Identifying GC-Rich DNA Segments via Linear-Time Algorithm for Maximum Average Binary Substring*

Hsueh-I Lu
Institute of Information Science
Academia Sinica
Taipei 115, Taiwan
http://www.iis.sinica.edu.tw/~hil/
Email: hil@iis.sinica.edu.tw
Phone: +886-2-2788-3799 ext 1731

Abstract

The GC-ratio *of a DNA segment S is the number of GC-base pairs in S divided by the total number of base pairs in S. Due to the importance of GC-rich DNA segments in gene recognition and comparative genomics, we address how to efficiently identify sufficiently long segments of a DNA sequence whose GC-ratios are high. The kernel issue is the following algorithmic string problem: Given an n-bit binary string B and a positive integer L, we would like to compute a substring of B with length no less than L whose* average, *i.e., the number of ones in the substring divided by the length of the substring, is maximized. Huang gave the first polynomial-time algorithm, running in $O(nL)$ time, for this maximum-average substring problem. Lin, Jiang, and Chao improved the time complexity to $O(n \log L)$ based upon the* right-skew decomposition *of B. We give the first $O(n)$-time algorithm for this problem by (a) providing an upper bound on the number of right-skew segments in the right-skew decomposition of a binary string, (b) working simultaneously on the right-skew decompositions of B and B's reversed string, and (c) performing some $O(n)$-time preprocessing that allows maximum average substrings with lengths $L + O((\log \log n)^3)$ to be obtainable $O(n)$ time.*

1 Introduction

Nonuniformity of nucleotide composition within genomic sequences was first revealed through thermal melting and gradient centrifugation experiments [17, 20]. The GC content of the DNA sequences in all organisms varies from 25% to 75%. GC-ratios have the greatest variations among bacteria's DNA sequences, while the typical GC-ratios of mammalian genomes stay in 45-50%. The GC content of human DNA varies widely throughout the genome, ranging between 30% and 60%. Despite intensive research effort in the past two decades, the underlying causes of the observed heterogeneity remain contested [1–3, 5, 6, 8, 9, 14, 28, 30]. Nekrutenko and and Li [23] observed that the compositional heterogeneity is highly correlated to the GC content of the genomic sequences. Other investigations showed that gene length [4], gene density [32], patterns of codon usage [26], distribution of different classes of repetitive elements [4, 27], number of isochores [1], lengths of isochores [23], and recombination rate within chromosomes [10] are all correlated with GC content. More research related to GC-rich segments can be found in [11, 13, 16, 18, 21, 22, 25, 29, 31] and the references therein.

Although GC-rich segments of DNA sequences are important in gene recognition and comparative genomics, only a couple of algorithms for identifying GC-rich segments appeared in the literature. A widely used window-based approach is based upon the GC-content statistics of a fixed-length window [7, 12, 23, 24]. Due to the fixed length of windows, these practically fast approaches are likely to miss GC-rich segments that span more than one window. Huang [15] proposed an algorithm to accommodate windows with variable lengths. Specifically, by assigning $-p$ points to each AT-pair and $1 - p$ points to each GC-pair, where p is a number with $0 \leq p \leq 1$, Huang gave a linear-time algorithm for computing a segment

*Research supported in part by NSC grants NSC-89-2218-E-001-017 and NSC-90-2218-E-001-005.

of length no less than L whose score is maximized. As observed by Huang, however, this approach tends to output segments that are significantly longer than the given L.

