-\\[ImaginaryI]\\), G:= i := \\[ImaginaryI], \n"
               "T := -1 := \\(-1\\), A := 1, \n"
               "and { a,b,. . . } list. \n", argv[0]);
        exit(1);
    }
    if( (fp_in = fopen( argv[1], "r" )) == NULL ) {
        fprintf( stderr, "fopen orig_file %s failure\n", argv[1]);
        exit(1);
    }
    if( (fp_out = fopen( argv[2], "w" )) == NULL ) {
        fprintf( stderr, "fopen new_file %s failure\n", argv[2]);
        exit(1);
    }

    fprintf( fp_out, "{");
    while( (ch = fgetc(fp_in)) != EOF ) {
        switch (ch){
```

```
        case 'A': case 'a':
            fprintf( fp_out, "1, ");
            break;
            /* printf("got an A!!\n"); */
        case 'G': case 'g':
            fprintf( fp_out, "\\[ImaginaryI], ");
            break;
        case 'C': case 'c':
            fprintf( fp_out, "-\\[ImaginaryI], ");
            break;
        case 'T': case 't':
            fprintf( fp_out, "(-1), ");
            break;
        default:
            /* some non-AGCT input in file ignored
               if (!(((ch == 'a') || (ch == 'g') || (ch == 'c') || (ch == 't')))) */
            DEBUGF("some non-AGCT in input file ignored, ch = %c \n", ch);
            break;
        }
    }

    fprintf( fp_out, "}");
    fclose( fp_in );
    fclose( fp_out );

    /* delete final comma ... */
    execl("commakill.sh","./commakill.sh", argv[2], (char *)0);

