BIOINFORMATICS

Vol. 21 Suppl. 1 2005, pages i311–i318
doi:10.1093/bioinformatics/bti1044

Computing the *P*-value of the information content from an alignment of multiple sequences

Niranjan Nagarajan[1], Neil Jones[2] and Uri Keich[1,*]

[1]Computer Science Department, 4130 Upson Hall, Cornell University, Ithaca, NY 14853, USA and [2]Department of Computer Science and Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA

Received on January 15, 2005; accepted on March 27, 2005

ABSTRACT

Motivation: The efficient and accurate computation of *P*-values is an essential requirement for motif-finding and alignment tools. We show that the approximation algorithms used in two popular motif-finding programs, MEME and Consensus, can fail to accurately compute the *P*-value.

Results: We present two new algorithms: one for the evaluation of the *P*-values of a range of motif scores, and a faster one for the evaluation of the *P*-value of a single motif score. Both exhibit more reliability than existing algorithms, and the latter algorithm is comparable in speed to the fastest existing method.

Availability: The algorithms described in this paper are available from http://www.cs.cornell.edu/~keich

Contact: keich@cs.cornell.edu

1 INTRODUCTION

Finding local similarities among a set of sequences is a common task in computational biology. For example, by finding similarities within a set of promoters from co-regulated genes, one hopes to recover transcription factor binding sites that guide the genes' expression patterns *in vivo*. Given a set of sequences, motif-finding algorithms such as MEME (Bailey and Elkan, 1994) and Consensus (Hertz and Stormo, 1999) return a number of possible alignments in some order of potential biological relevance. A critical part of any such study is for a researcher to discriminate between local alignments that are simply random artifacts of the sample and local alignments that are so improbable by chance that they are likely to be biologically relevant.

An ungapped local alignment of length L of sequences from an alphabet with A letters is typically summarized by its information content, or entropy (Stormo, 2000), as follows. Let n_{ij} denote the number of occurrences of the j-th letter in the i-th column of the alignment, and let n be the number of sequences in the alignment. The entropy score, or information content, of the alignment is defined as

$$I := \sum_{i=1}^{L}\sum_{j=1}^{A} n_{ij} \log \frac{n_{ij}/n}{b_j},$$

where b_j is the background frequency of the j-th letter (typically, b_j is the frequency of the j-th letter in the entire sample).[1] The entropy score for a given column i of the alignment is defined, similarly, as

$$I(i) := \sum_{j=1}^{A} n_{ij} \log \frac{n_{ij}/n}{b_j}.$$

While this score can be used to rank more than one alignment in a given sample, it cannot provide any direct information about an alignment's significance, and in particular cannot be used to compare two alignments of varying L and n. To assess the significance of an alignment with entropy score s_0, we rely on the alignment's P-value, which is the probability of seeing an entropy score of s_0 or better under the assumption that each of the L columns has n letters independently sampled according to the background distribution $\{b_1, \ldots, b_A\}$. If the P-value is near 1, the columns in the alignment are too similar to the background for the pattern to be interesting, but if the P-value is near 0, the alignment suggests a functional site.

Let p denote the probability mass function (pmf) of the column score $I(i)$ under the hypothesis that the column is noise—in the sense that it was sampled from the multinomial distribution described by the background probabilities $\{b_1, \ldots, b_A\}$. Assuming that the entropy score for each of the L columns in the alignment is an independent random variable, the pmf of the alignment's total entropy score I is given by the L-fold convolution of p:

$$p^{*L}(s) := \underbrace{p * \cdots * p}_{L} := \sum_{\substack{(s_1,\ldots,s_L):\\ s_1+\cdots+s_L=s}} p(s_1)\ldots p(s_L). \quad (1)$$

The P-value of an alignment with score s_0 is therefore $\bar{F}^{*L}(s_0) := \sum_{s \geq s_0} p^{*L}(s)$. Unfortunately, to naively compute this requires traversing all $s \geq s_0$, which is prohibitively

*To whom correspondence should be addressed.

[1]Strictly speaking, relative entropy is defined as I/n.

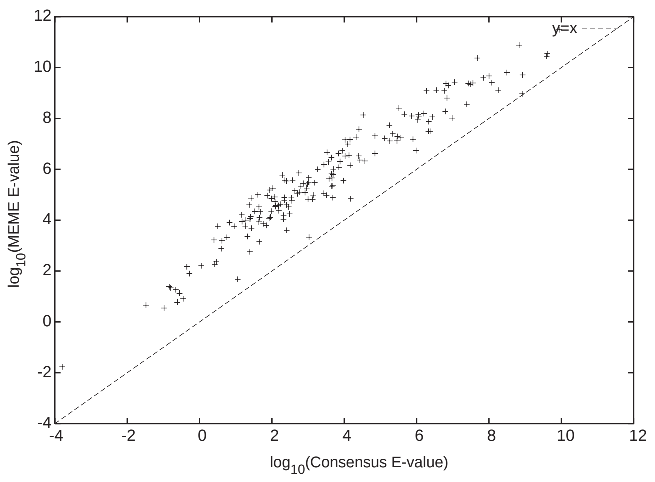

Fig. 1. A comparison of the MEME E-values and Consensus E-values for a number of alignments with $L = 15$ in $n = 20$ sequences of length 1000 each. Since the Consensus E-values are accurate over the range of scores considered here, MEME clearly overestimates the E-value in nearly every case; for alignments with E-values ≤ 1 according to Consensus, MEME may report an E-value as large as 100.