The most natural formulation for the problem of identifying GC-rich segments, also due to Huang [15], is to look for a segment of length no less than L that has the maximum average score, where each AT-pair is assigned zero point and each GC-pair is assigned one point. (One can easily verify that the problem remains the same even if p points are assigned to each AT-pair and q points are assigned to each GC-pair, where p and q are any arbitrary numbers with $p < q$.) Clearly, this problem is exactly the same as the algorithmic string problem of looking for a substring with at least L bits in the input n-bit string whose average number of one bits is maximized. With standard dynamic-programming techniques, Huang [15] gave the first polynomial-time algorithm, running in time $O(nL)$, for this maximum average segment problem. With a brilliant technique of decomposing the input sequence into *right-skew* segments, Lin, Jiang, and Chao [19] improved the running time to $O(n \log L)$. In the present paper, we further improve the running time and give the first known $O(n)$-time algorithm for this problem.[1]

Our algorithm is based upon Lin et al.'s linear-time right-skew decomposition for any sequence of numbers [19]. While focusing on binary strings, we observe that the number of right-skew components in the right-skew decomposition of any m-bit string is $O(m^{2/3})$. We first partition the input n-bit string into *short* substrings, each with $O(\log^3 n)$ bits, and further partition some small substrings into *tiny* substrings, each with $O((\log \log n)^3)$ bits. The above upper bound ensures that the total number of right-skew components in short and tiny substrings is sublinear, implying that the maximum average substring starting from some $O(n/\log\log n)$ positions of the input string can be obtained in linear time. Applying the same technique on the reversed string of the input string, we then obtain the maximum average substring that ends at some $O(n/\log\log n)$ positions of S. It turns out that the strings requiring further attention have to consist of $L + O((\log\log n)^3)$ bits. We finally provide an $O(n)$-time preprocessing which allows maximum average substrings with $L + O((\log\log n)^3)$ bits to be computable in $O(n)$ time.

The rest of the paper is organized as follows. Section 2 gives the preliminaries. Section 3 describes our linear-time algorithm. Section 4 concludes the paper with some open problems.

[1] The algorithms of Huang [15] and Lin, Jiang, and Chao [19] work for any sequence of numbers not necessarily binary, though.

2 Preliminaries

Let S be an n-bit binary string, i.e., the i-th character $S[i]$ of S is either 0 or 1 for each $i = 0, 1, 2, \ldots, n-1$. For any indices i and j with $0 \le i < j \le n$, let *substring* $S[i,j]$ consist of the $j - i$ bits $S[i], S[i+1], \ldots, S[j-1]$. The *sum* $\sigma(i,j)$ of $S[i,j]$ and the *average* $\mu(i,j)$ of $S[i,j]$ are defined as follows.

$$\begin{aligned} \sigma(i,j) &= S[i] + S[i+1] + \cdots + S[j-1]; \\ \mu(i,j) &= \frac{\sigma(i,j)}{j-i}. \end{aligned}$$

Clearly, $0 \le \mu(i,j) \le 1$ holds for any substring $S[i,j]$ of S. Since $\sigma(i,j) = \sigma(0,j) - \sigma(0,i)$, each $\sigma(i,j)$ and each $\mu(i,j)$ can be obtained in $O(1)$ time after we precompute $\sigma(0,1), \sigma(0,2), \ldots, \sigma(0,n)$ in $O(n)$ time.

Substring $S[i,j]$ with $i < j$ is *right-skew* if (a) $j = i+1$ or (b) $\mu(i,k) \le \mu(k,j)$ holds for each $k = i+1, \ldots, j-1$. For example, if S is as shown in Figure 1, then $S[0,1]$, $S[1,5]$, $S[8,12]$, $S[10,13]$, $S[13,15]$, and $S[15,17]$ are all right-skew, while $S[0,5]$ and $S[7,10]$ are not.

A *right-skew decomposition* of any substring $S[p,q]$, not necessarily right-skew, is a nonempty set of ℓ indices $i_1, i_2, \ldots, i_\ell$ with $p = i_1 < i_2 < \cdots < i_\ell = q$ such that all substrings

$$S[i_1, i_2], S[i_2, i_3], \ldots, S[i_{\ell-1}, i_\ell]$$

are right-skew and

$$\mu(i_1, i_2) > \mu(i_2, i_3) > \cdots > \mu(i_{\ell-1}, i_\ell).$$

Clearly, if $\{i_1, i_2, \ldots, i_\ell\}$ with $i_1 < i_2 < \cdots < i_\ell$ is a right-skew decomposition of $S[p,q]$, then $\{i_j, i_{j+1}, \ldots, i_\ell\}$ is a right-skew decomposition of $S[i_j, q]$ for each $j = 1, 2, \ldots, \ell$. For example, $\{0, 1, 5, 8, 15, 17\}$ is a right-skew decomposition of $S[0,17]$ and $\{9, 15, 16\}$ is a right-skew decomposition of $S[9,16]$, where S is as given in Figure 1.

