Shuffling Biological Sequences

D. Kandel * Y. Matias † R. Unger ‡ P. Winkler †

* Gordon McKay Lab
Harvard University
Cambridge, MA 02138
danny@olympos.harvard.edu

† AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974
{matias,pw}@research.att.com

‡ Department of Life Sciences
Bar-Ilan University
Ramat Gan 52900, ISRAEL
ron@biocom1.ls.biu.ac.il

May 7, 1996

Abstract

This paper considers the following *sequence shuffling* problem: Given a biological sequence (either DNA or protein) s, generate a random instance among all the permutations of s that exhibit the *same* frequencies of k-lets (e.g, dinucleotides, doublets of amino acids, triplets etc.). Since certain biases in the usage of k-lets are fundamental to biological sequences, effective generation of such sequences is essential for the evaluation of the results of many sequence analysis tools. This paper introduces two sequence shuffling algorithms: A simple swapping-based algorithm is shown to generate a near-random instance and appears to work well, although its efficiency is unproven; a generation algorithm based on Euler tours is proven to produce a precisely uniform instance, and hence solve the sequence shuffling problem, in time not much more than linear in the sequence length.

1 Introduction

Computational analysis of biological sequences (or "strings") has become an invaluable tool in modern molecular biology. Examples include detecting relationships among genes, proteins or species; constructing evolutionary trees; aligning sequences or sets of sequences; recognizing coding regions of DNA; and prediction of protein secondary structure. In all these applications the question of the statistical significance of the results is one of the most important and difficult to address. For example, if two DNA sequences are found to share a common subsequence of a certain length, does it imply that the two sequences are functionally or evolutionarily related? Clearly the answer depends on how "surprising" this finding is. Since the alphabet of proteins consists of 20 amino acids, and the alphabet of DNA is just the 4 nucleotides A, C, G, and T, certain repeats can (or must) occur by chance. The significance of any finding must therefore be judged relative to a background level expected by chance alone (Fitch [7]).

Mathematical results concerning significance level for sequence analysis algorithms are very difficult to obtain and are known only in some special situations (Karlin and Brendel [13], Karlin and Altschul [12]). Therefore simulations are often used to provide a statistical background. The basic idea is to compare the results of a run on "real" data to many runs on "random" data.

The difficulty addressed here is how to construct an appropriate random data. For example, note that two DNA sequences that are rich in C and G nucleotides are more likely to have a common subsequence of a given length than two sequences in which the 4 nucleotides are equally frequent. While the mere fact that the two sequences are G,C rich might be of some interest, we are usually interested in asking the next question: Given that the two sequences are C,G rich, what is the significance of finding a certain common subsequence?

To answer such questions it is natural to create random sequences that have the same nucleotide decomposition as the original sequences. If the object is merely to produce a uniformly random sequence with the same number of A's, C's, G's and T's as a given sequence, there are two simple, efficient procedures available. One can either tally the frequencies in the given sequence and *generate* a uniform permutation of the nucleotide multiset; or, one can *shuffle* the given sequence until it is adequately mixed.

Both of these methods are familiar to players of the game of bridge. In a home game the cards are shuffled by hand, but in a tournament the deals are randomly generated (by computer). Both methods have their pitfalls; although shuffling is capable of producing very nearly perfectly random deals (see, e.g., Bayer and Diaconis [3]), lazy shufflers produce non-random effects which, owing to the way the cards are collected in bridge, result in relatively tame deals. On the other hand a re-used seed for a random number generator resulted recently in having to invalidate results from a major tournament when players recognized the hands from a previous tournament [15].

In many cases, biological sequences are biased not only at the single letter level but also at higher levels (see, e.g., Karlin *et al.* [14]). Doublets, triplets, etc. (generically called k-lets) are counted in an overlapping manner, so that for example the sequence ACGAC contains two AC dinucleotides, one CG, and one GA; and one copy each of the ACG, CGA, and GAC trinucleotides. Certain biological sequences tend to have an excess of some k-lets, while others are underrepresented; for example, the dinucleotide TA is broadly underrepresented. In vertebrates, CG is underrepresented and TG and CA are overrepresented. The trinucleotides CCA and TGG are overrepresented in eukaryotic sequences. The tetranucleotide CTAG is underrepresented in bacterial and eukaryotic sequences.

The triplet frequency is of special interest since codons (triples of nucleotides in certain regions of DNA that code for specific amino acids) are subject to many evolutionary and functional pressures. Thus, as mentioned above, for many sequence analysis tools there is a need to produce random sequences that maintain the specific biases of the original sequence.

In any case the problem is the same: for fixed k and given sequence $s = s_1 s_2 \dots s_n$, let $X_k(s)$ be the set of all sequences which contain the same number of each type of k-let that s does. We wish to obtain an element of $X_k(s)$ chosen randomly from the uniform distribution. We will assume for the remainder of this paper (unless specified otherwise) that the sequence alphabet is of size 4, so that there are 4^k k-let types. However, everything we do generalizes in the obvious way.

It is not obvious, however, how to do either generation or shuffling when $k > 1$. In what follows we describe methods for both. Our shuffling method (the "swap") is simple and appears to work well, although its efficiency is unproven. Our generation method is an improvement of the method of Altschul and Erickson [2], and it produces a precisely uniform member of $X_k(s)$ in time not much more than linear in the sequence length n.

2 Previous Work

The first treatment of this problem appears to be due to Fitch [7]. Fitch noticed the connection between the doublet problem and Euler tours, but suggested an algorithm which does not generally achieve the uniform distribution among valid permutations. Altschul and Erickson [2] presented an algorithm based on Euler tours which does generate uniform valid permutations but relies on trial-and-error generation of random trees; we will show later how to fix this potential bottleneck in a fast and elegant manner.

A brief outline of the "swap" algorithm for shuffling was given in (Unger *et al.* [20]), without proof.