    return 0;
}
```

B.4 replacesize.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

/*
   Author: Russell Hanson
   2002.11.14
   Reed College

   compile with:
```

```
   gcc replacesize.c -o replacesize -Wall -ansi -pedantic

   standard usage:
   1985 r@hale ~> ./replacesize DNA-query.txt DNA-query-math.txt DNA-data.txt

    // uncomment for (inverted) Gaussian response function
    if (i <= N/2) {
      response[i] = exp(-0.5*sqr(i/0.05/N))/0.05/N/sqrt(2*Pi);
      response[N-1-i] = -response[i];
    }

$ cat commakill.sh
#!/bin/sh

awk '{gsub(", }", " }", $0); print > FILENAME}' $1

*/

typedef int boolean;
#define TRUE 1
#define FALSE 0
#define DEBUG FALSE

#if DEBUG
#define DEBUGF(a...) do{fprintf(stderr, "%s[%d]", __FILE__, __LINE__); \
                         fprintf(stderr, ##a);}while(0)
#else
#define DEBUGF(a...)
#endif

/* PROTOTYPES */
char *alloc (register int bytes);
int StringWidth(char *s);
char *progName( char *str );

/* DEFINES */

#define max_math_size 32 /* theo. number of bytes the "largest" (usually
                            mathka) output string corresponds to */

int main( int argc, char *argv[] ) {
    FILE *fp_in, *fp_out;
    char ch; int i,j,k;
    unsigned int Acount = 0, Gcount = 0, Ccount = 0, Tcount = 0, AGCTcount = 0;
    double Aratio = 0, Gratio = 0, Cratio = 0, Tratio = 0;
    double Aweight = 0, Gweight = 0, Cweight = 0, Tweight = 0;
    struct stat orig_statbuf;
    struct stat new_statbuf;
```

```
struct stat data_statbuf;
unsigned long int in_filesize = 0; /* DNA query file */
unsigned long int out_filesize = 0; /* DNA query filesize after processing */
unsigned long int data_filesize = 0; /* DNA data filesize (from argv[3]) */
unsigned int padsize = 0; /* data_filesize - AGCTcount */
char *parsebuf;

if( argc != 4 ){                                                    70
    printf("usage: %s DNA_query_file DNA_query_file_math DNA_data_file\n"
           "Convert a DNA_query_file to 1-vector complex plane base encoding \n"
           "padded for dyadic multiplication with data_file and in Mathematica format: \n"
           "C := -i := \\(-\\[ImaginaryI]\\), G:= i := \\[ImaginaryI], \n"
           "T := -1 := \\(-1\\), A := 1, \n"
           "and { a,b,. . . } list. \n", progName( argv[0] ));
    exit(1);
}
if( (fp_in = fopen( argv[1], "r" )) == NULL ) {
    fprintf( stderr, "fopen orig_file %s failure\n", argv[1]);      80
    exit(1);
}
if( (fp_out = fopen( argv[2], "w" )) == NULL ) {
    fprintf( stderr, "fopen new_file %s failure\n", argv[2]);
    exit(1);
}

/*
  New usage: instead of specifying the DNA source file _size_ as the 3rd
  argument, simply give the file _name_, and stat() will calculate the  90
  filesize for you.  Watch out for sparse files. */
stat( argv[3], &data_statbuf );
data_filesize=data_statbuf.st_size;
printf("DNA_data_file \t\"%s\" size = %8ld bytes \n", argv[3], data_filesize);
/*
  Old usage: data_filesize = (unsigned long int) atol(argv[3]) ; NUL
   terminated DNA-source added one by hand on command line, even though the
   NUL will always be there */

/* get orig_file size for max buffer size; allocate parsebuf */       100
stat( argv[1], &orig_statbuf );
in_filesize=orig_statbuf.st_size;
printf("DNA_query_file \t\"%s\" size = %8ld bytes \n", argv[1], in_filesize);

parsebuf = (char *) alloc( sizeof(char) * data_filesize );

/*
   Read agctAGCT's from fp_in to parsebuf, count up the total agctAGCT's
   DNA-query.txt may be ( uppercase || lowercase )
*/                                                                  110
```

```
while( (ch = fgetc(fp_in)) != EOF ) {
    switch (ch){
    case 'A': case 'a':
        strcat( parsebuf, "A");
        Acount++;
        break;
    case 'G': case 'g':
        strcat( parsebuf, "G");