Lemma 1 (See [19]). *For any indices i and j with $0 \le i < j \le n$, the right-skew decomposition of $S[i,j]$ is unique and obtainable in $O(j-i)$ time.*

Let $D_S(i,j)$ denote the right-skew decomposition of $S[i,j]$.

$$S = 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0,$$
$$\text{index} = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16$$

Figure 1: A binary string.

Lemma 2 (See [19]). *For each $i = 0, 1, \ldots, n-2$, $D_S(i, n) \subseteq \{i\} \cup D_S(i+1, n)$.*

If $P = \{p_1, p_2, \ldots, p_k, p_{k+1}\}$, where $p_1 < p_2 < \cdots < p_k < p_{k+1}$, then we define

$$D_S(P) = \bigcup_{i=1}^{k} D_S(p_i, p_{i+1}).$$

For example, if S is as given in Figure 1 and $P = \{0, 5, 10, 15, 17\}$, then $D_S(P) = \{0, 1, 5, 8, 10, 13.15, 17\}$. If it is clear from the context which string is referred to, we may omit the subscripts in $D_S(i,j)$ and $D_S(P)$. The following lemma is a generalization of Lemma 2 that is crucial to our improved result.

Lemma 3. *If $j \in D_S(P)$, then $D_S(j, n) \subseteq D_S(P)$.*

Proof. It suffices to prove the following statement.

> If $S[i,j]$ is right-skew, then $D_S(i,n) \subseteq \{i\} \cup D_S(j,n)$.

Assume for a contradiction that the above statement does not hold. It follows from Lemma 2 that some index p with $i < p < j$ is the largest index in $D_S(i,n) - (\{i\} \cup D_S(j,n))$. Let q be the least index in $D_S(i,n)$ that is larger than p. Clearly, $j \le q$ and $S[p,q]$ is right-skew. Since $i < p < j \le q$ and $S[i,j]$ and $S[p,q]$ are both right-skew, we know $\mu(i,p) \le \mu(p,j) \le \mu(p,q)$. However, by $\{p,q\} \subseteq D_S(i,n)$ and $p < q$, we have the contradictory inequality $\mu(i,p) > \mu(p,q)$. □

The following lemma provides a sublinear upper bound on the number of right-skew components in the right-skew decomposition of any binary string.

Lemma 4. $|D_S(i,j)| = O((j-i)^{2/3})$.

Proof. Let $\ell = j - i$. We say $S[p,q]$ is a right-skew substring determined by $D_S(i,j)$ if p and q are two indices in $D_S(i,j)$ with $p < q$ such that no index r in $D_S(i,j)$ satisfies $p < r < q$. A right-skew substring $S[p,q]$ determined by $D_S(i,j)$ is *long* (respectively, *short*) if $q - p \ge \ell^{1/3}$ (respectively, $q - p < \ell^{1/3}$). Clearly, the number of long right-skew substrings determined by $D_S(i,j)$ is at most $\frac{\ell}{\ell^{1/3}} = O(\ell^{2/3})$. Observe that any two right-skew substrings $S[p_1,q_1]$ and $S[p_2,q_2]$ determined by $D_S(i,j)$ have distinct averages, i.e., $\mu(p_1,q_1) \ne \mu(p_2,q_2)$. Thus, the number of length-k right-skew substrings determined by $D_S(i,j)$ is at most $k+1$, since the average of a length-k substring has to be one of $\frac{0}{k}, \frac{1}{k}, \ldots, \frac{k}{k}$. Therefore, the number of short right-skew substrings determined by $D_S(i,j)$ is also no more than $2+3+4+\cdots+\lceil \ell^{1/3} \rceil = O(\ell^{2/3})$. Now that the numbers of long and short right-skew substrings determined by $D_S(i,j)$ are both $O(\ell^{2/3})$, the lemma is proved. □