3 The Swapping Algorithm

The swapping algorithm to preserve k-lets is an extension of the simple swapping algorithm for single character. Because of the dependence of characters in higher k-lets, a simple swap of two characters at a time will generally change the k-let count; for example, If $\mathrm{TTACA\underline{CT}GATTCA\underline{AG}TTAAT}$ is swapped into $\mathrm{TTACA\underline{AG}GATTCA\underline{CT}TTAAT}$, the doublet TG is destroyed while the doublet GG is created.

Instead, we endeavor to locate two substrings (contiguous subsequences of arbitrary lengths) which are disjoint and flanked by the same $(k-1)$-lets; that is, the $k-1$ letters at the left end of one substring must be the same as those on the left end of the other substring, and similarly for the right ends. These substrings are then swapped.

The substrings need not be the same length, although if they are then that length must be at least $2k-1$ for the swap to accomplish anything. In any case it is easy to see that the k-let frequencies are preserved by such a swap.

For example, when $k = 2$

$$\mathrm{TTA\underline{\underline{C}ACTG\underline{A}}TT\underline{\underline{C}AAGTT\underline{A}}AT}$$

can be changed to

$$\mathrm{TTA\underline{\underline{C}AAGTT\underline{A}}TT\underline{\underline{C}ACTG\underline{A}}AT}$$

by a swap; the doublet count is not affected.

The swap algorithm entails running a Markov chain on the state space $X_k(s)$ beginning at state s. To step from a sequence $t = t_1 \dots t_n$, we randomly and uniformly choose 4 positions

a, b, c, d along the sequence with $1 \leq a < b < c < d \leq n - k + 2$. We then check whether it happens that

$$t_a t_{a+1} \dots t_{a+(k-2)} = t_c t_{c+1} \dots t_{c+(k-2)}$$

and

$$t_b t_{b+1} \dots t_{b+(k-2)} = t_d t_{d+1} \dots t_{d+(k-2)}$$

If these conditions are met then the substrings $t_a \dots t_{b+k-2}$ and $t_c \dots t_{d+k-2}$ are swapped; otherwise we remain in state t. (In practice a more sophisticated data structure can be maintained to make the selection of a, b, c and d more efficient. It is important though to be careful not to introduce any bias towards specific positions.)

Notice that the chosen substrings may overlap somewhat, in which case they will perforce overlap an equal amount after swapping. The start-and-end conditions assure that the overlap portion $t_c \dots t_{b+k-2}$ remains intact.

The swap defined above is not quite sufficient to move around $X_k(s)$ in the special case where s happens to begin and end with the same k-let. For example, ACGTAC and GTACGT have the same triplet counts but neither permits a swap. To overcome this obstacle we say that a sequence $s = s_1 s_2 \dots s_n$ is *k-cyclic* (or just "cyclic" when k is understood) if $s_1 \dots s_{k-1} = s_{n-k+2} \dots s_n$, and if the given sequence s is cyclic our algorithm is preceded by a *random rotation* as follows: a number m is chosen randomly and uniformly from $\{k, k+1, \dots, n\}$ and s is replaced by the sequence

$$s' = s_m s_{m+1} \dots s_{n-1} s_n s_k s_{k+1} \dots s_{m-1} s_m \dots s_{m+k-2}$$

where the subscripts are reduced modulo n if necessary. Note that s' is also cyclic and has the same k-let count as s; it may be equal to s, even when m is not equal to 1.

An intuitive way to think of the random rotation is as follows: cut off the "head" $(k-1)$-let of s and join the ends of the remaining sequence to make a necklace; then snip the necklace at a random point and add a copy of the $(k-1)$-let at its tail to the sequence's head, to get the new sequence s'. This procedure will always change the $(k-1)$-let count.

The Markov chain then proceeds from s' as above. Observe that swaps preserve both the initial and final $(k-1)$-let, hence in particular the sequences produced by the algorithm will be either all cyclic or all acyclic depending on s.

It is useful to note that while the swap procedure for a given k preserves the j-let count for all $j \leq k$, the rotation changes the $(k-1)$-let count unless the head $(k-1)$-let remains the same (in which case we could have gotten to s' by swaps). Let $Y_k(s)$ be the set of all sequences with the same k-let counts as s and the same starting $(k-1)$-let. Then we have the following facts, which can be easily verified based on the definitions of $X_k(s)$ and $Y_k(s)$:

- $Y_k(s) \subset X_k(s)$ with equality when s is acyclic (and in the degenerate case when all characters in s are the same).
- $X_k(s) = \cup Y_k(s')$ in the cyclic case, where the union is taken over all rotations s' of s, or at least one rotation for each possible starting $(k-1)$-let.
- Regardless of whether s is cyclic or not, $Y_k(s)$ is exactly the set of all sequences whose j-let counts match s's for all $j \leq k$; also the set of all sequences whose k-let counts and $(k-1)$-let counts match s's.

Because of this last fact, in a given experiment it might well be deemed preferable to obtain random sequences in $Y_k(s)$ rather than $X_k(s)$. This makes no difference in the acyclic case, but when s happens to be cyclic it simplifies matters by obviating the necessity for a random rotation.

4 Proving the Correctness of the Algorithm

While the algorithm is simple, to prove its correctness and efficiency we must show that:

1. The algorithm produces all the valid k-lets permutations (i.e., permutations that preserve the same k-lets count) of the input sequence;
2. All possible outputs of the algorithm are obtained with (approximately) the same probability;
3. The required number of iterations of the algorithm is reasonable, e.g., bounded by a polynomial in the length n of the input sequence.

In the following sections we will prove the first two assertions and present empirical results on behalf of the third.