        Gcount++;
        break;                                                    120
    case 'C': case 'c':
        strcat( parsebuf, "C");
        Ccount++;
        break;
    case 'T': case 't':
        strcat( parsebuf, "T");
        Tcount++;
        break;
    default:
        /* some non-AGCT in input file ignored */                 130
            DEBUGF("some non-AGCT in input file ignored, ch = %c \n", ch);
        break;
    }
}
AGCTcount = Acount+Gcount+Ccount+Tcount;
if(DEBUG){
    fputs(parsebuf, stdout);
    printf("\n AGCTcount = %d\n", AGCTcount);
}
/* remove mean above zero excess matches                         140
   A = 0.25/0.35= 0.71;
   C = -0.25i/0.18 = -1.38i, etc.
*/

Aratio = (double) Acount/AGCTcount;
Gratio = (double) Gcount/AGCTcount;
Cratio = (double) Ccount/AGCTcount;
Tratio = (double) Tcount/AGCTcount;
DEBUGF("Aratio = %f\n", Aratio);
                                                                  150
Aweight = 0.25/Aratio;
Gweight = 0.25/Gratio;
Cweight = 0.25/Cratio;
Tweight = 0.25/Tratio;
DEBUGF("Aweight = %f\n", Aweight);

if (data_filesize < AGCTcount) /* data smaller than query!! */
    exit( printf("ERROR: \t data smaller than query!\n"
```

```
                "\t make sure you're actually searching for something\n") );
padsize = abs(data_filesize − AGCTcount);
DEBUGF("padsize = %d \n", padsize);

/*
   strcat() the extra padding to parsebuf
*/

for(i=0; i<padsize; i++){
    strcat( parsebuf, "0" ); /* forget the padded AGCT's, use zeros */
    DEBUGF("padsize i = %d\n", i);
}

/*
  Write the output file.
  Sample mathka format:
  Cell[BoxData[ \({4.5*1, \(-\((3.2)\)\)*\[ImaginaryI],
  2.5*\[ImaginaryI], \ \(-\((1.2)\)\)*1}\)], "Input"]
*/
j = 0; /* count the number of fprintf()'s */
/* printf("test setting {A,G,C,T}weights = 1 \n");
    Aweight = Gweight = Cweight = Tweight = 1; */
fprintf( fp_out, "{ ");
while( *parsebuf != '\0' ) {
    switch (*parsebuf){
    case 'A':
        fprintf( fp_out, "(%.10f)*1, ", Aweight);
        break;
    case 'G':
        fprintf( fp_out, "(%.10f)*\\[ImaginaryI], ", Gweight);
        break;
    case 'C':
        fprintf( fp_out, "(%.10f)*(-1)*\\[ImaginaryI], ", Cweight);
        break;
    case 'T':
        fprintf( fp_out, "(%.10f)*(-1), ", Tweight);
        break;
    case '0':
        fprintf( fp_out, "0, ");
        break;
    }
    parsebuf++;          /* be sure to get first entry, increment after */
    j++;
}
DEBUGF("\n sanity check: padsize + AGCTcount = data_filesize = %lu \n"
      "(*parsebuf != '\\0'); parsebuf++; j++;) = %d \n", data_filesize, j );
fprintf( fp_out, "}");
```

```
    if( fflush( fp_in ) || fflush( fp_out ) || fflush(NULL))
        printf("fflush() error \n");

    printf("fp_out %d\n", fclose( fp_out ));
    printf("fp_in %d\n", fclose( fp_in ));

    stat( argv[2], &new_statbuf );
    out_filesize = new_statbuf.st_size;
    printf("DNA_query_file_math \t\"%s\" size = %8ld bytes \n", argv[2], out_filesize );

    /* delete final comma ... */
    execl("commakill.sh","./commakill.sh", argv[2], (char *)0);

    free( parsebuf );
    return 0;
}

/*
 * Return width of string s
 */

int
StringWidth(char *s) {
    unsigned int w = 0;

    while ((*s & 0xff) != '\0') {
        /* w += font->widths[*s & 0xff]; */
        w++;
        s++;
    }
    return w;
}

char *progName( char *str )
{
    char *value;
    if( ( value = strrchr( str, '/' ) ) != NULL )
        return( value+1 );
    else
        return( str );
}

/*
 * alloc :  allocate memory
 */

char *alloc (register int bytes) {
    register    char    *memory;
```

```
    /*      auto            char    *malloc();*/
    if ((memory = malloc( (unsigned) bytes )) == NULL){
        fputs("malloc error \n", stderr);
        exit(1);
    }
    return memory;
}
```