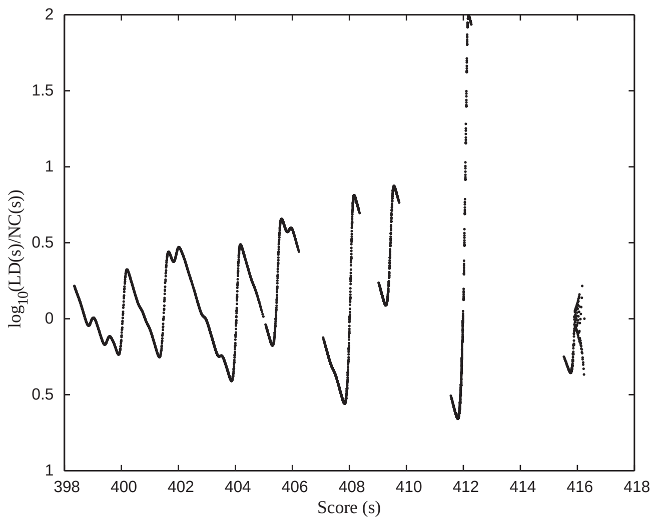

Fig. 2. The graph of $\log_{10}(\mathrm{LD}(s)/\mathrm{NC}(s))$ demonstrates how far off the Consensus-reported P-value may be from the value it estimates. The parameters for this graph are $n = 20$, $A = 4$, $L = 10$ and $b = [0.2497, 0.2499, 0.2501, 0.2503]$. The gaps in the graph indicate areas of unattainable entropy values.

expensive in practice because of the large number of possible values of s. As a result, multiple alignment programs rely on approximations to compute the P-value, striving for a balance between the time spent computing and the accuracy of the result.

To determine whether approximating the P-value computation introduces errors in practice, we modified the source code of MEME (version 3.0.3) and Consensus (version 6c, April 2001) to score arbitrary alignments, bypassing each algorithm's motif-finding step. In this way we are able to compare P-value estimates from different algorithms on a variety of different alignments. Figure 1 shows the result if we plot a point at (x, y) for an alignment with a Consensus E-value of x and a MEME E-value of y. The E-value of an alignment with score s_0 is the expected number of alignments in the sample with the same n and L and with entropy score $\geq s_0$. It can be obtained from the P-value by multiplying by the number of possible alignments in the sample. The MEME E-value is consistently larger than the Consensus E-value (which is reliable in this region) by roughly two orders of magnitude. In at least one case, the true E-value indicates an expectation of 10 alignments with comparable score existing in the sample, while MEME reports an expectation of 5000 alignments; it is conceivable that a researcher would arrive at two different conclusions about the significance of the same alignment by relying on the two estimates. Furthermore, we found at least two alignments of the same size that had inconsistent E-values according to MEME: one alignment had a lower entropy and also a lower E-value than the other (entropy of 13.583, E-value of 1.725×10^7 compared with entropy of 13.617, E-value of 4.1716×10^7), which is clearly a contradiction. Neither the approximation methods discussed in this paper, nor the methods proposed in Hertz and Stormo (1999), demonstrate this instability.

It is important to note that the E-values from Consensus were calculated using an algorithm (LD; see below) that is fast but at times inaccurate. An example of the ratio of Consensus P-values to the true P-values (as calculated by the slower but accurate NC algorithm; see below) is shown in Figure 2. Since the Consensus-reported estimates can be up to two orders of magnitude off, this work introduces a compromise that achieves nearly the accuracy of NC, but at speeds comparable to LD.

Hertz and Stormo (1999) suggest two possible approximation techniques for calculating the P-value. The first, NC, replaces I with its latticed cousin $I_\delta := \lfloor I/\delta \rfloor \cdot \delta$. In this case, the pmf of the single column score $I_\delta(i)$, or p_δ, is therefore also latticed, so the L-fold convolution of p_δ can be done more efficiently than the L-fold convolution of p. A naive algorithm for computing the L-fold convolution on a lattice requires $O(L^2 M^2)$ time, where M is the size of the lattice. Hertz and Stormo (1999) note that using the fast Fourier transform (FFT) to perform the convolution would decrease the running time to $O(LM \log(LM))$; however, the numerical instability of the FFT algorithm tends to wreak havoc on the computation's accuracy for small values, which is exactly the region we are most interested in when searching for motifs. The second method they suggest, LD, uses large deviation theory to estimate the tail of an exponentially shifted probability distribution. In practice this approximation scheme works quite well except for a range of values near the maximal (or minimal) score, where it may be off by an order of magnitude or more. Nevertheless, LD is nearly 200 times faster to compute than NC for $L = 10$, $A = 4$, $n = 100$ and $M = 16\,384$ and is therefore the method used in the popular Consensus tool.

Building on the idea of NC, Keich (2005) proposes the sFFT algorithm to overcome the numerical instability of the FFT for the L-fold convolution step and also delineates explicit bounds on the accuracy of the result. Although this method has lower complexity than NC, it is still somewhat time consuming on large sample sizes. In Section 2, we present improvements to sFFT that give rise to the fastest known algorithm that has accuracy comparable to that of NC. We then describe an optimization, the cyclic shifted FFT technique, to produce the csFFT algorithm, which is more efficient for the computation of a single P-value, with speed comparable to LD.

2 APPROACH

We introduce the shifted-FFT (sFFT) algorithm following the treatment in Keich (2005). The primary bottleneck of the algorithm presented in that paper is the computation of the pmf of one column's entropy score, which we show here can be done much more efficiently.