Now we give an overview of Lin et al.'s algorithm [19]. Their algorithm proceeds in $O(n)$ iterations, each takes $O(\log n)$ time.[2] The i-th iteration finds an index $g(i)$ with $g(i) - i \ge L$ and $\mu(i, g(i)) = \max_{i+L \le j \le n} \mu(i,j)$. In [19], $g(i)$ is called a *good partner* of i in S. As shown in [19], it is not difficult to prove that $D_S(i+L, n)$ always contains a good partner $g(i)$ of i in S. Moreover, if $D_S(i+L, n) = \{i_1, i_2, \ldots, i_\ell\}$ where $i_1 < i_2 < \cdots < i_\ell$, then the sequence $\mu(i,i_1), \mu(i,i_2), \ldots, \mu(i,i_\ell)$ is a bi-tonic, i.e., first monotonically increasing and then monotonically decreasing. To enable $O(\log n)$-time binary search of $g(i)$ in $D(i+L,n)$, a *jumping table* of depth $O(\log n)$ is required for each index in $D_S(i+L,n)$, where the k-th entry of the jumping table of depth ℓ for j, denoted $J(j,k,\ell)$, is the 2^k-th smallest index in $D_S(j,n)$, for each $k = 1, 2, \ldots, \ell$. Suppose M is a subset of $0, 1, \ldots, n-1$ such that $j \in M$ implies $J(i,k,\ell) \in M$ for each $k = 1, 2, \ldots, \ell$. Lin et al. showed that $J(j,k,\ell)$ for all $j \in M$ and $k = 1, 2, \ldots, \ell$ can be computed in $O(n + \ell|M|)$ time, implying that the jumping tables of depth $O(\log n)$ for all indices from 0 to $n-1$ can be obtained in $O(n \log n)$ time. Now that each iteration of Lin et al.'s algorithm finds a good partner $g(i)$ of i by binary search in $O(\log n)$ time with the help

[2] The algorithm presented in [19] is slightly more complicated than the simplified version explained here. Those omitted complicated part is to ensure that each iteration runs in $O(\log L)$ time. However, for the purpose of describing our $O(n)$-time algorithm, the simplified $O(n \log n)$-time version of Lin et al.'s algorithm suffices.

of the jumping tables of depth $O(\log n)$ for all indices from 0 to $n-1$. Therefore, the overall time complexity of Lin et al.'s algorithm is $O(n \log n)$. Let us summarize the above overview into the following lemma.

Lemma 5 (See [19]). *The following statements hold for any string S.*

1. *Let M be a subset of $\{0, 1, 2, \ldots, n-1\}$ such that $j \in M$ implies $J(j, k, \ell) \in M$ for each $k = 1, 2, \ldots, \ell$. Then $J(j, k, \ell)$ for all $j \in M$ and $k = 1, 2, \ldots, \ell$ can be computed in $O(n + \ell |M|)$ time.*
2. *For each index i for S, there always exists a good partner of i in $D_S(i + L, n)$.*
3. *If the jumping tables of depth ℓ for the indices j in $D(i + L, n)$ with $j \leq i + L + 2^\ell$ are available, then such an index j with maximum $\mu(i, j)$ can be found in $O(\ell)$ time.*