B.5 commakill.sh

```
#!/bin/sh

awk '{gsub(", }", " }", $0); print > FILENAME}' $1
```

B.6 DOIT.sh

```
#!/bin/sh

./cleanDNA.exe DNA.txt DNA-data.txt
./replaceplain.exe DNA-data.txt DNA-data-math.txt
./replacesize.exe DNA-query.txt DNA-query-math.txt DNA-data.txt
```

B.7 FFT.c

```
/* We aren't allowed to publish the "Numerical Recipes in C" version
of the FFT... imagine seeing it in its full glory at your local
bookstore or library. */

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

/* Numerical Recipes fast Fourier transform. */
void four1(float data[], unsigned long nn, int isign);
float * lotsadata(void);

int my_main (int argc, char **argv) {
    unsigned int nsamp = 1E8;
    static float * thedata;
    thedata = (float*)malloc( 2 * nsamp * (size_t) sizeof(float));
    thedata = lotsadata();
    four1(thedata-1, nsamp, 1);
    return 0;
}
```

B.8 nofasta.java

```
import java.io.*;

// Removes FASTA information from a database file
// Permits the encoding and correlation of an entire database file
// Accession numbers and names are removed, but may be (re)stored

// usage: java nofasta < est_human > est_human_ohne_gthan.txt

class nofasta {

    public static void main (String argv[]) {

        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        BufferedWriter out = new BufferedWriter(new OutputStreamWriter(System.out));
        String buffer;

        try {
            nextline: while ( (buffer=in.readLine()) != null ) {
                if ( buffer.charAt(0) == '>' ) {
                    System.err.println("we have evidence of a '>', Sir");
                    continue nextline;
                }
                else{
                    out.write(buffer.concat("\n"));
                }
            }
            out.flush();
        }
        catch (IOException ioe) {}
    }
}
```

B.9 fftconvolve.c

```
/* Find jahan's primers in the est_human database file.

build: g++ fftconvolve.c fft.c -o fftconvolve -Wall -pedantic -lm

input: some "complex data[]"-formatted arrays in the source file

output: a file dump of double * corr
```

```
*/
                                                                    10
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "fft.h"

static const complex A = { 1.0, 0.0 };
static const complex C = { 0.0, -1.0 };
static const complex G = { 0.0, 1.0 };                              20
static const complex T = { -1.0, 0.0 };
static const complex N = { 0.0, 0.0 };

#define dsize 256

/*
 * double * y is absolute value (modulus) of complex * x
 */
void abs_complex(complex * x, double * y, int n){
    int i;                                                          30

    /* modulus |x+iy|; */
    for (i = 0; i< n; i++) {
        y[i] = sqrt( pow(x[i].re,2) + pow(x[i].im,2) );
    }

}

int main (){
    int i,j,k;                                                      40
    FILE * fp;

    double sqrtdsize = sqrt(dsize);

    complex * datap;
    complex * queryp;
    double * corr;              /* fed to abs_complex */

    complex data[] = {
        C,A,C,C,C,C,C,G,G,A,T,G,A,G,C,A,T,T,A,G,T,C,T,A,C,C,T,G,C,T,A,A,  50
        A,G,A,A,G,G,C,T,G,G,C,A,C,T,G,T,A,C,G,A,A,C,T,C,A,G,C,A,C,A,T,A,
        T,G,C,A,G,C,C,C,A,A,T,T,G,T,T,T,T,T,G,G,C,G,T,C,T,G,C,C,T,T,G,A,
        C,T,T,A,G,T,A,A,G,C,A,A,A,C,T,C,A,T,G,A,T,A,C,C,A,T,A,T,A,C,C,A,
        G,C,T,T,C,C,A,T,C,A,G,T,T,A,C,G,A,T,C,T,T,G,C,T,G,A,C,G,A,C,T,T,
        C,C,A,C,C,C,G,T,A,C,T,C,G,G,G,T,T,A,A,A,C,G,A,C,C,T,A,G,T,C,C,C,
        C,G,G,T,C,A,G,A,C,A,T,C,G,A,C,T,G,T,T,C,A,G,C,A,T,C,T,C,G,T,A,C,
```

```
    A,A,T,T,T,A,C,A,T,G,C,C,T,G,G,A,A,C,G,G,G,A,T,A,G,C,A,C,A,C,C,T
};

complex query[] = {
    A,G,A,A,G,G,C,T,G,G,C,A,C,T,G,T,A,C,G,A,N,N,N,N,N,N,N,N,N,N,N,N,
    N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
    N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
    N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
    N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
    N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
    N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
    N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N
};

datap = new_complex_signal(dsize);
queryp= new_complex_signal(dsize);
corr = new_real_signal(dsize);

for (i=0;i<dsize;i++){
    datap[i] = data[i];
    queryp[i] = query[i];
}

/*
   convolve_complex_fourier(complex *x, complex *y, int n, int initial);

   fft_ split(y, n);
   mul_dyadic_complex(x, y, n);
   ifft_split(y   ,n);

   saves y (y := x cyclic y)
*/

fft_split(queryp, dsize);
conjugate_signal(queryp, dsize);
fft_split(datap, dsize);
mul_dyadic_complex(datap, queryp, dsize);
ifft_split(queryp, dsize);

abs_complex(queryp, corr, dsize);

scale_signal(sqrtdsize, corr, dsize);


fp=fopen("convolve.out","w");
fprintf(fp,"yo!\n");

for (i=0;i<dsize;i++){
```

```
            fprintf(fp,"%d\t%.20f\n",i,corr[i]);
        }

        return 0;
    }
```

References

Altschul, S. F., Gish, W., Miller, W., Meyers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, 215(3), 403–410.