4.1 Reaching all Valid Permutations

The fact that an individual swap step is k-let preserving does not of course imply that every k-let preserving permutation is accessible using iterative application of random swaps. To prove this crucial aspect of the algorithm, it suffices to show that there is a path of specific swaps that can transform any given valid permutation to any other. We do this by defining a metric ρ for $X_k(s)$ with values in the set $\{0, 1, \ldots, n-2k+2\}$, then showing that for any two distinct valid strings u and v there is a sequence u' reachable by a swap from u such that $\rho(u', v) < \rho(u, v)$.

Our Theorem 1, below, is slightly stronger than Pevzner's result [18] used in the analysis of the double digest problem. Pevzner answered a question of Ukkonen [19] by showing that (in our terminology) any valid string can be reached by a sequence of swaps and rotations.

It is convenient to denote by $G_k(s)$ the graph whose vertices consist of the valid sequences which begin with the same $(k-1)$-let as s, i.e. all sequences in $Y_k(s)$, with two sequences adjacent in $G_k(s)$ just when they can be obtained from each other by a single swap.

Theorem 1 *For any sequence s of length n the graph $G_k(s)$ is connected and has diameter at most $n-2k+2$.*

Proof. The metric ρ is defined as follows: if $j(u,v)$ is the least number for which $u_j \neq v_j$, then $\rho(u,v) = n-k+2-j(u,v)$. Of course, in case that $u=v$ we put $\rho(u,v)=0$.

Note that since all vertices in $G_k(s)$ begin with the same $(k-1)$-let (namely $s_1 \ldots s_{k-1}$) and end with the same $(k-1)$-let $(s_{n-k+2} \ldots s_n)$, the range of $j(u,v)$ when $u \neq v$ is contained in $\{k, k+1, \ldots, n-k+1\}$ and therefore the full range of ρ is contained in $\{0, 1, \ldots, n-2k+2\}$ as promised. Thus, it suffices to prove that for any two distinct vertices u, v there is a swap changing u to u' such that $j(u',v) > j(u,v)$.

In fact, we may assume $j(u,v) = k$ because otherwise the first $j(u,v)-k$ coordinates of u and v can be ignored without loss, in effect replacing n by $n-(j(u,v)-k)$. Letting α be the common initial $(k-1)$-let for the vertices of $G_k(s)$, we have that u begins with αu_k and v with αv_k where $u_k \neq v_k$.

Let h be the largest index for which $u_h u_{h+1} \ldots u_{h+k-1} = \alpha v_k$; this number exists, or course, since u and v have the same k-let count. Let A be the set of all $(k-1)$-lets which occur in u before h and B after, that is,

$$A := \{u_i \ldots u_{i+k-2} : \ i < h\} \text{ and } B := \{u_i \ldots u_{i+k-2} : \ i > h\}.$$

We claim that A and B cannot be disjoint. To see this suppose otherwise and note that in the sequence v the second $(k-1)$-let, $v_2 \ldots v_k$, belongs to B; furthermore A includes at least one $(k-1)$-let which is not $v_1 \ldots v_{k-1}$, and thus somewhere later in v there is a k-let of the form $\beta x = y\gamma$ where β and γ are $(k-1)$-lets, and such that β is in B and γ is not. But there is no place in u for such a k-let, a contradiction.

Hence we may choose a common $(k-1)$-let $\delta \in A \cap B$, occurring say at positions i_1 and i_2 with $1 < i_1 < h < i_2$. Swapping in u in accordance with locations $a = 1$, $b = i_1$, $c = h$, and $d = i_2$ replaces u_k by v_k, increasing $j(u, v)$ as desired and proving the theorem. $\square$

It follows from the theorem that we can get from an acyclic s to any $t \in X_k(s)$ in at most $n - 2k + 2$ steps; if s is cyclic we first rotate to obtain an s' with $s'_1 \ldots s'_{k-1} = t_1 \ldots t_{k-1}$, then apply the theorem to the graph $G_k(s')$. We remark that the bound $n - 2k + 2$ for the diameter of $G_k(s)$ is not tight, but a degree argument shows that it is not off by more than a factor of function of k times $\log n$.

4.2 The Limiting Probability of a Permutations

When s is acyclic the swap algorithm is exactly a "simple random walk" (see, for example, [5]) on the graph $G_k(s)$, which is regular of degree $\binom{n-k+1}{4}$ since we have in effect put in loops wherever the conditions for the locations a, b, c and d are not met. The stationary distribution for a simple random walk on a connected, regular graph that is not bipartite is easily seen to be uniform. Note that $G_k(s)$ can be bipartite only if all the possible swaps of s are valid, which may be the case only if $k = 1$ or if the sequence is short; both cases are not very interesting in our context, but are nevertheless considered below. Thus a sufficiently long walk on $G_k(s)$ will end at as nearly a uniformly random sequence as desired.

If s is cyclic we have the additional task of showing a member v of $X_k(s)$ with different initial $(k-1)$-lets α from s has the same probability as s in the limit. Let j be such that a rotation by j transforms s to a sequence s' beginning with α. Then s' has the same probability as v, but also s' has the same probability as s because rotation by j is a one-to-one bijection on $X_k(s)$ and all rotations are equally likely.

We elaborate below on the proof that the limiting probability distribution of the swap algorithm is uniform, using the *ergodic theorem* (see, for example, [11]). First, define the matrix T of conditional probabilities, such that for any $t, t' \in X_k(s)$, $T(t, t')$ is the conditional probability that the sequence t is obtained after m swap attempts, assuming that the sequence t' was obtained after $m-1$ swap attempts. The probability $P^{(m)}(t)$ of obtaining t after m swap attempts is therefore $P^{(m)}(t) = \sum_{t'} T(t, t') P^{(m-1)}(t')$, or in more compact notation $P^{(m)} = T\ P^{(m-1)}$. Iterating this equation m times, we can get the vector $P^{(m)}$ from the initial probability distribution of sequences: $P^{(m)} = T^m\ P^{(0)}$.

