A Faster Reliable Algorithm to Estimate the p-Value of the Multinomial llr Statistic

Uri Keich and Niranjan Nagarajan

Department of Computer Science, Cornell University, Ithaca, NY-14850, USA
{keich,niranjan}@cs.cornell.edu

Abstract. The subject of estimating the p-value of the log-likelihood ratio statistic for multinomial distribution has been studied extensively in the statistical literature. Nevertheless, bioinformatics laid new challenges before that research by often concentrating its interest on the "thin tail" of the distribution where classical statistical approximation typically fails. Hence, some of the more recent development in this area have come from the bioinformatics community ([5], [3]).
Since algorithms for computing the exact p-value have an exponential complexity, the only generally applicable algorithms for reliably estimating the p-value are lattice based. In particular, Hertz and Stormo have a dynamic programming algorithm whose complexity is $O(QKN^2)$, where Q is the size of the lattice, K is the size of the alphabet and N is the size of the sample. We present a new algorithm that is practically as reliable as Hertz and Stormo's and has a complexity of $O(QKN \log N)$. An interesting feature of our algorithm is that it can guarantee the quality of its estimated p-value.

1 Introduction

The subject of goodness-of-fit tests in general and of using the (generalized) log-likelihood ratio (*llr*) statistic, in particular, is of great importance in applications of statistics. In many applications, an important question to answer is how unlikely is it that an observed sample came from a particular multinomial distribution (H_0)? But in order to answer this question, we first need to quantify the similarity level between the observed sample distribution and the null distribution. The llr statistic, G^2 (defined below) is a popular measure as it is provably optimal under some conditions. Indeed, it is so popular that it has several other names which are more commonly used in the information theory and bioinformatics community: entropy distance, relative entropy, information content, Kullbak-Leibler divergence etc., all of which (upto a factor of N) stand for $I = G^2/2$, where

$$I = \sum_k X_k \log \left(X_k/(N\pi_k)\right) \quad [1],$$

for a null multinomial distribution $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_K)$ and a random sample $\boldsymbol{X} = (X_1, \ldots, X_K)$ of size $N = \sum_k X_k$. Note that $I = 0$ if and only if the empirical

[1] One can readily show that $G^2 = 2I$ is a generalized llr (e.g. [12]).

© Springer-Verlag Berlin Heidelberg 2004

distribution is identical to $\boldsymbol{\pi}$ which is to be expected from something that is supposed to measure the distance (not in a metric sense) between these distributions.

The question of how unlikely is it that the particular sample $\boldsymbol{n} = (n_1, \ldots, n_K)$ of size $N = \sum n_k$ came from $\boldsymbol{\pi}$ can then be translated to the following p-value that we need to compute:

$$P_{H_0}\left(I \geq \sum_k n_k \log \frac{n_k}{N\pi_k}\right),$$

or more generally, given an observed score s, what is $P_{H_0}(I \geq s)$? The latter question has been studied extensively in the statistical literature and has several types of estimates and means of computation which we survey below before describing our own novel technique.

The first type of estimates are in the form of universal upper and lower bounds such as the following one from Hoeffding [7]:

$$c_0 N^{-(K-1)/2} \exp(-s) \leq P(I \geq s) \leq \binom{N+K-1}{K-1} \exp(-s), \qquad (1)$$

where c_0 is a positive absolute constant which can be taken to be $1/2$. Kallenberg has provided sharper (and more complicated) bounds [8] but he added that "...the bounds are not intended as direct numerical approximations of the involved probabilities ... they are useful because they have the right order of magnitude" and this will be relevant to us later on.

We can obtain asymptotically correct estimates based on the result that keeping $\boldsymbol{\pi}$ fixed and letting the sample size $N \to \infty$,

$$P_{H_0}(G^2 \geq s) \longrightarrow \chi^2_{K-1}(s)$$

(e.g. [12]). The rate of convergence and various corrections have been studied and are discussed in [4]. While the χ^2 approximation is a valid asymptotic result, in a typical application N is fixed and as s approaches the tail of the distribution the approximation can be quite poor. For example, for a null distribution of $\pi_i = i/10$ with $i = 1, \ldots, 4$ and $N = 40$, the p-value of $s = 120$ is roughly 7.8e-27 while the χ^2 approximation yields 7.7e-26, a factor of 10 off (and all else being equal, as s grows this will become worse).