2.1 The sFFT algorithm

The L-fold convolution of an arbitrary vector $v \in \mathbb{C}^M$, written v^{*L}, can be computed as follows. Let $N = ML$ and extend v to N dimensions by padding it with zeros. The discrete Fourier transform, or DFT, of v is defined as (Press *et al.*, 1992)

$$(Dv)(k) := \sum_{j=0}^{N-1} \mathrm{e}^{ijk\omega} v(j),$$

where $\omega = 2\pi/N$ and $i = \sqrt{-1}$. Define $w \in \mathbb{C}^N$ as $w(k) = [(Dv)(k)]^L$. Then v^{*L} is given by the inverse DFT of w:

$$v^{*L}(l) := (D^{-1}w)(l) := \frac{1}{N} \sum_{j=0}^{N-1} \mathrm{e}^{-ijl\omega} w(j).$$

The straightforward implementations of D and D^{-1} require $O(N^2)$ time, but using a recursive divide-and-conquer strategy results in the FFT which takes $O(N \log N)$ time. If $\widetilde{D}$ and $\widetilde{D^{-1}}$ are the respective implementations of D and D^{-1}, then, owing to numerical errors, $\widetilde{D}$ and $\widetilde{D^{-1}}$ are not exactly the linear and mutually inverse operators that D and D^{-1} are. The DFT of a vector v produces a weighted sum of the entries of v whose magnitudes can span from 10^{-5} to 10^{-70} in a typical application. Since ε_*, the relative machine precision for double-precision arithmetic, is only 10^{-16}, the precision of entries of v with values $<\varepsilon_* \max_j v(j)$ is not preserved for such a sum. Correspondingly, we cannot recover these values by applying $\widetilde{D^{-1}}$.

To avoid the problem of roundoff errors that dominate the smaller P-values, we can emphasize the values of p_δ in the region surrounding s_0 by applying an appropriate exponential shift prior to performing the L-fold convolution. Let

$$p_{\theta,\delta}(s) := p_\delta(s)\mathrm{e}^{\theta s}/M_\delta(\theta), \tag{2}$$

where $M_\delta(\theta) = E\left[\mathrm{e}^{\theta I_\delta(i)}\right]$ is the moment-generating function of the lattice score for one column. This particular form of shifting commutes with the convolution operator, which makes it easy to convert between $p_{\theta,\delta}^{*L}$ and p_δ^{*L}. Note that, as s is latticed, $p_{\theta,\delta}$ is an M-dimensional vector. We will use the notation $p_{\theta,\delta}(j)$ to refer to the j-th entry in that vector, and $p_{\theta,\delta}(s)$ to refer to the value of $p_{\theta,\delta}$ for entropy score s.

Since θ is a parameter, we can choose it in such a way that for a given alignment score s_0 we get the maximal 'resolving power' relative to noise resulting from numerical error in the DFT. Intuitively, the most significant contributions to the P-value should come from values of p_δ^{*L} close to s_0, so we choose to center the mean of the shifted pmf for one column at s_0/L so that $p_{\theta,\delta}^{*L}$ is centered about s_0. This can be satisfied, based on a standard large deviation procedure (Dembo and Zeitouni, 1998), by setting

$$\theta_0 = \arg\min_\theta \left[\log M_\delta(\theta) - \theta s_0/L\right]. \tag{3}$$

Of course, in order to proceed with the convolution, we need an estimate for the pmf of a single column. This could be performed by naively enumerating all possible empirical distributions for a column. Although this approach has the advantage of being the most accurate, it requires $O(n^{A-1})$ time. For small values of A (as is the case for nucleotide sequences) this algorithm is still computationally tractable. However, in our experiments we found that even for small values of n, with $A = 4$, this stage tends to dominate the runtime of the algorithm.

An algorithm with runtime $O(AMn^2)$ to calculate the pmf on a lattice was proposed by Hirji (1997), and later rediscovered by Hertz and Stormo (1999). This particular algorithm produces the pmf over the entire range of possible values in one execution by using dynamic programming. An improvement to the runtime of the algorithm can be obtained by noting that, for small values of n, the number of non-zero lattice points in the intermediate stages of the calculation is small, which allows one to employ a list-based data structure to reduce the runtime to $O(AM'n \log(n))$, where M' is significantly smaller than M in practice (<10 for the parameters in Table 1). The resulting algorithm is more efficient than the original but it still suffers from the overhead of computing with log-values in order to avoid underflows. (Addition of log-values in a C program, for example, was found to be more than 10 times slower than regular addition.)

The underflow conditions in Hirji's algorithm arise because it multiplies and adds terms of the form $r_a(n') = {b_a}^{n'}/n'!$ (where b_a is the background distribution for $a \in [1..A]$ and

$n' \in [0..n]$) that are exponentially small in n'. These terms are used to recursively compute the vector $p_{\delta,a,n'}$, where

$$p_{\delta,a,n'}(j) = \sum_{n''=0}^{n'} r_a(n'') \cdot p_{\delta,a-1,n'-n''}(j - j_a(n'')) \quad (4)$$

$$p_{\delta,1,n'}(j) = \begin{cases} n! \cdot r_1(n') & \text{if } j = j_a(n') \text{ and } n' \in [0..n] \\ 0 & \text{otherwise} \end{cases}$$

and $j_a(n') = \text{round}\left(\delta^{-1} n' \log\left(\frac{n'/n}{b_a}\right)\right)$. As is shown in Hertz and Stormo (1999), $p_\delta(j) = p_{\delta,A,n}(j)$ and so this procedure recovers the pmf p_δ.