3 Our linear-time algorithm

This section describes our $O(n)$-time algorithm for the maximum average substring problem. In order to improve the running time to $O(n)$, we have to get around with two $O(n \log n)$ steps in Lin et al.'s algorithm [19]: the precomputation for jumping tables, and the binary searches for obtaining good partners for all indices. More precisely, we can only afford (a) computing jumping tables of depth ℓ for $O(n/\ell)$ indices, and (b) looking for good partners for $O(n/\ell)$ indices in a range of length 2^ℓ. Let $B = \lceil \log_2^3 n \rceil$ and $b = \lceil (\log_2 \log_2 n)^3 \rceil$. Let R be the reversed string of S. Roughly speaking, our algorithm runs in the following three phases:

1. Carefully choose $O(n/\log n)$ indices i of S such that if $g(i) - i \geq B$ holds for any of such i, then $g(i)$ can be found in $O(\log n)$ time. A similar procedure has to be applied to R as well.
2. Carefully choose $O(n/\log \log n)$ indices i of S such that if $B \geq g(i) - i \geq b$ holds for any of such i, then $g(i)$ can be found in $O(\log \log n)$ time. A similar procedure has to be applied to R as well.
3. Find $g(i)$ for all indices i such that $g(i) - i \leq b$.

We will show that the above three phases suffice. The following lemma is used several time in our analysis.

Lemma 6. *If I consists of $O(n/\ell)$ indices for S and*

$$\left| \bigcup_{i \in I} D_S(i + L, n) \right| = O(n/\ell),$$

then the maximum $\mu(i, j)$ over all i and j such that $i \in I$ and $j \leq i + L + 2^\ell$ can be found in $O(n)$ time.

Proof. Straightforward by Lemmas 3, 5(1), and 5(3). □

The rest of the section elaborates each phase per subsection.

3.1 Phase 1

Define

$$P_{\text{short}} = \{p \mid p \bmod B \equiv 0 \text{ and } 0 \leq p < n\} \cup \{n\}.$$

Clearly, $|P_{\text{short}}| = \Theta(n/\log^3 n)$. By Lemma 4, we have $|D_S(P_{\text{short}})| = O(n/\log n)$ and $|D_R(P_{\text{short}})| = O(n/\log n)$. In this phase, we consider the following cases.

Case 1: $g(i) \in D_S(P_{\text{short}})$. We compute this kind of maximum from R. Observe that $R[p, q]$ is the reversed string of $S[n-p, n-q]$. Without loss of generality, we may assume that n is an integral multiple of B. It follows that $p \in P_{\text{short}}$ if and only if $n - p \in P_{\text{short}}$. Let $j = n - g(i)$. This present case is equal to the problem of computing the good partner $g(j)$ of j in R. By Lemma 6, if $g(j) \in D_R(P_{\text{short}})$, then maximum of this kind can be computed in $O(n)$ time. Note that if $g(j) \notin D_R(P_{\text{short}})$, then, by Lemmas 2 and 3, we know that $g(j) \notin D_R(P_{\text{short}})$ implies $g(j) - j \leq B$. We will take care of this remaining case in the next phase.

Case 2: $g(i) \notin D_S(P_{\text{short}})$. Now we have $i + L \notin D_S(P_{\text{short}})$ and $g(i) \notin D_S(P_{\text{short}})$. Let j be the smallest index in $D_S(P_{\text{short}})$ such that $i < j$. One can show $g(i) < j$ as follows. By Lemma 3, $D(j, n) \subseteq D_S(P_{\text{short}})$. Also, by Lemma 3, if $g(i) \geq j$, then $g(i) \in D(j, n)$ and thus $g(i) \in D_S(P_{\text{short}})$. It follows that $g(i) < j$. Thus, we have $g(i) - j \leq B$. We take care of this case in the next phase.