Algorithmically, the simplest approach to computing the p-value is by naively enumerating all possible empirical distributions. However, the number of possible distributions grows like $\binom{N+K-1}{N}$, thus giving us a $O(N^{K-1})$ algorithm. Aware of these problems, Baglivo et al. [1] designed a polynomial time algorithm to approximate the p-value using a lattice. In principle the lattice can also be used to guarantee the quality of the approximation. However Baglivo et al.'s Algorithm employs the DFT (discrete Fourier transform) [10] and is therefore prone to exceedingly large numerical errors which originate from the inherent roundoff errors that would accompany any implementation of the DFT[6].

In bioinformatics I is heavily used in the context of evaluating the quality of an ungapped multiple sequence alignment [13] as in the popular motif-finder programs Meme [2] and Consensus [5]. Other usages were recently surveyed in [3]. Given the typical size of bioinformatics data, we are often forced to deal with exceedingly small p-values for which the χ^2 approximation breaks down. Thus, it should be of no surprise that some of the advancements in this area came from this community. Hertz and Stormo [5] provide a dynamic programming algorithm which, similar to Baglivo et al., uses a lattice approximation of the p-value. Both algorithms have a complexity of $O(QKN^2)$, where Q is the size of the lattice. However, Hertz and Stormo's algorithm has a much better handle of the numerical errors and by and large their algorithm is accurate to the mesh of the lattice. A slight modification of this algorithm is implemented as part of Meme's statistical evaluation of its results (version 3.0.3).

More recently Bejerano [3] introduced a new branch and bound algorithm to find the *exact* p-value. Since this approach does not use a lattice and is also a numerically stable algorithm, in general, it yields the most accurate result. However, it is only suitable for small Ks as it exhibits an exponential behavior in K. For $K = 4$ it has a runtime of the order of N^2 and in general it seems to have a runtime function that has the order of N^{K-2}. In another recent work Rahmann [11] apparently re-discovered Hertz and Stromo's dynamic programming method but in addition the paper also includes a clearer exposition of the problem and the algorithm. The paper also makes the important observation that in order to preserve accuracy, Q has to be increased linearly with N (assuming fixed π).

In this paper, we present a new algorithm that yields a lattice approximation of the p-value, $P_{H_0}(I \geq s)$ in $O(QKN \log N)$ time. We start with Baglivo et al.'s Algorithm and modify it using a technique we recently developed in [9] to control the numerical errors in FFT (fast Fourier transform) based convolutions. An interesting feature of our algorithm is that it provides a fairly reliable (and useful) upper bound on the numerical error in our estimate for the p-value.

2 Baglivo et al.'s Algorithm

Instead of computing the pmf (probability mass function) of I, Baglivo et al. suggest that we compute the pmf of the lattice valued random variable I_Q which approximates I, where

$$I_Q = \sum_k \text{round}\left[\delta^{-1} X_k \log(X_k/(N\pi_k))\right] \; {}^2.$$

Here $\delta = \delta(Q) = I_{\max}/(Q-1)$ is the mesh size, $I_{\max} = N \log \pi_{\min}^{-1}$ is the maximal entropy and $\pi_{\min} = \min\{\pi_k\}$. By estimating p_Q, the pmf of I_Q, we can use

$$\sum_{\lceil s/\delta + K/2 \rceil} p_Q(j) \leq P(I \geq s) \leq \sum_{\lfloor s/\delta - K/2 \rfloor} p_Q(j).$$

to get a good estimate of $P(I \geq s)$ (assuming that the lattice is fine enough.)

[2] Note that due to rounding effects I_Q might be negative but we shall ignore this as the arithmetic we perform is modulo Q. The concerned reader can redefine $\delta = I_{\max}/(Q - 1 - \lceil K/2 \rceil)$.

In order to compute p_Q Baglivo et al.'s Algorithm starts by computing the DFT of p_Q, $\varPhi = Dp_Q$:

$$\varPhi(l) = \sum_{j=0}^{Q-1} p_Q(j) e^{i\omega_0 jl} \qquad \text{for } l = 0, 1, \ldots, Q-1,$$

where $\omega_0 = 2\pi/Q$. Once we have $\varPhi$, we can recover p_Q by applying D^{-1}, the inverse-DFT:

$$p_Q(j) = (D^{-1}\varPhi)(j) = \frac{1}{Q}\sum_{l=0}^{Q-1} \varPhi(l) e^{-i\omega_0 lj}.$$