In order to avoid the use of logarithms in these computations we design the following procedure: instead of computing with the r_a we shift them to get

$$r'_a(n') = r_a(n') \mathrm{e}^{\delta j_a(n') + n'(\log(n)-1)}$$

and perform the recursion in Equation (4) using r'_a. Let the corresponding result be p'_δ. We can then recover p_δ based on the following claim.

CLAIM 1. $p_\delta(j) = p'_\delta(j)\mathrm{e}^{-\delta j - n(\log(n)-1)}$.

The proof of this claim is based on simple induction using Equation (4) and is therefore omitted.

The shifted computation described above avoids the underflow conditions of Hirji's original algorithm. This is because, even though $r_a(n')$ decreases exponentially with respect to n', $r'_a(n')$ remains approximately $1/\sqrt{2\pi n'}$. For practical values of n this improvement to Hirji's algorithm does not introduce any numerical errors into the result, and in some cases it may be more accurate than relying on logarithms. As can be seen from Figure 3, it also substantially improves the runtime (by more than a factor of 10, on average).

The modified sFFT algorithm is shown in Figure 4. It is important to note that the proof of correctness for the original sFFT algorithm (Keich, 2005) is trivially extended to this case where the complete enumeration of the pmf is replaced with the shifted-Hirji algorithm. Thus, this version of the sFFT algorithm is faster than the original, yet just as reliable.

2.2 The cyclic shifted FFT (csFFT) algorithm

The sFFT algorithm above computes P-values for a range of possible alignment scores, which is wasteful when all we need is a single P-value. Fortunately, most of the mass of the shifted pmf $p^{*L}_{\theta,\delta}$ arises from a restricted range of possible s-values, as Figure 5a suggests. We would like to avoid computing $p^{*L}_{\theta,\delta}$ on those large intervals where it is practically zero. To that end, consider the following cyclic sum of $p^{*L}_{\theta,\delta}$:

$$q(i) = \sum_{\{j : j \bmod M = i\}} p^{*L}_{\theta,\delta}(j). \quad (5)$$

In the example described in Figure 5a, $p^{*L}_{\theta,\delta} \approx 0$ for $j \notin [(L-1)M, LM]$; therefore, it follows that $q(i)$ (where

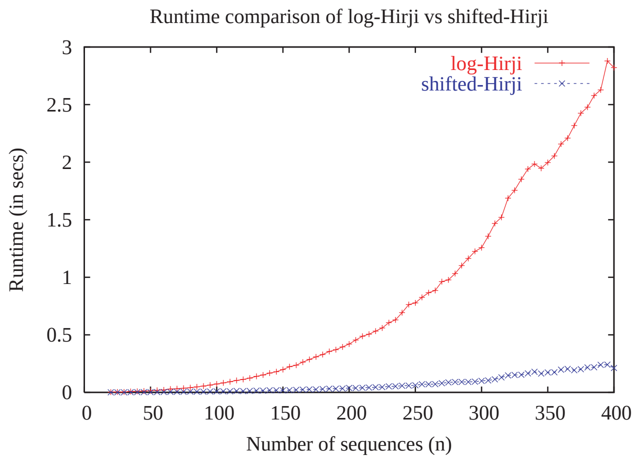

Fig. 3. A comparison of the running times of Hirji's algorithm implemented using log-calculations (log-Hirji) and the shifted 'logless' version described in the text. Here $M = 1024$ and the background distribution is uniform.

The input to sFFT is

- n, the number of sequences
- L, the number of columns in the alignment
- $b_1, \ldots, b_A$, the background frequencies of the A letters
- M, the size of the lattice
- s_0, the observed score.

Given the input, sFFT

1. Computes $\widetilde{p_\delta}$, an estimate of p_δ, by using the shifted (logless) Hirji algorithm.
2. Finds θ_0 by numerically solving Equation (3).
3. Computes $\widetilde{p_{\theta_0,\delta}}(s)$ according to Equation (2).
4. Computes $\widetilde{p^{*L}_{\theta_0,\delta}}$ by applying the FFT-based convolution to $\widetilde{p_{\theta_0,\delta}}(s)$.
5. Computes $\widetilde{p^{*L}_\delta}(j) = \widetilde{p^{*L}_{\theta_0,\delta}}(j)\mathrm{e}^{-\theta_0 j\delta + L\log \widetilde{M_\delta}(\theta_0)}$ for $j_0 \le j \le j_{\max}$, where j_0 and $j_{\max}$ are the lattice indices corresponding to s_0 and the maximum score $s_{\max}$.
6. Returns sFFT$(s_0) := \sum_{j=j_0}^{j_{\max}} \widetilde{p^{*L}_\delta}(j)$.

Fig. 4. The sFFT algorithm.

$i = j \mod M$) approximates $p^{*L}_{\theta,\delta}(j)$ on the interval $j \in [(L-1)M, LM]$. Since q is defined on a lattice of size M rather than on a lattice of size LM, we can immediately save a factor of L, provided q is efficiently computable. Since q is the cyclic convolution of $p_{\theta,\delta}$ it can be efficiently computed by

CLAIM 2. $q = D_M^{-1} w$, where $w(k) = \left[\left(D_M\, p_{\theta,\delta}\right)(k)\right]^L$.

