Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Journal of Computer and System Sciences 70 (2005) 128–144

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

www.elsevier.com/locate/jcss

Linear-time algorithms for computing maximum-density sequence segments with bioinformatics applications[☆]

Michael H. Goldwasser[a], Ming-Yang Kao[b,1], Hsueh-I Lu[c,d,2]

[a]*Department of Mathematics and Computer Science, Saint Louis University, 221 N. Grand Blvd, St. Louis, MO 63103-2007, USA*
[b]*Department of Computer Science, Northwestern University, Evanston, IL 60201, USA*
[c]*Institute of Information Science, Academia Sinica, 128 Academia Road, Section 2, Taipei 115, Taiwan*
[d]*Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan*

Received 3 October 2002; received in revised form 3 August 2004

Available online 18 November 2004

Abstract

We study an abstract optimization problem arising from biomolecular sequence analysis. For a sequence A of pairs (a_i, w_i) for $i = 1, \ldots, n$ and $w_i > 0$, a *segment* $A(i, j)$ is a consecutive subsequence of A starting with index i and ending with index j. The *width* of $A(i, j)$ is $w(i, j) = \sum_{i \leqslant k \leqslant j} w_k$, and the *density* is $(\sum_{i \leqslant k \leqslant j} a_k)/w(i, j)$. The *maximum-density segment* problem takes A and two values L and U as input and asks for a segment of A with the largest possible density among those of width at least L and at most U. When U is unbounded, we provide a relatively simple, $O(n)$-time algorithm, improving upon the $O(n \log L)$-time algorithm by Lin, Jiang and Chao. We then extend this result, providing an $O(n)$-time algorithm for the case when both L and U are specified.
© 2004 Elsevier Inc. All rights reserved.

Keywords: Bioinformatics; Sequences; Density

[☆] A preliminary version of these results appeared under the title, "Fast Algorithms for Finding Maximum-Density Segments of a Sequence with Applications to Bioinformatics," in: R. Guigó, D. Gusfield (Eds.), Proceedings of the Second Workshop on Algorithms in Bioinformatics (WABI), Lecture Notes in Computer Science, vol. 2452, Springer, Berlin, 2002, pp. 157–171.

E-mail addresses: goldwamh@slu.edu, (M.H. Goldwasser), kao@cs.northwestern.edu (M.-Y. Kao), hil@iis.sinica.edu.tw (H.-I. Lu)

URLs: http://euler.slu.edu/∼goldwasser, http://www.cs.northwestern.edu/∼kao, http://www.iis.sinica.edu.tw/∼hil

[1] Supported in part by NSF Grant EIA-0112934.

[2] Supported in part by NSC Grant NSC-90-2218-E-001-005.

doi:10.1016/j.jcss.2004.08.001

1. Introduction

Non-uniformity of nucleotide composition within genomic sequences was first revealed through thermal melting and gradient centrifugation experiments [23,27]. 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 are between 45% and 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 [3–5,9,10,12,13,20,37,39]. Researchers [30,36] observed that the compositional heterogeneity is highly correlated to the GC content of the genomic sequences. Other investigations showed that gene length [8], gene density [41], patterns of codon usage [34], distribution of different classes of repetitive elements [8,35], number of isochores [3], lengths of isochores [30], and recombination rate within chromosomes [14] are all correlated with GC content. More research related to GC-rich segments can be found in [17,19,22,24,28,29,32,38,40] and the references therein.

Although GC-rich segments of DNA sequences are important in gene recognition and comparative genomics, only a few algorithms for identifying GC-rich segments appear in the literature. A widely used approach measures the GC-content statistics for fixed-length windows [11,18,30,31]. Due to the fixed length of these windows, the approaches are simple and efficient yet likely to miss GC-rich segments that do not precisely align with a window. Huang [21] 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 $0 \leqslant p \leqslant 1$, Huang gave a linear-time algorithm for computing a segment of length no less than L whose score is maximized. However, as observed by Huang, this approach tends to output segments that are significantly longer than the given L.

In this paper, we study the following abstraction of the problem. Let A be a sequence of pairs (a_i, w_i) for $i = 1, \ldots, n$ and $w_i > 0$. A *segment* $A(i, j)$ is a consecutive subsequence of A starting with index i and ending with index j. The *width* of $A(i, j)$ is $w(i, j) = \sum_{i \leqslant k \leqslant j} w_k$, and the *density* is $(\sum_{i \leqslant k \leqslant j} a_k)/w(i, j)$. Let L and U be positive values with $L \leqslant U$. The *maximum-density segment* problem takes A, L, and U as input and asks for a segment of A with the largest possible density among those of width at least L and at most U. This generalizes a previously studied model, which we term the *uniform* model, in which $w_i = 1$ for all i. All of the previous work discussed in this section involves the uniform model. We introduce the generalized model as it might be used to compress a sequence A of real numbers to reduce its sequence length and thus its density-analysis time in practice or theory.