At first glance this seems like a page out of the adventures of Baron Munchausen since after all we need p_Q in order to compute $\varPhi$ to begin with. However, there is an alternative way to compute $\varPhi$ as we outline next. As is explained in [1], we know that

$$\varPhi(l) = \frac{1}{P(X_+ = N)} \sum_{\boldsymbol{x}\in\mathbb{Z^+}^K : \sum x_j = N} \prod_{j=1}^{K} p_j(x_j) e^{i\omega_0 l s_j(x_j)} = \frac{\psi_K(N,l)}{P(X_+ = N)},$$

where X_+ is a Poisson $\lambda = N$ random variable, p_k is the Poisson $\lambda = N\pi_k$ pmf and $s_k(y) = \text{round}[\delta^{-1} y \log(y/N\pi_k)]$ (the contribution to I_Q from the k-th letter appearing y times). It is not difficult to check that ψ_k actually satisfies the following recursive formula [1]:

$$\psi_k(n,l) = \sum_{x=0}^{n} p_k(x) e^{il\omega_0 s_k(x)} \psi_{k-1}(n-x, l). \tag{2}$$

Thus using (2) $\varPhi(l)$ can be recovered in $O(KN^2)$ steps for each l separately and hence $O(QKN^2)$ steps overall. Finally, using an FFT implementation of DFT [10] they get an estimate of p_Q in an additional $O(Q \log Q)$ steps (which should typically be absorbed in the first term). However, as we mentioned earlier, the algorithm as it is has a serious limitation in that numerical errors introduced by the use of the FFT can quickly become dominant in the calculations. An example of this phenomena can be observed with the parameter values, $Q = 8192$, $N = 100$, $K = 20$ and $\pi_i = 1/20$, where Baglivo et al.'s Algorithm yields a *negative* p-value for $P(I \geq 40)$.

3 Our Algorithm 1.0

To reduce the often unacceptable level of numerical errors in Baglivo et al.'s Algorithm we follow [9] and apply an exponential shift to p_Q. A simple example can help explain the idea. Let $p(x) \propto e^{-x}$ for $x \in \{0, 1, \ldots, 255\}$. In Figure 1 we compare p with $q = \widetilde{D^{-1}}(\widetilde{D}p)$, where $\widetilde{D}$ and $\widetilde{D^{-1}}$ are the machine implemented FFT and inverse FFT operators. As can be seen, while theoretically equal, in practice the two differ significantly. Now, if we apply an exponential shift to p prior to invoking the FFT operators then we get $\max_x |\log q_\theta(x)/p(x)| < 1.78 \cdot 10^{-15}$, where $\theta = 1$, $p_\theta(x) = p(x)e^{\theta x}$ and $q_\theta(x) = \left(\widetilde{D^{-1}}\left(\widetilde{D}p_\theta\right)\right)(x) \cdot e^{-\theta x}$. So p is recovered almost up to machine precision

($\varepsilon_0 \approx 2.2 \cdot 10^{-16}$). The reason this works is that by applying the correct exponential shift we "flatten" p so that the smaller values are not overwhelmed by the largest ones during the computation of the fourier transforms.

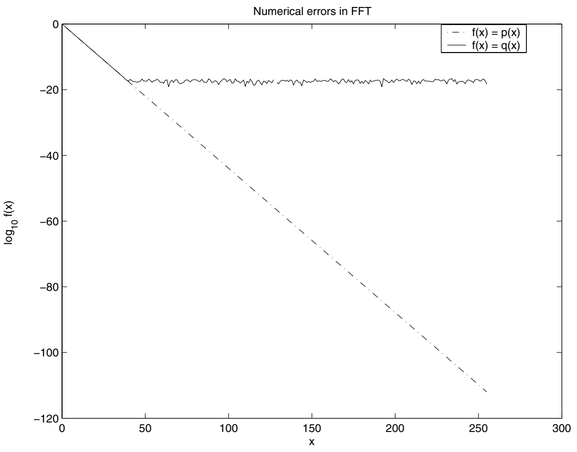

Fig. 1. The destructive effects of numerical roundoff errors in FFT.

This figure illustrates the potentially overwhelming effects of numerical errors in applications of FFT. $p(x) \propto e^{-x}$ for $x \in \{0, 1, \ldots, 255\}$ is compared with what should (in the absence of numerical errors) be the same quantity: $q = \widetilde{D^{-1}}(\widetilde{D}p)$, where $\widetilde{D}$ and $\widetilde{D^{-1}}$ are the machine implemented FFT and inverse FFT operators, respectively. This dramatic difference all but vanishes when we apply the correct exponential shift prior to applying D.