T is a stochastic matrix, i.e., it is nonnegative and $\sum_t T(t,t') = 1$ for any t'. Since in our case T is symmetric, the uniform probability distribution $P_u(t) = 1/N_k(s)$ (where $N_k(s)$ is the number of sequences in $X_k(s)$) is an eigenvector of T with eigenvalue 1. Thus P_u is a stationary distribution of the Markov process. It can be shown [11], using theorems of Perron [16, 17] and Frobenius [8, 9, 10], that 1 is a simple eigenvalue of T (i.e. there is a single eigenvector associated with this eigenvalue). This eigenvalue is also the dominant one (i.e. there are no eigenvalues of modulus greater than 1). Moreover, T has exactly one eigenvalue of modulus 1 if and only if there is a power of T which is positive. Hence if this condition holds, P_u is the only eigenvector of T with eigenvalue 1.

The ergodic theorem states [11] that, under the same condition, our Markov process approaches this uniquely defined distribution in the limit of an infinite number of swaps. Therefore, the limiting distribution of the swapping algorithm is uniform if and only if there is an integer l such that T^l is positive. Positivity of all the elements of T^l means that $P^{(l)}(t) > 0$ for any sequence t no matter what the initial distribution of sequences $P^{(0)}$ is. We have already shown (see section 4.1) that any sequence in $X_k(s)$ can be brought into any other sequence with a finite number of swaps. We denote the minimal number of swaps that brings sequence t into sequence t' by $l(t,t')$. Let us define $l \equiv \max_{t,t'} l(t,t')$. To complete our proof it is sufficient to show that the sequence t' remains unchanged with a positive probability from swap attempt $l(t,t')$ till swap attempt l.

This is obviously correct when $k > 1$ and the sequence is long enough, since in such cases there is always a positive probability of attempting to make illegal swaps (and hence rejecting the attempt). A problem arises, however, when $k = 1$. All swaps are legal in this case, and therefore none are rejected. As a result there is no integer l for which T^l is positive and the swapping algorithm is not guaranteed to approach the uniform distribution. Consider, for example, the sequences GT and TG. If we start with GT we obtain the sequence TG with probability 1 after any odd number of swap attempts. After an even number of swap attempts we get GT with probability 1. The limiting distribution does not exist here; instead we approach a limit two-cycle. In all cases where such a problem arises it is easily solved by modifying the algorithm so that a swap attempt is rejected with a positive probability even if it is legal.

In conclusion, the limiting distribution of sequences obtained using the swapping algorithm is uniform, except when $k = 1$ or when the sequence is very short. In these cases a minor modification of the algorithm restores ergodicity.

4.3 The Convergence Rate

We have shown that the swap algorithm works, in principle, to any desired degree of accuracy; but to show that it is efficient we would need to prove that the number of swaps required to reach near-uniformity is reasonable. Our empirical tests (illustrated below) and other indications, such as the small diameter of $G_k(s)$, lead us to guess that we have "polynomial mixing time," as follows:

Conjecture 1 *For any fixed k and alphabet size, there is a polynomial $p = p(n)$ such that for any sequence s of length n and any number m, a walk of $mp(n)$ steps on $X_k(s)$ will produce a final distribution of variation distance less than 2^{-m} from uniform.*

4.4 Some Empirical Results

We now demonstrate a certain mixing property in several executions of the swap algorithm. The experimental results are consistent with conjecture 1, and provide positive indication as to the convergence property of the swap algorithm. More experiments are necessary, however, in order to constitute a rigorous empirical study on behalf of conjecture 1; these are left for future research.

For actual biological sequences the number of valid k-lets is huge (see the end of section 5 for the actual count). Hence, it is not realistic to demonstrate the uniformity of the resulting distribution just by sampling. Instead we measured a time dependent correlation function of the generated sequences. To define this correlation function we first quantify the difference between two sequences through the following metric:

$$O_k(s_i, s_j) = \frac{1}{\sigma^{k+1}} \sum_{l=1}^{\sigma^{k+1}} |f_l(s_i) - f_l(s_j)|,$$

where σ is the size of the alpha-bet, and $f_l(s_i)$ is the frequency of the $(k+1)$-let subsequent l in sequence s_i. This metric is a measure of the difference between the two sequences since it compares their $k+1$-let counts (recall that the number of k-lets is preserved). For example, in DNA sequences with doublet preserving ($k = 2$), we average the difference in the frequencies of all the 64 possible triplets between the two sequences.

We compute $\langle O(t) \rangle$ by averaging $O_k(s_t, s_0)$ over different runs (i.e., $\langle O(t) \rangle$ represents the ensemble average of $O_k(s_t, s_0)$), using the same starting sequence s_0. We used to following procedure: 10 DNA sequences were randomly created ranging in size from 300 to 4800