The proof of this claim can be found in Press *et al.* (1992). The difference between the formula above and its non-cyclic

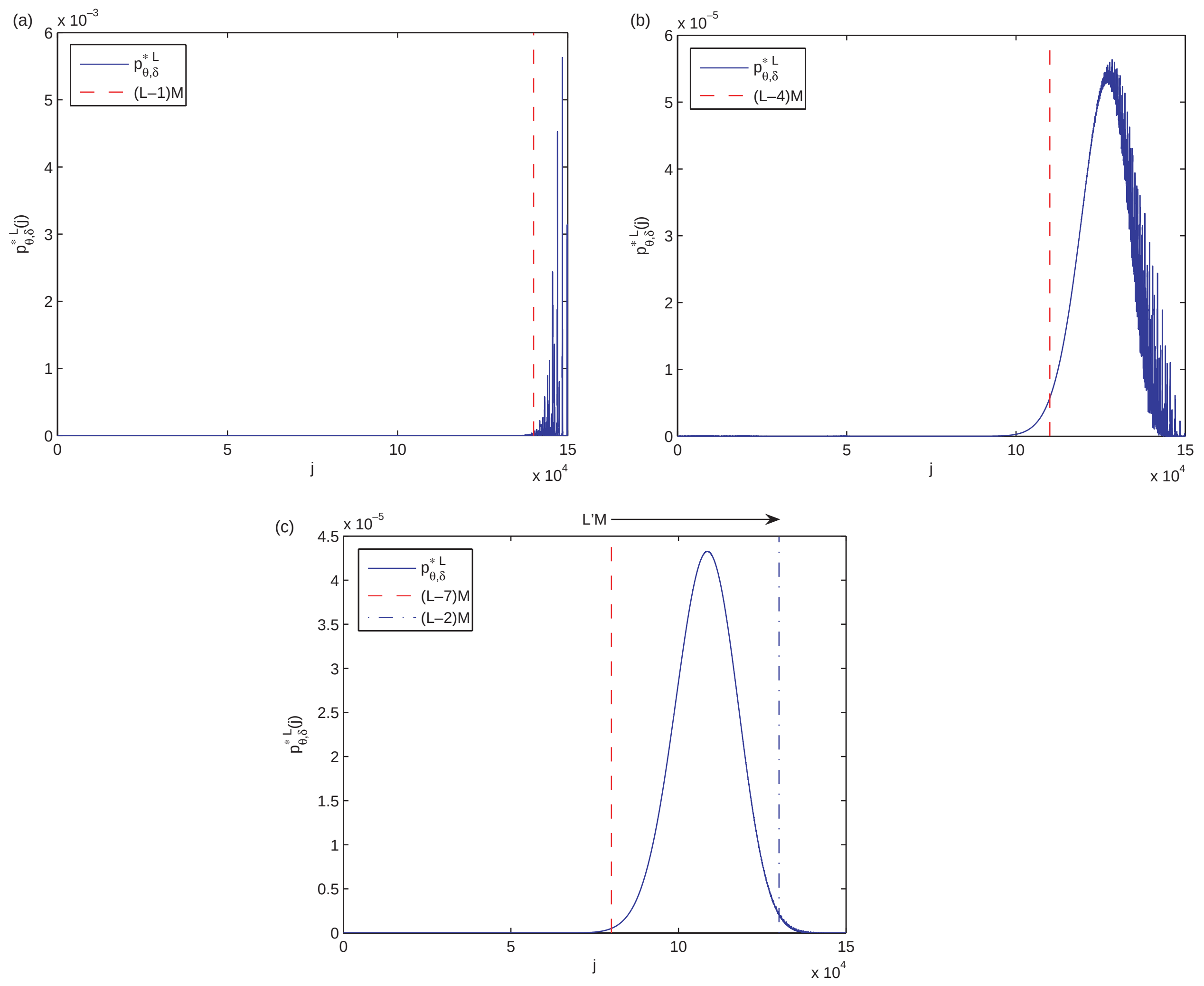

Fig. 5. The convoluted pmf is 0 over much of the range of valid values for s. Here, $M = 10\,000$, $n = 20$, $L = 15$, $b = [0.2499, 0.2501, 0.2497, 0.2503]$. The 'essential support' intervals here are described by indices for the latticed pmf. (**a**) An example where $p^{*L}_{\theta,\delta}$ has its essential support in a narrow interval defined by $[(L-1)M, LM]$; $s_0 = 405$. (**b**) Here, $p^{*L}_{\theta,\delta}$ has its essential support in an interval larger than M, defined by $[(L-4)M, LM]$; $s_0 = 350$. (**c**) The support for $p^{*L}_{\theta,\delta}$ is mainly in an interval of sizes $L'M$ for $L' = 5$, but not of the form $[(L-L')M, LM]$; $s_0 = 350$.

analog is the dimensionality of the DFT operator: here it is M, whereas the DFT operator previously had dimensionality LM with $p_{\theta,\delta}$ appropriately padded with $(L-1)M$ zeros.

More generally, the essential support interval of $p^{*L}_{\theta,\delta}$ may be of size $L'M$ (e.g. Fig. 5b). Such an interval may also not be strictly of the form $[(L-L')M, LM]$; instead it may be centered about s_0 (e.g. Fig. 5c). In this case, rather than directly calculating

$$\bar{F}^{*L}_{\delta}(j_0) := \sum_{j \ge j_0} p^{*L}_{\delta}(j),$$

we approximate it with

$$\hat{F}^{*L}_{\theta}(j_0) := \sum_{j_0 \le j \le J} q(j) e^{-\theta_0 j\delta + L \log M(\theta_0)},$$

where $J = \min(j_0 + L'M/2, j_{\max})$. This is justified by

$$\begin{aligned} &\sum_{j_0 \le j \le J} q(j) e^{-\theta_0 j\delta + L \log M(\theta_0)} \\ &\approx \sum_{j_0 \le j \le J} p^{*L}_{\theta,\delta}(j) e^{-\theta_0 j\delta + L \log M(\theta_0)} \\ &\approx \sum_{j \ge j_0} p^{*L}_{\theta,\delta}(j) e^{-\theta_0 j\delta + L \log M(\theta_0)}. \end{aligned}$$