Needless to say this exponential shift will not always work. However, we do know that "to first order" our p-value behaves like e^{-s} (with fixed N and K) as is evident from (1). This suggests that we would benefit from applying an exponential shift to p_Q. Let

$$p_\theta(j) = \frac{p_Q(j) e^{\theta \delta j}}{M(\theta)},$$

where $M(\theta) = E e^{\theta I_Q}$ is the MGF (moment generating function) of I_Q. Figure 2 shows an example of the flattening effect such a shift has on p_Q. Note that for a given θ, $M(\theta)$ can be reliably estimated in $O(KN^2)$ steps by replacing $e^{il\omega_0 s_k(x)}$ with $e^{\theta s_k(x)}$ in (2) and essentially repeating Baglivo's et al. procedure (but for a single value of θ).

The discussion so far implicitly assumed that we know p_Q which of course we do not. Nevertheless, we can compute $\Phi_\theta = D p_\theta$ by slightly modifying the algorithm of Baglivo et al. All we need to do is replace the Poisson pmfs p_k with a shifted version $p_{k,\theta}(x) = p_k(x) e^{\theta s_k(x)} / M(\theta)^{\pi_k}$ and also replace ψ_k with the obvious $\psi_{k,\theta}$.

$$\psi_{k,\theta}(n, l) = \sum_{x=0}^{n} p_{k,\theta}(x) e^{il\omega_0 s_k(x)} \psi_{k-1,\theta}(n - x, l). \tag{3}$$

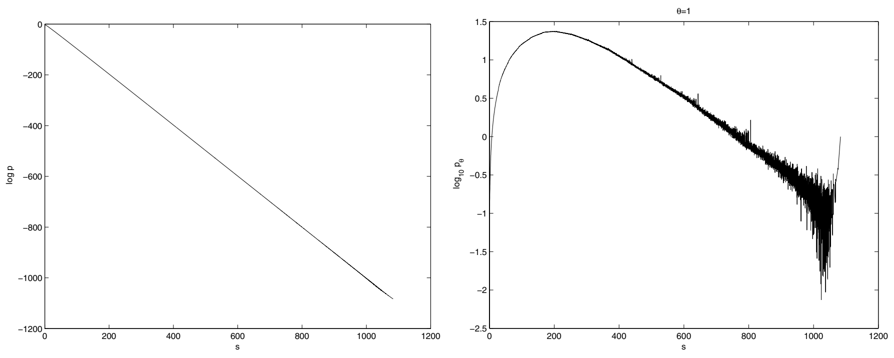

Fig. 2. How can an exponential shift help?

The graph on the left is that of $\log p_Q(s/\delta)$ where $N = 400$, $Q = 8192$ and $\pi_k = k/15$ for $k = 1, \dots, 5$. The graph on the right is of the log of the shifted pmf, $\log p_\theta(s/\delta)$ where $\theta = 1$. Note the dramatic flattening effect of the exponential shift (keeping in mind the fact that the scales of the y-axes are different).

This allows us to compute $\widetilde{\Phi_\theta}(l)$, an estimate of[3] $\Phi_\theta(l) = \psi_{K,\theta}(N, l)/P(X_+ = N)$ in the same $O(KN^2)$ steps for each fixed l. We then compute an estimate $\widetilde{p_Q}$ of p_Q based on $p_Q(j) = \left(D^{-1}\Phi_\theta\right)(j)e^{-\theta\delta j}M(\theta)$. It is important to note that should the need arise (for example, for estimating very small p-values in the tail of the distribution) we can just as well estimate $\log p_Q(j) = \log p_\theta(j) - \theta\delta j + \log M(\theta)$ which significantly extends the range of p-values that can be computed by our algorithm. Finally, the p-value is estimated by $\sum_{j \ge s/\delta} \widetilde{p_Q}(j)$ (or the logarithmic version of that summation).

We are still left with the question of which θ to use. Equation (1) suggests $\theta = 1$ and indeed it typically yields the widest range of js for which $\widetilde{p_Q}(j)$ provides a "decent" approximation of $p_Q(j)$. However, for a given $s_0 = j_0\delta$ there would typically be a better choice of θ. Intuitively, we want to center p_θ about s_0 and that is the case with $\theta_{s_0} = \operatorname{argmin}_\theta \left[-\theta s_0 + \log M(\theta)\right]$. The minimization procedure can be carried out numerically[4] by using, for example, Brent's method [10] which would cost us another $O(KN^2)$ (but is outside the main loop on l). Finally, the next claim whose technical proof is outlined in the appendix gives an upper bound on our error.