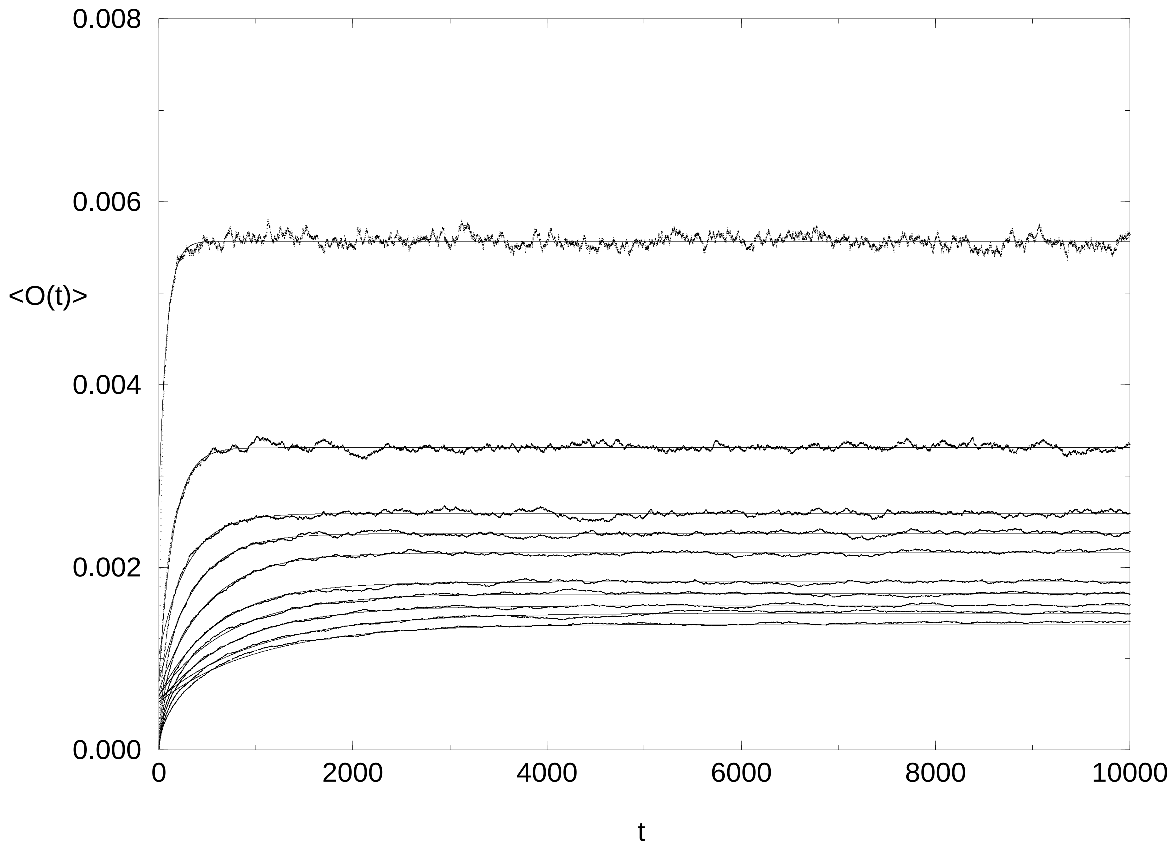

Figure 1: $\langle O(t) \rangle$ is plotted versus t for 10 random sequences of different lengths from 300 to 4800. Each sequence has undergone 10000 swap steps preserving the doublet distribution. For each sequence, 100 simulations were run, and the correlation function $\langle O(t) \rangle$ reflects the average difference in the triplet distribution between the sequence after t swap steps and the original sequence (see text). After time shorter than the length of each sequence (the shortest sequence (300bp) appears on the top and the longest (4800bp) on the bottom) a plateau in $\langle O(t) \rangle$ is reached. The results were fitted, using least square method, into functions of the form $A \cdot e^{-t/\tau} + C$.

(steps of 500). For each sequence we ran the swapping algorithm with doublet preserving ($k = 2$) for 100 independent runs, each consisting of 10000 swap steps, and took $\langle O(t) \rangle$ as the average over the computed $O_k(s_t, s_0)$. Figure 1 shows $\langle O(t) \rangle$ vs. t for these sequences. The curves show exponential behavior, and the plateau in each curve starts after a number of steps which is smaller than the length of the sequence. These curves were fitted, using least square method, into functions of the form $A \cdot e^{-t/\tau} + C$ with a good fit (see Figure 1), where A and C are dependent of the length of the sequence, but are constants with respect to t. Thus, τ reflects the rate in which the curves approach their limiting values C.

We would like to show that the limiting values of these curves are indeed what is expected from a comparison between random valid k-let permutations of a given sequence. That is, to show that $\langle O(t) \rangle$ approaches the value $O_k(s, s')$ averaged over all s' in $X_k(s)$. Again, as the size of $X_k(s)$ is too big, we used a sample to estimate the expected $\langle O(t) \rangle$. The sampling was made from very long swapping simulations (1 million steps), randomly selecting 300 sequences, and calculating $O_k(s_i, s_j)$ for each pair of these sequences. In the case of the sequence of length 800bp $\langle O(t) \rangle$ was calculated to be 0.00331 with standard deviation of 0.00047. This value exactly matches the limiting value $C = 0.00331$ that was fitted to the corresponding curve (the second curve from the top in Figure 1), implying that the discrepancy is smaller than the numerical errors, which are of low order magnitude. So, the correlation of the $k+1$-let count of the produced sequences with the original sequence reached a fixed level which is similar to the one expected between any two random valid k-let permutations of the sequence.

Figure 2 shows the near-linear correlation between τ and the length of the sequence N, with a best fit of $\tau \approx 0.2 \cdot N$. Similar results were obtained for different sequences with various sizes and using different preserving levels (e.g. $k = 3$ and $k = 4$).

We conclude that the correlation of the initial random sequence s_0 with the sequence s_t decays exponentially in t, and after time τ that scales linearly with N, only a small fixed degree of correlation is left. Note that the empirical results do not preclude the possible existence of "bad" initial sequences s_0 for which the convergence rate is slower, even for the specific correlation properties considered here. The issue of identifying the convergence rate in the worst case remains an open question.