Altschul, S. F. & Lipman, D. J. (1990). Protein databse searches for multiple alignments. *Proc. Natl. Acad. Sci. USA*, 87, 5509–5513.

Altschul, S. F., Madden, T., Schaffer, A., Zhang, J., Zhang, Z., Miller, W., & Lipman, D. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17), 3389–3402.

Arndt, J. (2002). Algorithms for programmers. http://www.jjj.de/fxt/.

Benson, D. C. (1990a). Digital signal processing methods for biosequence comparison. *Nucleic Acids Research*, 18(10), 3001–3006. http://nar.oupjournals.org/cgi/content/abstract/18/10/3001.

Benson, D. C. (1990b). Fourier methods for biosequence analysis. *Nucleic Acids Research*, 18(21), 6305–6310. http://nar.oupjournals.org/cgi/content/abstract/18/21/6305.

Biedenharn, L. & Louck, J. (1981). *Angular Momentum in Quantum Physics*. Reading, MA: Addison-Wesley.

Blahut, R. E. (1985). *Fast algorithms for digital signal processing*. Reading, MA: Addison-Wesley.

Bracewell, R. N. (1986). *The Fourier Transform and its Applications*. New York: McGraw-Hill.

Burge, C. (1997). *Identification of Genes in Human Genomic DNA (GENSCAN).* PhD thesis, Stanford University, Stanford, CA.

Burge, C. & Karlin, S. (1997). Prediction of complete gene structures in human genomic DNA. *J. Mol. Biol.*, 268, 78–94.

Cheever, E. A., Overton, G. C., & Searls, D. B. (1991). Fast Fourier transform-based correlation of DNA sequences using complex plane encoding. *Computer Applications in the Biosciences*, 7(2), 143–154.

Christoffels, A., van Gelder, A., Greyling, G., Hide, R. M. T., & Hide, W. (2001). Stack: sequence tag alignment and consensus knowledgebase. *Nucleic Acids Research*, 29, 234–238.

Chuang, I. L., Gershenfeld, N., & Kubinec, M. (1998). Experimental implementation of fast quantum searching. *Phys. Rev. Lett.*, 80(15), 3408–3411.

Cooley, J. & Tukey, J. (1965). An algorithm for the machine calculation of complex Fourier series. *Math. Comput.*, 19, 297–301.

Cormen, T. H. (1998). *Introduction to Algorithms.* Cambridge, MA: MIT Press.

Crandall, R. & Pomerance, C. (2001). *Prime Numbers: a computational perspective.* New York: Springer-Verlag.

Crandall, R. E. (1994). *Projects in Scientific Computation.* New York: Springer-Verlag.

Danielson, G. & Lanczos, C. (1942). Some improvements in practical Fourier analysis and their application to X-ray scattering from liquids. *J. Franklin Inst.*, 233, 365–380.

Dayhoff, M., Schwartz, R., & Orcutt, B. (1978). A model of evolutionary change in proteins. In M. Dayhoff & R. Ech (Eds.), *Atlas of Protein Sequence and Structure* (pp. 345–352). MD: National Biomedical Research Foundation.

Dembo, A., Karlin, S., & Zeitouni, O. (1994). Critical phenomena for sequence matching with scoring. *The Annals of Probability*, 22(4), 1993–2021.

Edwards, R. E. (1979). *Fourier Series, a modern introduction, Vols. 1 & 2*. New York: Springer-Verlag.