Claim 1

$$|\widetilde{p_Q}(j) - p_Q(j)| \le C(NK + \log Q)\varepsilon_0 e^{-\theta\delta j + \log M(\theta)}, \tag{4}$$

where C is some small universal constant and ε_0 is the machine precision.

Note that for $s_0 = \delta j_0$ this upper bound is exactly minimized for θ_{s_0}, thus giving us another justification for our choice of θ.

Although (4) can be used to measure the quality of our approximation, we would like to emphasize a different "quality-control" method that works well in practice.

[3] Due to unavoidable numerical errors we cannot expect to recover $\Phi_\theta(l)$ precisely.

[4] A crude approximation of θ_{s_0} would typically suffice for our purposes.

As observed in [9] $\mathrm{Im}\,\widetilde{p_\theta}$ is not 0 only because of numerical errors. Thus, $\varepsilon_{Im} = \max_j \left|\mathrm{Im}\,\widetilde{p_\theta}(j)\right|$ is typically an indicator of the level of noise in $\widetilde{p_\theta}$. For example, with $N = 400$, $Q = 8192$, and $\pi_k = k/15$ for $k = 1, \dots, 5$ we applied our algorithm 1.0 with $\theta = 1$ to find that setting a noise threshold of $\mathrm{Re}\,\widetilde{p_\theta}(j) > 10^3\varepsilon_{Im}$ correctly recovers all of the non vanishing entries of p_θ at 9-digit accuracy[5].

4 Our Algorithm 2.0

Algorithm 1.0 fixed the problem of numerical errors that plagued Baglivo et al.'s Algorithm but its runtime complexity of $O(QKN^2 + Q\log Q)$ is essentially the same as that of Hertz and Stormo. An advantage of Baglivo et al.'s Algorithm, however, is that it has a stingier memory requirement that scales as $O(Q + N)$ as opposed to $O(QN)$ for Hertz and Stormo. An important observation that helps us to improve on the runtime of our algorithm is the fact that (3) can be expressed as a convolution between the vectors $p_{k\theta l}(x) = p_{k,\theta}(x)e^{il\omega_0 s_k(x)}$ and $\psi_{k-1,\theta}(\cdot, l)$. A naively implemented convolution requires $O(N^2)$ steps and hence that factor in the overall complexity. Alternatively, an FFT-based convolution, justified by the equation $\left(D(u * v)\right)(j) = (Du)(j)(Dv)(j)$ [10], would only require $O(N\log N)$ steps cutting down the overall complexity to $O(QKN\log N + Q\log Q)$ [6].

Simply implementing (3) using FFT, however, reintroduces the severe numerical errors we worked hard to get rid of. The following example illustrates what is happening: for $\theta = 1$ one can easily verify that $p_{k,\theta}(x) \approx e^{-\lambda_k + x}/\sqrt{2\pi x}$. Computing $Dp_{k,\theta}(x)$ therefore faces essentially the same problem (only mirrored) as the one demonstrated in our example of FFT applied to e^{-x}. The solution is therefore to apply a negative exponential shift to $p_{k\theta l}$ and $\psi_{k-1,\theta}(\cdot, l)$ (i.e. multiply by $e^{-\theta_2(k)x}$).

The problem of choosing $\theta_2(k)$ is more involved than that of choosing θ. To begin with we have to choose a shift for each $k = 1, \dots, K$. In addition, in each case we have to worry about simultaneously shifting three vectors: $p_{k\theta l}$, $\psi_{k-1,\theta}(\cdot, l)$ and their convolution $\psi_{k,\theta}(\cdot, l)$. We propose the following solution:

$$\theta_2(k) = \mathrm{argmin}_{\theta'} \left[\theta' \sum_{i=1}^{k} \lambda_i + \log M_k(-\theta')\right], \tag{5}$$

where M_k is the MGF of $q_{k,\theta}(x) = \left(\psi_{k-1,\theta}(\cdot, 0) * p_{k,\theta}\right)(x)$ for $x = 0, \dots, 2N$ [7]. The intuition behind this choice of $\theta_2(k)$ is that it guarantees that the mean of $q_{k,\theta}$ is at $\sum_{i=1}^{k} \lambda_i$ which, loosely speaking, says that the distribution of $\sum_{i=1}^{k} X_i$ has maximal resolving power (relative to numerical noise) about its mean $\sum_{i=1}^{k} \lambda_i$.