An appropriate choice of L' would ensure that, say, 95% of the mass of $p^{*L}_{\theta,\delta}$ lies in the interval of size $L'M$ centered about j_0. This would be relatively easy if we had an explicit function for $p^{*L}_{\theta,\delta}$, but this is exactly the function we are trying to estimate. Instead, we rely on the following formula for L', under the assumption that $p^{*L}_{\theta,\delta}$ is roughly normally distributed (an assumption made by the LD algorithm):

$$L' := \left\lceil \frac{k_\sigma \sqrt{L}\sigma_\theta}{M} \right\rceil. \tag{6}$$

Here, $\sigma_\theta^2 :=$ Var $p_{\theta,\delta}$, where $p_{\theta,\delta}$ is the integer lattice version of Equation (2) and the variance is computed on the lattice indices. Note that $k_\sigma \sqrt{L}\sigma_\theta = k_\sigma \sqrt{\mathrm{Var}(p^{*L}_{\theta,\delta})}$, so an interval of size $L'M$ centered about j_0 extends roughly $k_\sigma/2$ standard deviations on each side. Thus, if we arbitrarily set $k_\sigma := 4$, then Equation (6) roughly ensures the desired 95% condition under the assumption of normality.

2.3 Boosting θ when $s_0 \to s_{\max}$

As observed in Hertz and Stormo (1999), when s approaches $s_{\max}$, θ increases while σ_θ decreases. Thus, for s_0 close to $s_{\max}$, if we increase or boost θ beyond the computed $\theta_0 = \theta(s_0)$ from Equation (3), we reduce σ_θ. Since L' depends linearly on σ_θ [from Equation (6)], such boosting can effectively decrease the runtime by reducing L'. Another reason to boost θ for s near $s_{\max}$ is that it reduces the error introduced by approximating sFFT with the cyclic sum in csFFT, as shown next.

Claim 3. *Let $d = L'M$ and $j' \equiv j \mod d$. Then*

$$\hat{F}^{*L}_\theta(j_0) - \bar{F}^{*L}_\delta(j_0) \le \sum_{j_0 \le j \le J} \sum_{j' < j} p^{*L}_\delta(j')\mathrm{e}^{-\theta_0(j-j')\delta} \tag{7}$$

$$+ \sum_{j_0 \le j \le J} \sum_{j' > j} p^{*L}_\delta(j')(\mathrm{e}^{-\theta_0(j-j')\delta} - 1) \tag{8}$$

$$\bar{F}^{*L}_\delta(j_0) - \hat{F}^{*L}_\theta(j_0) \le \sum_{j_0 + d/2 < j \le J} \sum_{j' > j} p^{*L}_\delta(j'). \tag{9}$$

The proof of the claim is straightforward from the definitions and is thus omitted.

Suppose s is sufficiently close to $s_{\max}$ that $j_0 + d/2 > j_{\max}$. In that case the right-hand side of Equation (9) vanishes, leaving $\hat{F}^{*L}_\theta(j_0)$ as an upper bound of the P-value, $\bar{F}^{*L}_\delta(j_0)$. Moreover, term (8) vanishes as well, and we are left with

$$0 \le \hat{F}^{*L}_\theta(j_0) - \bar{F}^{*L}_\delta(j_0) \tag{10}$$

$$\le \sum_{j_0 \le j \le j_{\max}} \sum_{j' < j} p^{*L}_\delta(j')\mathrm{e}^{-\theta_0(j-j')\delta}. \tag{11}$$

This upper bound on the error decreases as θ_0 increases, which supports our assertion that boosting θ is beneficial for s close to $s_{\max}$. One might be tempted to boost θ by a large amount, but although this would indeed reduce the error in Equation (11) it would have the unfortunate side-effect of increasing the numerical errors in the FFT [discussed at length in Keich (2005)].

An intermediate solution is to boost θ by adding

$$\theta_{\text{boost}} = \log(10^9)/[(j_{\max} - j_0)\delta].$$

This solution can boost θ significantly and bring corresponding savings in runtime, as well as reduce the error in Equation (11). It is also designed (based on some assumptions about $p^{*L}_{\theta,\delta}$) to still preserve the important entries of $p^{*L}_{\theta,\delta}$ (for computing the P-value) during the FFT. Finally, although this solution is heuristic, it works well in practice, as is shown in Sections 3.1 and 3.2.

The csFFT algorithm with boosting is shown in Figure 6. For typical values of L, the csFFT algorithm is simultaneously more accurate than and comparable in speed to LD.

3 ANALYSIS

3.1 Runtime characterization

Assuming that the time-limiting step of sFFT is the calculation of the FFT itself, csFFT is roughly L/L' times faster than the sFFT algorithm described in the previous section. Interestingly, the saving of L/L' varies with s_0: the speed-up for values of s_0 near the center of the distribution is modest, wheras the best gains occur near the ends of the range of possible s-values. This follows from the fact that as s_0 approaches $s_{\max}$ (or $s_{\min}$), the corresponding σ_θ goes to 0, yielding a smaller L' in Equation (6). In any case, the complexity of csFFT is lower than that of sFFT: by Equation (6) the complexity of the FFT step is now $O(\sqrt{L}M \log(\sqrt{L}M))$.