Felsenstein, J., Sawyer, S., & Kochin, R. (1982). An efficient method for matching nucleic acid sequences. *Nucleic Acids Research*, 10(1), 133–139.

Fiqueroa, H. & Gracia-Bondia, J. (Mar. 2000). On the antipode of Kreimer's Hopf algebra. *Los Alamos HEP Archive*. arXiv:hep-th/9912170.

Forney, G. (1974). The Viterbi algorithm. *Proceedings of the IEEE*, 61, 268–278.

Fourier, B. J.-B. J. (1820). *Œuvres de Fourier: Théorie analytique de la chaleur*. Paris: Imprimerie de Gauthier-Villars et fils.

Friedberg, I., Kaplan, T., & Margalit, H. (2000). Evaluation of PSI-BLAST alignment accuracy in comparison to structural alignments. *Protein Science*, 9, 2278–2284.

Friedberg, I. & Margalit, H. (2002a). PeCoP: automatic determination of persistently conserved positions in protein families. *Bioinformatics Applications Note*, 18(9), 1276–1277.

Friedberg, I. & Margalit, H. (2002b). Persistently conserved positions in structurally similar, sequence dissimilar proteins: Roles in preserving protein fold and function. *Protein Science*, 11, 350–360.

Frigo, M. & Johnson, S. G. (1998). FFTW: An adaptive software architecture for the FFT. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, volume 3 (pp. 1381–1384). Seattle, WA: IEEE. http://citeseer.nj.nec.com/frigo98fftw.html.

Ganapathiraju, A., Hamaker, J., Picone, J., & et al. (1997). A comparative analysis of FFT algorithms. http://citeseer.nj.nec.com/256396.html.

Gotoh, O. (1993). Optimal alignment between groups of sequences and its application to multiple sequence alignment. *Comput. Appl. Biosci.*, 9, 361–370.

Grantham, R. (1974). Amino acid difference formula to help explain protein evolution. *Science*, 185, 862–864.

Grover, L. (1996). A fast quantum mechanical algorithm for database search. In *Proc. 28th Annual ACM Symposium on the Theory of Computation* (pp. 212–219). New York: ACM Press.

Grover, L. (1998). Quantum computers can search rapidly by using almost any transformation. *Phys. Rev. Lett.*, 80(19), 4329–4332.

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

Heideman, M. T. (1988). *Multiplicative complexity, convolution, and the DFT*. New York: Springer-Verlag.

Henikoff, S. & Henikoff, J. (1992). Amino acid subtitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA*, 89(10), 915–919.

Karlin, S. & Altschul, S. F. (1990). Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc. Natl. Acad. Sci. USA*, 87(6), 2264–2268.

Katoh, K., Misawa, K., Kuma, K., & Miyata, T. (2002). MAFFT: a novel method for rapid multiple sequence alignment based on the fast Fourier transform. *Nucleic Acids Research*, 30(14), 3059–3066.

Kent, W. J. (2002). BLAT—the BLAST-like alignment tool. *Genome Research*, 12, 656–664.

Kent, W. J. & Haussler, D. (2001). Assembly of the working draft of the human genome with GigAssembler. *Genome Research*, 11, 1541–1548.

Kent, W. J., Sugnet, C. W., Furey, T. S., Roskin, K. M., Pringle, T. H., Zahler, A. M., & Haussler, D. (2002). The human genome browser at UCSC. *Genome Research*, 12(6), 996–1006.

Knuth, D., Morris, J., & Pratt, V. (1977). Fast pattern matching in strings. *SIAM J. Comput.*, 6, 323–350.

Lander, E. & the International Human Genome Sequencing Consortium (2001). Initial sequencing and analysis of the human genome. *Nature*, 409(15), 860–921.

Lander, E. & Waterman, M. (1988). Genomic mapping by fingerprinting random clones: a mathematical analysis. *Genomics*, 2, 231–239.

Lander, E. & Waterman, M. S. (1998). Genomic mapping by fingerprinting random clones: a mathematical analysis. *Genomics*, 2, 231–239.