3.2 Phase 2

In this phase we focus on the case that an index i (respectively, j) and its good partner $g(i)$

(respectively, $g(j)$) in S (respectively, R) have difference no more than $L+B$, i.e., $g(i)-i \le L+B$ and $g(j)-j \le L+B$. For brevity, we explain this phase for S only. Similarly, we define

$$P_{\text{tiny}} = \{p \mid p \bmod b \equiv 0 \text{ and } 0 \le p < n\} \cup \{n\}.$$

Case 1: $g(i) \in D_S(P_{\text{tiny}})$**.** Once again we compute this kind of maximum from R. Without loss of generality, we may assume that n is an integral multiple of b. It follows that $p \in P_{\text{tiny}}$ if and only $n-p \in P_{\text{tiny}}$. Let $j = n - g(i)$. This present case is equal to the problem of computing the good partner $g(j)$ of j in R. By Lemma 6, if $g(j) \in D_R(P_{\text{tiny}})$, then maximum of this kind can be computed in $O(n)$ time. Note that if $g(j) \notin D_R(P_{\text{tiny}})$, then, by Lemmas 2 and 3, we know that $g(j) \notin D_R(P_{\text{tiny}})$ implies that $g(j)-j \le b$. Will will take care of this remaining case in the next phase.

Case 2: $g(i) \notin D_S(P_{\text{tiny}})$**.** Now we have $i + L \notin D_S(P_{\text{tiny}})$ and $g(i) \notin D_S(P_{\text{tiny}})$. Let j be the smallest index in $D_S(P_{\text{tiny}})$ such that $i < j$. One can show $g(i) < j$ as follows. By Lemma 3, $D(j,n) \subseteq D_S(P_{\text{tiny}})$. Also, by Lemma 3, if $g(i) \ge j$, then $g(i) \in D(j,n)$ and thus $g(i) \in D_S(P_{\text{tiny}})$. It follows that $g(i) < j$. Thus, we have $g(i)-j \le b$. We take care of this case in the next phase.

3.3 Phase 3

It is not difficult to see that by continuing the above process of breaking S and R into right-skew components of smaller and smaller sizes, one can obtain an $O(n \log^* n)$-time algorithm for the maximum average substring problem. In order to further improve the running time to $O(n)$, we adapt an $O(n)$-time preprocessing that enable the remaining cases to be computable in $O(n)$ time. Recall that it remains to consider the case that an index i (respectively, j) and its good partner $g(i)$ (respectively, $g(j)$) in S (respectively, R) have difference no more than $L+b$, i.e., $g(i)-i \le L+b$ and $g(j)-j \le L+b$. For brevity, we explain this phase for S only.

Consider a table M whose (x,y) entry contains the index z satisfying the following condition: If C is a binary string of $L+b$ bits such that (1) the number of '1' in the first L bits of C is x, and (2) y is the binary string consisting of the last b bits of C, then z is the good partner of index 0 in C. Observe that (a) the number of possible values for y is $O(n^c)$ for some constant $c < 1$; (b) the number of possible values for x is $O(L)$; and (c) each (x,y) entry of M can be obtained in $O(b)$ time. One can verify that if M is already available, then the left-over case to be coped with in this phase can be resolved in $O(n)$ time, since each good partner $g(i)$ with $g(i)-i \le L+b$ can be computed in $O(1)$ time by looking up M. The problem is that M has $\omega(n)$ entries, so it is impossible to compute all the (x,y) entries in $O(n)$ time in a preprocessing. However, we can show as follows that in order resolve the case in this phase, it suffices to have b rows of M precomputed. Specifically, let $\sigma^* = \max_{0 \le i < n} \sigma(i, i+L)$, which is computable in $O(n)$ time. Clearly, if

$$\frac{x+b}{L+b} < \frac{\sigma^*}{L},$$

then this number of '1' bits in the first L bits of any string can never let the substring of S in consideration be the maximum average substring of S. By simple calculation, one can see that $x < \sigma^* - b$ implies the above inequality. Therefore, only $O(bn^c)$ entries of M requires precomputation, each takes $O(b)$ time. It follows that this phase can also be done in $O(n)$ time.