We conducted tests to verify that csFFT is indeed more efficient than sFFT. Since sFFT and csFFT differ mainly in the convolution step of the algorithm, where the running times are roughly linear in L and L', respectively, we focus on the growth of L' in terms of L. Figure 7a demonstrates that, if we take the average value of L' over the range of s values, it grows roughly as $\sqrt{L}$ when all other parameters are fixed. In addition, the average value of L' is roughly constant for different b_a from Table 1 (based on a test with $L = 10$ and $n = 10$) and decreases as n increases[2] (see Figure 7b). Furthermore, we found that boosting, when it is applicable, gives substantial runtime gains, halving the runtime in many cases. Finally, since csFFT relies on a substantially faster convolution than sFFT, we found that for tests with large n, small L and s close to $s_{\max}$ the runtime of the algorithm is no longer dominated by the time for the convolution. For example, in a test case with $n = 20$, $L = 15$, $b_a =$ uniform, $M = 16\,384$ and $s = 380$, csFFT takes 0.09 s to compute the answer, of which 0.01 s is spent in the shifted-Hirji step, 0.07 s is spent computing the shift and 0.01s is required for the FFT ($L' = 2$). We are currently working on techniques to reduce the time spent in computing the shift for such examples.

[2]In practice the runtime increases with n as we have to increase M proportionally to maintain the granularity of the lattice.

Table 1. Range of test parameters

Parameter	**Values**
L	5, 10, 15, 30
n	2, 5, 10, 15, 20, 50
b_a	Uniform, sloped, blocked, perturbed uniform

Uniform refers to the case where $b = [0.25, 0.25, 0.25, 0.25]$, sloped refers to $b = [0.1, 0.2, 0.3, 0.4]$, blocked refers to $b = [0.2, 0.2, 0.3, 0.3]$ and perturbed uniform refers to $b = [0.2497, 0.2499, 0.2501, 0.2503]$.

Table 2. Runtime comparison between csFFT and LD

n	L	s	Runtime for csFFT (s)	Runtime for LD (s)
40	5	200	0.04	0.06
15	5	100	0.01	0.01
15	30	600	0.05	0.01
40	30	600	0.12	0.06
40	5	260	0.02	0.06

The comparisons were made using the uniform b_a (Table 1) and with Q set to 16 384

The input to csFFT is

- n, the number of sequences
- L, the number of columns in the alignment
- $b_1, \ldots, b_A$, the background frequencies of the A letters
- M, the size of the lattice
- s_0, the observed score

Given the input, csFFT

1. Computes $\widetilde{p_\delta}$, an estimate of p_δ, by using the shifted (logless) Hirji algorithm.
2. Finds θ_0 by numerically solving Equation (3).
3. Computes L' according to Equation (6) and using the default $k_\sigma = 4$.
4. Boosts θ_0 if $j_0 + L'M/2 > j_{\max}$.
5. Computes $\widetilde{p_{\theta_0,\delta}}(s)$ according to Equation (2).
6. Computes $\widetilde{p^{*L}_{\theta_0,\delta}}$ by applying the FFT-based cyclic-convolution to $\widetilde{p_{\theta_0,\delta}}(s)$ with period $L'M$.
7. Computes $\widetilde{p^{*L}_{\delta}}(j) = \widetilde{p^{*L}_{\theta_0,\delta}}(j)e^{-\theta_0 j\delta + L\log \widetilde{M_\delta}(\theta_0)}$ for $j_0 \leq j \leq J$.
8. Returns $\text{csFFT}(s_0) := \sum_{j=j_0}^{J} \widetilde{p^{*L}_{\delta}}(j)$.

Fig. 6. The csFFT algorithm for computing the P-value of an entropy score s_0.

3.2 Error analysis

For each combination of parameter values in Table 1 we tested >100 roughly evenly spaced values for s, and separately another set of 100 points lying in the tail of the pmf. Because we have latticed s, the P-value of s has an inherent lattice error, as discussed in Keich (2005). For any given value of s, sFFT(s) and csFFT(s) fall within a small range; the true value falls somewhere in between the minimum and maximum values in that range. The bounding interval for the P-values computed by csFFT $(\text{comp}_{\min}, \text{comp}_{\max})$ was then compared with the provably reliable bounding interval from sFFT $(\text{real}_{\min}, \text{real}_{\max})$. In the cases where these intervals do not overlap we report an error ratio of

$$\min(|\text{comp}_{\max}/\text{real}_{\min}|, |\text{real}_{\max}/\text{comp}_{\min}|).$$

In all cases that we tested, we found that the error ratio is at most 1.1, corresponding to at most 10% error. The cases with the worst error ratios were usually found to be for values of s close to the average of the pmf, where the P-values are large and therefore not very relevant in most applications. In addition, for the values tested in the tail of the pmf, the error ratio indicated that the numerical error in the method was smaller than the lattice error.

3.3 Stitching LD and csFFT

The csFFT algorithm is simultaneously more accurate than and comparable in speed to LD. For example, for $L = 10$, $n = 100$, $b_a =$ uniform, $M = 16\,384$ and $s = 380$, Consensus's P-value computation required 0.32 s, while csFFT required 0.20 s with $L' = 3$. Admittedly, this is a somewhat biased example as $n = 100$ is probably larger than typical problems. For the example in the previous section, on the other hand, LD is faster by a factor of 4. We present a few more examples in Table 2. In general, LD is faster than csFFT for small n and large L and also for values of s that are away from the tail with larger L'. We can exploit this by designing a heuristic rule that switches to LD for appropriate values of n and L. In designing a switching criterion we also need to consider the approximation errors inherent to LD; an example is given in the introduction in which LD gives a very poor approximation. Hertz and Stormo (1999) present an empirical test that can be used to gauge the reliability of the LD-based normal approximation. Essentially, if s is $<3\sigma$ (σ is SD (of the shifted pmf)) from the tail of the pmf, then the normal approximation is no longer reliable. We calculated the observed error of the LD method in the range defined by this test for the set of parameters in Table 1 and found that in all cases the error ratio was <1.24, corresponding to <24% error. We can therefore use this test in conjunction with csFFT to yield an algorithm that is efficient and accurate over a larger range of n and L values.