We currently do not have theoretical bounds on the error that arises due to the use of $\theta_2(k)$. We instead tested our algorithm on a wide range of parameters and compared

[5] Except for $p_\theta(Q - 1)$ which has only 4-digits accuracy.

[6] While Hertz and Stormo make a passing remark that they can also use FFT-based convolution to cut the complexity to $O(QKN\log N)$ it seems unsubstantiated to us given that (17) in [5] is not strictly a convolution.

[7] Note that $q_{k,\theta}(x) = \psi_{k,\theta}(x, 0)$ for $x = 0, \dots, N$.

Table 1. Range of test parameters.

Parameter	Values
K	4, 10, 20
N	50, 100, 200, 400
π	$Uniform$, $Sloped$, $Blocked$
s	$\frac{i}{21} * I_{max}\ i \in [1..20]$

Uniform refers to the distribution where $\pi_j = 1/K$, Sloped refers to the case where $\pi_j = j/(K*(K+1)/2)$, and Blocked refers to the case where

$$\pi_j = \begin{cases} 3/(4*\lfloor K/4 \rfloor) & j \le \lfloor K/4 \rfloor \\ 1/(4*(K - \lfloor K/4 \rfloor)) & otherwise \end{cases}$$

our results with those obtained using Hertz and Stormo's algorithm (which is provably accurate) to get an estimate of the errors that arise in our algorithm. The range of parameters is given in Table 1. With Q set to 16384 and the other parameters exhaustively varying over the sets specified we found that our algorithm agreed with Hertz and Stormo's algorithm to at least 9 decimal places in all cases. This was found to be true even when we ran an experiment where we choose values of s much closer to I_{max} (using an interval halfing process on the range $[(\frac{20}{21} * I_{max})..I_{max}]$ to get 8 values of s) and let the other parameters vary as before. This gives us reasonable confidence in the belief that our methodology for choosing $\theta_2(k)$ works. We are currently working on a formal justification for this observation. In terms of complexity, the main loop now takes $O(QKN \log N)$. The other terms add $O(KN^2 + Q \log Q)$ to the runtime but this should be small compared to the runtime cost of the main loop[8] thus giving us a $O(QKN \log N)$ algorithm.

5 Comparison to Other Algorithms

For estimating a single p-value the complexity of version 2.0 of our algorithm is an improvement over Baglivo et al.'s Algorithm which has a time complexity of $O(QKN^2)$. More importantly, our algorithm offers much better control over the accumulation of numerical errors. For example, when $N = 50$, $K = 10$ and $\pi_i = 1/10$, Baglivo et al.'s Algorithm is able to recover the p-value for only 8 out of the 20 s values that we test on (where we only require correctness to 1 decimal place.) In contrast, our algorithm recovers the p-value accurately to at least 10 decimal places in all cases. In addition, our algorithm has a built-in quality control mechanism and should the p-value be too small for machine representation (not uncommon in bioinformatics applications) we can give the result in terms of log(p-value).

Bejerano's algorithm is more accurate than ours and is faster for $K \le 4$. However even for mildly large Ks it becomes impractical (in particular this is the case with $K = 20$) as it grows exponentially with K, presumably like $O(N^{K-2})$. Hertz and

[8] As observed in [11], in order to preserve the bound on the distance between p_Q and our real subject of interest, p_I, (the pmf of I), Q has to grow linearly with N.

Stormo are overall our closest competitors but their complexity is $O(QKN^2)$. In addition, if their algorithm is implemented explicitly as written [5], then it tends to suffer from intermediate underflows. For example, when applied to $N = 200$, $Q = 16384$, and $\pi \equiv 1/20$, all entries of p_Q less than 10^{-135} are estimated as 0. These intermediate errors can be eliminated if we switch to performing arithmetic on $\log p_Q$ instead of on p_Q, but that results in a non-trivial constant sitting in front of the $O(QKN^2)$. Alternatively, one can speed up the log arithmetic by using tables, as in Meme's (v3.0.3) implementation of Hertz and Stormo's algorithm, although that has the potential of introducing uncomfortably large numerical errors. For example, for $N = 50$, $\pi_i = i/10$ $i = 1, \ldots, 4$ and $Q = 10^4$, Meme's implementation seems to estimate the p-value of $s = 6$ as 0.0027 whereas the correct answer is 0.0095. An additional advantage of our method over Hertz and Stormo's algorithm (that it shares with Baglivo et al.'s Algorithm) is that the computation for each value of l can be carried out separately, incurring a space requirement that is $O(Q + N)$ whereas for Hertz and Stormo it is $O(QN)$.