4 Concluding Remarks

It would be interesting to come up with a polynomial-time algorithm for computing a maximum average substring whose length is between two given positive L and U. As noted by Huang [15], if $U \ge 2L$, then the generalized problem is in fact the same as the problem only with lower bound L. For the case with $U < 2L$, no polynomial-time algorithm is known to the best of our knowledge.

Also, Lemma 4 can be generalized as follows: If the input string S is t-ary, i.e., each number of S belongs to $\{0,1,2,3,\ldots,t-1\}$, then one can analogously prove that the number of right-skew substrings in the right-skew decomposition of S is $O(m^{2/3}t^{1/3})$, where m is the length of S (by replacing $m^{1/3}$ with $(m/t)^{1/3}$ in the proof for Lemma 4). Therefore, an interesting problem is to generalize our linear-time algorithm for $t > 2$.

Acknowledgment

We thank Kun-Mao Chao, Tao Jiang, and Yaw-Ling Lin for providing [19]. We also thank Ming-Yang Kao, Tao Jiang, Yaw-Ling Lin, Kun-Mao

Chao, Chuan-Yi Tang, Tak-Man Ma, and Alva Chung-Ming Lin for helpful comments and suggestions.

References

[1] G. Barhardi. Isochores and the evolutionary genomics of vertebrates. *Gene*, 241:3–17, 2000.

[2] G. Bernardi and G. Bernardi. Compositional constraints and genome evolution. *Journal of Molecular Evolution*, 24:1–11, 1986.

[3] B. Charlesworth. Genetic recombination: patterns in the genome. *Current Biology*, 4:182–184, 1994.

[4] L. Duret, D. Mouchiroud, and C. Gautier. Statistical analysis of vertebrate sequences reveals that long genes are scarce in GC-rich isochores. *Journal of Molecular Evolution*, 40:308–371, 1995.

[5] A. Eyre-Walker. Evidence that both G+C rich and G+C poor isochores are replicated early and late in the cell cycle. *Nucleic Acids Research*, 20:1497–1501, 1992.

[6] A. Eyre-Walker. Recombination and mammalian genome evolution. *Proceedings of the Royal Society of London Series B, Biological Science*, 252:237–243, 1993.

[7] C. A. Fields and C. A. Soderlund. gm: a practical tool for automating DNA sequence analysis. *Computer Applications in the Biosciences*, 6:263–270, 1990.

[8] J. Filipski. Correlation between molecular clock ticking, codon usage fidelity of DNA repair, chromosome banding and chromatin compactness in germline cells. *FEBS Letters*, 217:184–186, 1987.

[9] M. P. Francino and H. Ochman. Isochores result from mutation not selection. *Nature*, 400:30–31, 1999.

[10] S. M. Fullerton, A. B. Carvalho, and A. G. Clark. Local rates of recombinantion are positively corelated with GC content in the human genome. *Molecular Biology and Evolution*, 18(6):1139–1142, 2001.

[11] P. Guldberg, K. Gronbak, A. Aggerholm, A. Platz, P. thor Straten, V. Ahrenkiel, P. Hokland, and J. Zeuthen. Detection of mutations in GC-rich DNA by bisulphite denaturing gradient gel electrophoresis. *Nucleic Acids Research*, 26(6):1548–1549, 1998.

[12] R. C. Hardison, D. Drane, C. Vandenbergh, J.-F. F. Cheng, J. Mansverger, J. Taddie, S. Schwartz, X. Huang, and W. Miller. Sequence and comparative analysis of the rabbit alpha-like globin gene cluster reveals a rapid mode of evolution in a G+C rich region of mammalian genomes. *Journal of Molecular Biology*, 222:233–249, 1991.

[13] W. Henke, K. Herdel, K. Jung, D. Schnorr, and S. Loening. Betaine improves the PCR amplification of GC-rich DNA sequences. *Nucleic Acids Research*, 25(19):3957–3958, 1997.

[14] G. P. Holmquist. Chromosome bands, their chromatin flavors, and their functional features. *American Journal of Human Genetics*, 51:17–37, 1992.