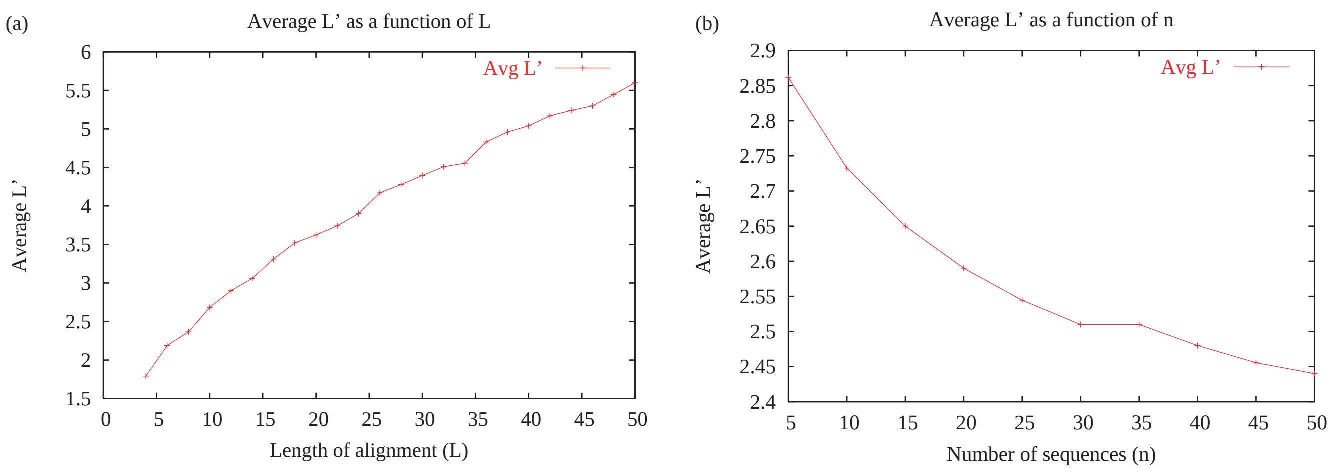

Fig. 7. Average values of L' versus (**a**) L and (**b**) N using the perturbed uniform b_a (Table 1). (a) $n = 10$ and $M = 16384$ and (b) $L = 10$ and $M = 16384$.

4 CONCLUSION

Accurate methods for estimating the P-value of an alignment score are critical in aiding the discovery of biologically meaningful signals from sets of related sequences. Although existing tools provide estimates, it is clear that some estimates are better than others. The method employed by MEME is overly pessimistic about an alignment, which could conceivably lead to missed signals. Although the method used by Consensus is more accurate, it can still improperly estimate the P-value.

Two methods were presented in this paper. Althogh the first (sFFT) is not quite as fast as LD, it is significantly faster than NC, has bounded error estimates and returns P-values for a range of entropy scores. The second (csFFT) is comparable in speed to LD and is empirically more accurate, but like LD returns a P-value for only a single entropy score.

The algorithms described in this paper are available from http://www.cs.cornell.edu/~keich. They provide a general method for the computation of P-values and we believe that it is possible for a researcher to use these algorithms as either an integral part of, or as a post-processing step, to currently existing motif-finding algorithms. Extending these methods to account for gapped alignments is, however, an important and interesting topic for future research. The methods described in this paper can also be used for applications other than motif-finding. These tools may be helpful wherever a statistical significance of a multiple alignment is desired, for example in the problem of profile–profile alignment or in the analysis of protein families.

ACKNOWLEDGEMENT

The authors would like to thank Pavel Pevzner for constructive initial direction.

REFERENCES

Baglivo,J., Olivier,D. and Pagano,M. (1992) Methods for exact goodness-of-fit tests. *J. Am. Stat. Assoc.*, **87**, 464–469.

Bailey,T.L. and Elkan,C. (1994) Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, Menlo Park, CA, pp. 28–36.

Hertz,G.Z. and Stormo,G.D. (1999) Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, **15**, 563–577.

Hirji,K.A. (1997) A comparison of algorithms for exact goodness-of-fit tests for multinomial data. *Comm. Stat.—Simulation and Comput.*, **26**, 1197–1227.

Keich,U. (2005) Efficiently computing the P-value of the entropy score. *J. Comput. Biol.* (In press).

Keich,U. and Nagarajan,N. (2004) A faster reliable algorithm to estimate the P-value of the multinomial llr statistic. In *Proceedings of the Fourth Workshop on Algorithms in Bioinformatics (WABI-04)*.

Press,W.H., Teukolsky,S.A., vetterling,W.T. and Flannery,B.P. (1992) *Numerical Recipes in C. The Art of Scientific Computing*, 2nd edn. Cambridge University Press, Cambridge, UK.

Stormo,G.D. (2000) DNA binding sites: representation and discovery. *Bioinformatics*, **16**, 16–23.

Dembo,A. and Zeitouni,O. (1998) *Large Deviation Techniques and Applications*, 2nd edn. Springer-Verlag, NY.