Lange, K. (1997). *Mathematical and Statistical Methods for Genetic Analysis*. New York: Springer-Verlag.

Lynch, M. (2002). Intron evolution as a population-genetic process. *Proc. Natl. Acad. Sci. USA*, 99, 6118–6123.

Maizel, J. J. & Lenk, R. (1981). Enhanced graphic matrix analysis of nucleic acid and protein sequences. *Proc. Natl. Acad. Sci. USA*, 78(12), 7665–7669.

Mosca, M. (1999). *Quamtum Computer Algorithms*. PhD thesis, University of Oxford.

Myers, G. & Durbin, R. (2002). Invited Lecture—Accelerating Smith-Waterman Searches. In R. Guig & D. Gusfield (Eds.), *Algorithms in Bioinformatics*, Lecture Notes in Computer Science 2452(LNCS) (pp. 331–342).: Second International Workshop, WABI 2002, Rome, Italy, September 17-21, 2002. Proceedings Springer-Verlag.

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

Nielson, M. & Chuang, I. (2000). *Quantum Computation and Quantum Information*. Cambridge: Cambridge University Press.

Percus, J. K. (2002). *Mathematics of Genome Analysis*. Cambridge: Cambridge University Press.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Recipes in C*. Cambridge: Cambridge University Press.

Rabiner, L. & Juang, B. (Jan. 1986). An introduction to hidden Markov models. *IEEE ASSP Magazine*, (pp. 4–16).

Rabiner, L. R. (Feb. 1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 257–286.

Rajasekaran, S., Jin, X., & Spouge, J. (2002). The efficient computation of postition-specific match scores with the fast Fourier transform. *Journal of Computational Biology*, 9(1), 23–33.

Rajasekaran, S., Nick, H., Pardalos, P., Sahni, S., & Shaw, G. (2001). Efficient algorithms for local alignment search. *Journal of Combinatorial Optimization*, 5, 117–124.

Schäffer, A., Aravind, L., Madden, T., Shavirin, S., Spouge, J., Wolf, Y., Koonin, E., & Altschul, S. (2001). Improving the accuracy of PSI-BLAST protein database searches with composition-based statistics and other refinements. *Nucleic Acids Research*, 29(14), 2994–3005.

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

Speed, T. & Waterman, M. S., Eds. (1996). *Genetic Mapping and DNA Sequencing*. New York: Springer-Verlag.

Stone, C. (1996). *A Course in Probability and Statistics*. Belmont, CA: Wadsworth Publishing Company.

Storey, J. & Siegmund, D. (2001). Approximate p-values for local sequence alignments: numerical studies. *Journal of Computational Biology*, 8(5), 549–556.

Thompson, J., Higgins, D., & Gibson, T. (1994). 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, 4673–4680.

Walsh, J. (1923). A closed set of normal orthogonal functions. *Amer. J. Math.*, 45, 5–24.

Waterman, M. S. (1984). General methods of sequence comparison. *Bulletin of Mathematical Biology*, 46(4), 473–500.

Waterman, M. S., Ed. (1989). *Mathematical Methods for DNA Sequences*. Boca Raton, Florida: CRC Press, Inc.

Waterman, M. S. (1995). *Introduction to Computational Biology: maps, sequences, and genomes*. Boca Raton, Florida: Chapman & Hall/CRC.

Waterston, R., Lander, E., & Sulston, J. (2002). On the sequencing of the human genome. *Proc. Natl. Acad. Sci. USA*, 99(6), 3712–3716.

Winograd, S. (1978). On computing the discrete Fourier transform. *Math. Comput.*, 32(141), 175–199.

Zambrowicz, B. P. & Sands, A. T. (2003). Knockouts model the 100 best-selling drugs—will they model the next 100? *Nature Reviews Drug Discovery*, 2, 38–51.

Zhang, Y., Perry, K., & Vinci, V. (2002). Genome shuffling leads to rapid phenotypic improvement in bacteria. *Nature*, 415, 644–646.

Index