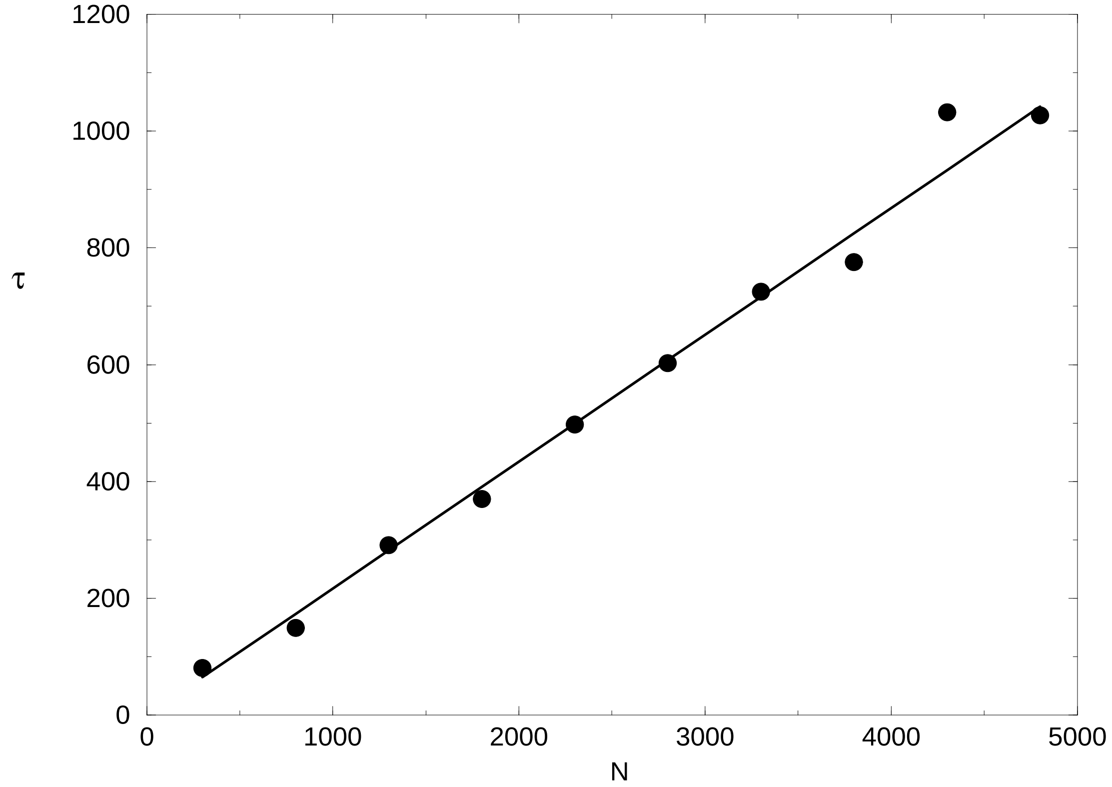

Figure 2: τ is plotted versus N. The rate of increase of the correlation function $\langle O(t) \rangle$ for each sequence (Figure 1) is reflected by τ. (Recall that τ is taken from the functions of the form $A \cdot e^{-t/\tau} + C$ which were fitted to the data.) τ scales linearly with the length of each sequence N which is consistent with conjecture 1.

5 Exact Sampling

The swap algorithm, although it appears to work well, produces valid sequences which are only approximately uniform and does not (as yet) have a proof of efficiency. We now present a provably efficient variation of the Altschul-Erickson algorithm [2] which yields an exactly uniformly random valid sequence. This technique, which we dub the "Euler algorithm," operates on a directed graph $D_k(s)$ whose size is *constant* with respect to n instead of exponential.

The vertices of $D_k(s)$ are the distinct $(k-1)$-lets which appear in s, thus $|V(D_k(s))| \leq 4^{k-1}$. Clearly, also $|V(D_k(s))| \leq n-k+1$. With each of the $n-k+1$ k-lets $s_i \ldots s_{i+k-1}$ we associate an arc e_i from the vertex $s_i \ldots s_{i+k-2}$ to the vertex $s_{i+1} \ldots s_{i+k-1}$. (A k-let of the form XXX...X causes a loop.) Thus, $|E(D_k(s))| \leq n-k+1$.

Letting $\alpha = s_1 \ldots s_{k-1}$ and $\beta = s_{n-k+2} \ldots s_n$, we see that the arc-sequence $e_1, \ldots, e_{n-k+1}$ constitutes an Euler trail from α to β, that is, a directed path in $D_k(s)$ which passes over each arc exactly once. Conversely any directed trail in $D_k(s)$ produces a sequence and if the trail is Eulerian, the sequence is a valid k-permutation of s.

If there are two or more arcs in $D_k(s)$ from, say, vertex γ to vertex δ, the choice of which arc the Euler trail takes in first passing from γ to δ makes no difference in the sequence generated. Hence the correspondence between Euler trails and valid sequences is not one-to-one; however,

Lemma 1 *Let $f_1, \ldots, f_m$ be the frequencies of those k-lets which are present in s. Then every valid sequence in $X_k(s)$ corresponds to exactly $\prod_{i=1}^{m} f_i!$ Euler trails in $D_k(s)$.*

Proof. This is merely a matter of observing that the frequencies correspond to multiplicities of arcs in $D_k(s)$, and f_i parallel arcs can be taken in any of their $f_i!$ possible orders. □

There is a slight subtlety in the statement of Lemma 1 which arises when s is cyclic. In the acyclic case, the outdegree of α exceeds its indegree by one, and vice-versa for β, so that every Euler trail must begin at α and end at β. But in the cyclic case α and β are the same vertex, with equal indegree and outdegree like every other vertex of $D_k(s)$. Here every Euler trail is closed, but we still regard an Euler trail as having a start and (identical) finish rather than as an endless circuit; otherwise we would not know how to begin the corresponding sequence.

On account of Lemma 1 we have reduced the problem of generating a uniform valid sequence to the problem of generating a uniform Euler trail in $D_k(s)$. To do this we make another reduction. Fix an Euler trail E, and for any vertex $\gamma \neq \beta$ let $e(\gamma)$ be the arc from γ taken in the last exit of E from γ.

The set $T = T(E)$ of such arcs form an inbound spanning tree, or *arborescence*, rooted at β. To see this it suffices to note that T has out-degree 1 at every vertex other than β and has no cycles; for, the trail E would have no way to last exit a cycle.