We implemented our algorithm and Hertz and Stormo's algorithm in Matlab to compare the accuracy of the two algorithms. As a by-product, we also measured the running time of the two algorithms in Matlab and found that ours was on average between 10 and 100 times faster than Hertz and Stormo's algorithm on the range of parameters that we tested. In the case where $N = 400$ and $K = 20$, while Hertz and Stormo's algorithm took nearly a day and a half to compute the p-value, our algorithm took less than 7 minutes to do so. For more accurate runtime comparisons we have also written C programs that implement the two alogrithms. We have also worked on optimizing the C code for producing a fair comparison of the two algorithms (and there is still some scope for improvement, especially in the FFT implementations.) This is important because we found that while the code for Hertz and Stormo's algorithm shows little speedup when we turn on C compiler optimizations, the code for our algorithm runs twice as fast with even minor improvements to the code. The asymptotic behavior of the two algorithms is however clear even for small values of N. Figure 3 shows the behavior with increasing N for a fixed choice of the other parameter values (the graph looks the same for other choices of the parameter values too.)

Finally, while our algorithm was designed for finding a single p-value it turns out that in practice it can be easily adapted to reliably estimate p_Q in its entirety. The latter task is also performed by Hertz and Stormo's algorithm. In some cases our algorithm already does that. For example, setting $s = 500$ for $N = 400$, $Q = 8192$ and $\pi = i/15$ $i = 1, \ldots, 5$ we get a reliable estimate for the entire p_Q (relative error $< 10^{-6}$). More generally, in the cases that we have tried we can reliably recover the entire range of values of p_Q using as little as 2-3 different ss, or equivalently, θs (recall that each estimate has a quality control factor which allows us to choose the estimate which has better error gaurantees). Using version 2.0 of our algorithm this approach can still be significantly cheaper than running Hertz and Stormo's algorithm. We should also point out that Baglivo et al.'s comment regarding their algorithm being easily parallelizable is still valid for our algorithm since (3) can be computed separately for each $l = 0, 1, \ldots, Q - 1$.

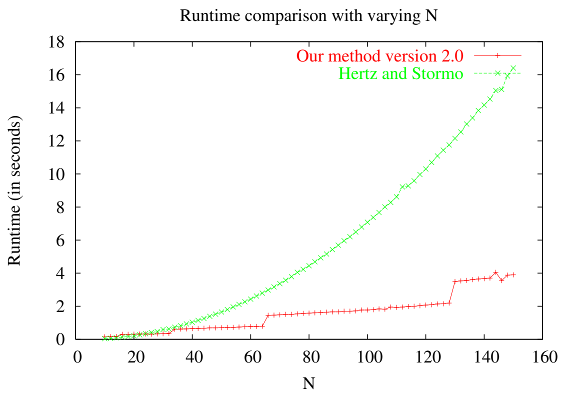

Fig. 3. Runtime comparison of Hertz and Stormo's algorithm and version 2.0 of our method.

The parameter values used in this comparison are $K = 20$, $\pi_j = j/(K * (K+1)/2)$, $s = 20$ and $Q = 1024$. Note that the discontinuities in the curve for our method are due to the fact that our implementation of FFT works with arrays whose sizes are powers of 2.

6 Future Work

There are many questions related to our work that deserve further study. For example, the theoretical and experimental question of how many different θs are needed to recover p_Q in its entirety is wide open. Another interesting area of study involves finding a theoretical grounding for our methodology for computing $\theta_2(k)$. On the practical side, the code for our algorithm is still being optimized, but we hope to make it available soon for the general public.

Finally, the most exciting prospect resulting from this work is that of combining it with our previous work [9] to obtain a fast and accurate system to compute the p-value of the entropy score of a multiple sequence alignment of biosequences. Note that the current work addresses the issue of computing the p-value of the entropy score of one column whereas our previous work focuses on computing the p-value of the sum of the entropy scores from all columns assuming that we have p_Q.

Acknowledgements

The first author would like to thank Gill Bejerano for the numerous discussions that we had on this subject and for generously sharing his code with us.

References

1. J. Baglivo, D. Olivier, and M. Pagano. Methods for exact goodness-of-fit tests. *Journal of the American Statistical Association*, 87(418):464–469, 1992.
2. T.L. Bailey and C. Elkan. 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*, pages 28–36, Menlo Park, California, 1994.