[15] X. Huang. An algorithm for identifying regions of a DNA sequence that satisfy a content requirement. *Computer Applications in the Biosciences*, 10:219–225, 1994.

[16] K. Ikehara, F. Amada, S. Yoshida, Y. Mikata, and A. Tanaka. A possible origin of newly-born bacterial genes: significance of GC-rich nonstop frame on antisense strand. *Nucleic Acids Research*, 24(21):4249–4255, 1996.

[17] R. B. Inman. A denaturation map of the 1 phage DNA molecule determined by electron microscopy. *Journal of Molecular Bilogy*, 18:464–476, 1966.

[18] R. Jin, M.-E. Fernandez-Beros, and R. P. Novick. Why is the initiation nick site of an AT-rich rolling circle plasmid at the tip of a GC-rich cruciform? *The EMBO Journal*, 16(14):4456–4466, 1997.

[19] Y.-L. Lin, T. Jiang, and K.-M. Chao. Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. Submitted for publication, 2001.

[20] G. Macaya, J.-P. Thiery, and G. Bernardi. An approach to the organization of eukaryotic genomes at a macromolecular level. *Journal of Molecular Biology*, 108:237–254, 1976.

[21] C. S. Madsen, C. P. Regan, and G. K. Owens. Interation of CArG elements and a GC-rich repressor element in transcriptional regulation of the smooth muscle myosin heavy chain gene in vascular smooth muscle cells. *Journal of Biological Chemistry*, 272(47):29842–29851, 1997.

[22] S.-i. Murata, P. Herman, and J. R. Lakowicz. Texture analysis of fluorescence lifetime images of AT- and GC-rich regions in nuclei. *Journal of Hystochemistry and Cytochemistry*, 49:1443–1452, 2001.

[23] A. Nekrutenko and W.-H. Li. Assessment of compositional heterogeneity within and between eukaryotic genomes. *Genome Research*, 10:1986–1995, 2000.

[24] P. Rice, I. Longden, and A. Bleasby. EMBOSS: the European molecular biology open software suite. *Trends in Genetics*, 16:276–277, 2000.

[25] L. Scotto and R. K. Assoian. A GC-rich domain with bifunctional effects on mRNA and protein levels: implications for control of transforming growth factor beta 1 expression. *Molecular and Cellular Biology*, 13(6):3588–3597, 1993.

[26] P. M. Sharp, M. Averof, A. T. Lloyd, G. Matassi, and J. F. Peden. DNA sequence evolution: the sounds of silence. *Philosophical Transactions of the Royal Society of London Series B, Biological Sciences*, 349:241–247, 1995.

[27] P. Soriano, M. Meunier-Rotival, and G. Bernardi. The distribution of interspersed repeats is nonuniform and conserved in the mouse and human genomes. *Proceedings of the National Academy of Sciences of the United States of America*, 80:1816–1820, 1983.

[28] N. Sueoka. Directional mutation pressure and neutral molecular evolution. *Proceedings of the National Academy of Sciences of the United States of America*, 80:1816–1820, 1988.

[29] Z. Wang, E. Lazarov, M. O'Donnel, and M. F. Goodman. Resolving a fidelity paradox: Why Escherichia coli DNA polymerase II makes more base substitution errors in at- compared to GC-rich DNA. *Journal of Biological Chemistry*, To appear.

[30] K. H. Wolfe, P. M. Sharp, and W.-H. Li. Mutation rates differ among regions of the mammalian genome. *Nature*, 337:283–285, 1989.

[31] Y. Wu, R. P. Stulp, P. Elfferich, J. Osinga, C. H. Buys, and R. M. Hofstra. Imrpvoed mutation detection in GC-rich DNA fragments by combined DGGE and CDGE. *Nucleic Acids Research*, 27(15):e9, 1999.

[32] S. Zoubak, O. Clay, and G. Bernardi. The gene distribution of the human genome. *Gene*, 174:95–102, 1996.