Conversely, let T be any arborescence rooted at β and for each $\gamma \neq \beta$ let us order all the arcs leaving γ other than the one belonging to T; we also order all the arcs, if any, which exit β. Now we begin at α and walk according to the following rule: at each vertex we exit by the first exit-arc not previously chosen, using the T-arc only when all the alternatives are eliminated. When the tour ends (perforce at β) we will have covered every arc, else we could not have got from the vertices whose T-arcs have been used to the rest. From all this we have:

Lemma 2 *Every arborescence rooted at β corresponds to exactly*

$$d^+(\beta)! \prod_{\gamma \neq \beta} (d^+(\gamma) - 1)!$$

Euler trails ending at β (and thus beginning at α), where $d^+(\gamma)$ is the outdegree of γ, i.e., the number of exiting arcs.

Since it is very easy to generate random permutations of exit arcs, we are reduced now to the problem of generating uniform random arborescences. This is done in [2] by choosing random exit arcs from each $\gamma \neq \beta$ and hoping the result is a tree, else repeating the procedure. Unfortunately there may be many places in $D_k(s)$ where a cycle is likely, so that the expected number of tries before achieving a tree could be as high as order $n^{c4^{k-1}}$.

We can dilate this bottleneck with the help of the Matrix-Tree Theorem, (see *e.g.* [6] Thm 2.9) but a very fast and elegant alternative is now available thanks to recent work on random walks. Aldous [1] and Broder [4] proved independently that a uniform random spanning tree for an undirected graph can be obtained by taking a simple random walk on the graph and marking the first edge used to reach each vertex. It turns out that their theorem can be extended to Eulerian digraphs.

A digraph is said to be *Eulerian* if it is connected and $d^+(\gamma) = d^-(\gamma)$ for every vertex γ. Our $D_k(s)$ is connected since it has an Euler trail, and is thus Eulerian if s is cyclic. Otherwise we add a "phony" arc from β to α to produce the Eulerian alteration $D'_k(s)$.

The method, for a general Eulerian digraph D, is as follows: We take a *backward* random walk on D, beginning at the root vertex (β). At each vertex γ we randomly and uniformly choose among all arcs leading to γ (loops can be ignored) and proceed next to the vertex (say, γ'') at the tail of the chosen arc. If we have reached γ'' for the first time, the arc just traversed (which points from γ'' to γ) is added to T. The procedure terminates when D is covered, that is, every vertex has been reached; clearly we then have an arborescence rooted at β.

Theorem 2 *The arborescence T constructed above is drawn precisely from the uniform distribution among all arborescences rooted at β.*

Proof. The proof follows the same lines as Aldous', using the Eulerian property instead of reversibility at the critical point.

Let $\{\mathbf{X}_j\}$ be a doubly infinite (thus stationary) backward random walk on D and let T_j be the arborescence rooted at $\mathbf{X}_j$ defined as above starting at time j.

Because indegree = outdegree for each vertex of D, the probability for each j that $\mathbf{X}_{j-1} = \gamma$ given $\mathbf{X}_j = \gamma'$, where γ is a successor of γ' in D, is $1/d^+(\gamma')$.

Given T_j and knowing that $X_{j-1} = \gamma$, the tree T_{j-1} is determined; it is obtained from T_j by deleting the arc leading from γ toward the root, and inserting an arc from γ' to the new root γ. Thus, the trees themselves, moving backwards in time, constitute a Markov chain.

Let Q be the transition matrix for this reverse-time, tree-valued Markov chain T_j. If $r(T)$ is the (out)degree of the root of T, then the probability $Q_{T,T'}$ of proceeding to T' from T is $1/r(T)$ for exactly $r(T)$ arborescences T', one rooted at each successor of the root of T.

On the other hand if $T = T_j$ is rooted at $\gamma' = \mathbf{X}_j$ then for each predecessor γ'' of γ' (in D) there is a unique tree $T' = T_{j+1}$ in which γ'' precedes γ', i.e. there are exactly $r(T)$ trees T' for which $Q_{T,T'} = 1/r(T')$, and $Q_{T,T'} = 0$ for all other trees T'.

Thus we have for each fixed T' that $\sum_T r(T)Q_{T,T'} = r(T')$.

Since it is easily checked that Q is irreducible, we deduce that Q has stationary distribution proportional to $r(T)$. But the arborescence T_j generated from time j has no dependence on the past. It follows that if we begin a backward random walk on D at β, first-entries provide a uniform random arborescence rooted at β. $\square$

Let us now review our Euler algorithm for generating a uniform random valid permutation of s:

1. Construct the digraph $D_k(s)$ from the k-let counts. (For small k and large n, this is best done by recording the counts as arc multiplicities in a $4^{k-1} \times 4^{k-1}$ matrix indexed by all $(k-1)$-lets.)

2. If s is cyclic, perform a random rotation to get a sequence $s' \in X_k(s)$ as in the swap algorithm, then put $\alpha = \beta = s'_1 \dots s'_{k-1}$. If s is acyclic, add an arc from $\beta = s_{n-k+2} \dots s_n$ to $\alpha = s_1 \dots s_{k-1}$ to make the digraph Eulerian.

3. Take a simple backward random walk from β until all other vertices have been hit. Whenever a vertex $\gamma \neq \beta$ is reached for the first time, put the arc (γ, γ') just traversed into T.

4. For each vertex γ of $D_k(s)$, randomly order all the arcs exiting γ except the one in T; in the case of β, randomly order all the exiting arcs (but not the one added from β to α in the acyclic case).

5. Read off the desired sequence by starting at α and, at each vertex γ, following the first arc from γ not yet used, using the T-arc last.

The number of vertices of $D_k(s)$ is of course constant with respect to n, when k is held fixed. Hence the computing time required for steps (1), (2), (4), and (5) above is basically linear in n (with, in theory, a penalty of a factor of $\log n$ for having to deal with numbers of size about n). So the only remaining problem is to get a bound on the "cover time" of $D_k(s)$, that is, the expected time to hit all other vertices from β.