3. G. Bejerano. Efficient exact value computation and applications to biosequence analysis. In M. Vingron, S. Istrail, P.A. Pevzner, and M.S. Waterman, editors, *Proceedings of the Seventh Annual International Conference on Computational Molecular Biology (RECOMB-03)*, pages 38–47, Berlin, Germany, 2003. ACM Press.
4. N. Cressie and T.R.C. Read. Person's χ^2 and the loglikelihood ratio statistic g^2: A comparative review. *International Statistical Review*, 57(1):19–43, 1989.
5. G.Z. Hertz and G.D. Stormo. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, 15:563–577, 1999.
6. K.A. Hirji. A comparison of algorithms for exact goodness-of-fit tests for multinomial data. *Communications in Statistics-Simulation and Computations*, 26(3):1197–1227, 1997.
7. W. Hoeffding. Asymptotically optimal tests for multinomial distributions. *Annals of Mathematical Statistics*, 36:369–408, 1965.
8. W.C.M. Kallenberg. On moderate and large deviations in multinomial distributions. *Annals of Statistics*, 13(4):1554–1580, 1985.
9. U. Keich. Efficiently computing the p-value of the entropy score. *Journal of Computational Biology*, in press.
10. W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical recipes in C. The art of scientific computing.* Cambridge University Press, second edition, 1992.
11. S. Rahmann. Dynamic programming algorithms for two statistical problems in computational biology. In Gary Benson and Roderic D. M. Page, editors, *Proceedings of the Third International Workshop on Algorithms in Bioinformatics (WABI-03)*, volume 2812 of *Lecture Notes in Computer Science*, pages 151–164, Budapest, Hungary, 2003. Springer.
12. J.A. Rice. *Mathematical Statistics and Data Analysis*. Duxbury Press, second edition, 1995.
13. G.D. Stormo. DNA binding sites: representation and discovery. *Bioinformatics*, 16(1):16–23, 2000.

Appendix: Outline of the Proof of Claim 1

Proof. The following lemma can be readily derived from the results in [9] (see Lemmas 1-3, (20) & (21)). For $\alpha \in \mathbb{C}$ we denote by $\widetilde{\alpha}$ its machine estimator and define $e_\alpha = \widetilde{\alpha} - \alpha$. For $\alpha, \beta \in \mathbb{C}$, we define

$$e_{\alpha+\beta} = \widetilde{\widetilde{\alpha} + \widetilde{\beta}} - (\alpha + \beta),$$

and similarly for $e_{\alpha\beta}$.

Lemma 1. *If* $|e_\alpha| < c_\alpha \varepsilon_*$ *and* $|e_\beta| < c_\beta \varepsilon_*$*, then*

$$|e_{\alpha+\beta}| \le (\max\{c_\alpha, c_\beta\} + 1)\varepsilon_*(|\alpha| + |\beta|)$$
$$|e_{\alpha\beta}| \le (c_\alpha + c_\beta + 5)\varepsilon_*(|\alpha\beta|).$$

Using this lemma one can use (2) to prove by induction on k that

$$|\psi_k(n,l) - \widetilde{\psi_k}(n,l)| \le cNk\varepsilon_*\psi_k(n,0).$$

Note that when computing $\psi_k(n,0)$ we only deal with positive numbers. It follows that

$$|\widetilde{\Phi}_\theta(l) - \Phi_\theta(l)| \le CNK\varepsilon_*|\Phi_\theta(0)| = CNK\varepsilon_*,$$

since $\Phi_\theta(0) = 1$. Using this result, Lemma 4 from [9], the fact that for $x \in \mathbb{C}^Q$, $\|D^{-1}x\|_\infty \leq \frac{1}{Q}\|x\|_1$ and the triangle inequality we get

$$\begin{aligned}
\|D^{-1}\Phi - \widetilde{D^{-1}}\widetilde{\Phi}\|_\infty &\leq \|D^{-1}(\Phi - \widetilde{\Phi})\|_\infty + \|(D^{-1} - \widetilde{D^{-1}})\widetilde{\Phi}\|_\infty \\
&\leq \frac{1}{Q}\|\Phi - \widetilde{\Phi}\|_1 + \frac{(C\log Q)\varepsilon_*}{Q}\|\widetilde{\Phi}\|_1 \\
&\leq CNK\varepsilon_* + (C\log Q)\varepsilon_* .
\end{aligned}$$

Claim 1 now follows by applying the inverse exponential shift (i.e. by multiplying $e^{-\theta\delta j+\log M(\theta)}$) to this error bound.