Theorem 3 *Let D be an Eulerian digraph consisting of n arcs on q vertices. Then the expected cover time of D is less than $q^2 n$.*

Proof. We show first that if U is any proper subset of the vertices of D, then there is a vertex γ outside U from which the expected hitting time to U is at most $n-1$. To see this, note that the graph D/U obtained by contracting U to a single vertex and eliminating all loops at U is still Eulerian, and thus (as in the proof of Theorem 2 above) the stationary distribution for a random walk on D/U is proportional to the degrees of its vertices. Since the sum of the (out)degrees is at most n and the degree of U is at least 1, the stationary probability of U is at least $1/n$. Thus the expected time to revisit U starting from U is at most n, but the first step exits U; thus there must be a vertex $\gamma \neq U$ in D/U from which the expected hitting time to U is no more than $n-1$.

Next we argue that for any two vertices α and β, the expected hitting time from α to β is bounded by $(q-1)(n-1)$. For, we may set $U_1 := \{\beta\}$ and define $U_{i+1} := U_i \cup \{\gamma_i\}$, where γ_i is a vertex not in U_i from which expected access to U_i takes at most $n-1$ steps. Then U_q contains all the vertices, α in particular; thus expected hitting time from α to β can not be more than $(q-1)(n-1)$.

Finally we conclude that expected cover time cannot exceed $(q-1)^2(n-1)$, even if we insist that a subsequence of the walk contain all the vertices in a particular order. $\square$

We thus have that the whole Euler algorithm is essentially linear in n when k (and the alphabet size) are held constant. When $k = 3$ the digraph $D_k(n)$ has at most 16 vertices, so there is nothing to prevent the Euler algorithm from being run with extremely long sequences.

Obviously the Euler algorithm can be used to generate a uniform random Eulerian trail in any Eulerian digraph; interestingly no one seems to have found a way to do the same for *undirected* graphs, where the neat relationship between spanning trees and Euler trails breaks down. There is a way to generate uniform random Eulerian *orientations* of an undirected graph [MW], but this cannot be used to generate trails because different Eulerian orientations may have widely different numbers of trails.

Finally, we note that a variation of the Euler algorithm can be used to get an exact count of $|Y_k(s)|$ (or of the number of Euler trails in an Eulerian digraph). To do this we use the Matrix Tree Theorem to count the number t of arborescences rooted at β, then in the acyclic case we have

$$|X_k(s)| = td^+(\beta)! \prod_{\gamma \neq \beta} (d^+(\gamma) - 1)! / \prod_{i=1}^{m} f_i!$$

where the f_i's are the k-let frequencies as in Lemma 1.

References

[1] D.J. Aldous. The random walk construction for spanning trees and uniform labelled trees. *SIAM J. Discrete Math*, 3:450–465, 1990.

[2] S.F. Altschul and B.W. Erickson. Significance of nucleotide sequence alignments: A method for random sequence permutation that preserves dinucleatide and codon usage. *Mol. Biol. Evol.*, 2:526–538, 1985.

[3] D. Bayer and P. Diaconis. Trailing the dovetail shuffle to its lair. *Ann. Appl. Probab.*, 2:294–313, 1992.

[4] A. Broder. Generating random spanning trees. In *Proc. 30th IEEE Symp. on Foundations of Computer Science*, pages 442–447, 1989.

[5] P.G. Doyle and J.L. Snell. *Random Walks and Electric Networks*. Mathematical Assoc. of America, Washington, DC, 1984.

[6] S. Even. *Graph Algorithms*. Computer Science Press, Potomac, MD, 1979.

[7] W.M. Fitch. Random sequences. *J. Mol. Biol.*, 163:171–176, 1983.

[8] G. Frobenius. Über matrizen aus positiven elementen. *Sitzber. Akad. Wiss. Berlin, Phys. math. Kl.*, pages 471–476, 1908.

[9] G. Frobenius. Über matrizen aus positiven elementen. *Sitzber. Akad. Wiss. Berlin, Phys. math. Kl.*, pages 514–518, 1909.

[10] G. Frobenius. Über matrizen aus nicht negativen elementen. *Sitzber. Akad. Wiss. Berlin, Phys. math. Kl.*, pages 456–477, 1912.

[11] F. R. Gantmacher. *Application of the theory of matrices*. Interscience Publishers, Inc., New York, NY, 1959.

[12] S. Karlin and S.F. Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc Natl Acad Sci U S A*, 87:2264–2268, 1994.

[13] S. Karlin and V. Brendel. Chance and statistical significance in protein and DNA sequence analysis. *Science*, 257:39–49, 1992.

[14] S. Karlin, I. Ladunga, and B.E. Blaisdell. Hetrogeneity of genomes: Measures and calues. *Proc Natl Acad Sci U S A*, 91:12837–12842, 1994.

[15] A.C.B.L. Bulletin (American Contract Bridge League), May 1995.

[16] O. Perron. Grundlagen für eine theorie des jacobischen kettenbruchalgorithmus. *Math. Ann.*, 64:1–76, 1907.

[17] O. Perron. Zur theorie der matrices. *Math. Ann.*, 64:248–263, 1907.

[18] P.A. Pevzner. DNA physical mapping and alternating eulerian cycles in colored graphs. *Algorithmica*, 13:77–105, 1995.

[19] E. Ukkonen. Approximate string matching with q-grams and maximal matches. *Theoretical Computer Science*, 92:191–211, 1992.

[20] R. Unger, G. Avrahami, D. Harel, and J. L. Sussman. Simple general shuffling scheme which preserves fragment frequencies up to any required length. In *Proc. Macromolecules, Genes, and Computers conf.*, 1986.