In its most basic form, the sequence A corresponds to the given DNA sequence, where $a_i = 1$ if the corresponding nucleotide in the DNA sequence is G or C; and $a_i = 0$ otherwise. In the work of Huang, sequence entries took on values of p and $1 - p$ for some real number $0 \leqslant p \leqslant 1$. More generally, we can look for regions where a given set of patterns occur very often. In such applications, a_i could be the relative frequency with which the corresponding DNA character appears in the given patterns. Further natural applications of this problem can be designed for sophisticated sequence analyses such as mismatch density [33], ungapped local alignments [1], and annotated multiple sequence alignments [36].

Nekrutendo and Li [30], and Rice et al. [31] employed algorithms for the case where $L = U$. This case is trivially solvable in $O(n)$ time using a sliding window of the appropriate width. More generally, when $L \neq U$, this yields a trivial $O(n(U - L + 1))$ algorithm. Huang [21] studied the case where $U = n$, i.e., there is effectively no upper bound on the width of the desired segment. He observed that an optimal segment exists with width at most $2L - 1$. Therefore, this case is equivalent to the case with $U = 2L - 1$

and thus can be solved in $O(nL)$ time. Recently, Lin et al. [26] gave an $O(n \log L)$-time algorithm for this case based on the introduction of right-skew partitions of a sequence.

In this paper, we present an $O(n)$-time algorithm solving the maximum-density segment problem. Our techniques exploit the structure of locally optimal segments to improve upon the $O(n \log L)$-time algorithm of Lin et al. [26], while also extending the results to arbitrary values of U and to the non-uniform model. The remainder of this paper is organized as follows. Section 2 introduces some notation and definitions. In Section 3, we carefully review the previous work of Lin, Jiang and Chao, in which they introduce the concept of right-skew partitions. Our main results are presented in Section 4: first a simple, $O(n)$-time algorithm for the special case where U is unbounded, and then an $O(n)$-time algorithm for general values of L and U.

Other related works include algorithms for the problem of computing a segment $\langle a_i, \ldots, a_j \rangle$ with a maximum sum $a_i + \cdots + a_j$ as opposed to a maximum density. Bentley [2] gave an $O(n)$-time algorithm for the case where $L = 0$ and $U = n$. Within the same linear time complexity, Huang [21] solved the case with arbitrary L yet unbounded U. More recently, Lin et al. [26] solved the case with arbitrary L and U.

2. Notation and preliminaries

We consider A to be a sequence of n objects, where each object is represented by a pair of two real numbers (a_i, w_i) for $i = 1, \ldots, n$ and $w_i > 0$. If $w_i = 1$ for all i, we denote this as the *uniform* model. For $i \leqslant j$, we let $A(i, j)$ denote that segment of A which begins at index i and ends with index j. We let $w(i, j)$ denote the *width* of $A(i, j)$, defined as $w(i, j) = \sum_{i \leqslant k \leqslant j} w_k$. We let $\mu(i, j)$ denote the *density* of $A(i, j)$, defined as

$$\mu(i, j) = \left(\sum_{i \leqslant k \leqslant j} a_k \right) \Bigg/ w(i, j).$$

We note that the prefix sums of the input sequence can be precomputed in $O(n)$ time. With these, the values of $w(i, j)$ and $\mu(i, j)$ can be computed in $O(1)$ time for any (i, j) using the following formulas:

$$w(i, j) = \sum_{1 \leqslant k \leqslant j} w_k - \sum_{1 \leqslant k \leqslant i-1} w_k,$$

$$\mu(i, j) = \left(\sum_{1 \leqslant k \leqslant j} a_k - \sum_{1 \leqslant k \leqslant i-1} a_k \right) \Bigg/ w(i, j).$$

The maximum-density segment problem is to find a segment $A(i, j)$ of maximum density, subject to $L \leqslant w(i, j) \leqslant U$. Without loss of generality, we assume that $w_i \leqslant U$ for all i, as items with larger width could not be used in a solution.

For a given index i, we let L_i denote the minimum index such that $w(i, L_i) \geqslant L$ if such an index exists, and we let U_i denote the maximum index such that $w(i, U_i) \leqslant U$. A direct consequence of these definitions is that segment $A(i, j)$ has width satisfying $L \leqslant w(i, j) \leqslant U$ if and only if L_i is well defined and $L_i \leqslant j \leqslant U_i$. Recalling that $w_i > 0$ for all i, another consequence of these definitions is the following lemma.

```
j ← n
for i ← n downto 1 do
    while (w(i, j) > U) do
        j ← j − 1
    end while
    U_i ← j
end for
```

Fig. 1. Algorithm for precomputing U_i for all i.

Lemma 1. *For indices $i < j$, $U_i \leqslant U_j$, and if both L_i and L_j are well defined then $L_i \leqslant L_j$.*

Proof. Since $A(j, U_i)$ is contained in $A(i, U_i)$, the fact that $w(i, U_i) \leqslant U$ implies that $w(j, U_i) \leqslant U$. Thus U_j must be at least U_i, by definition. Similarly, if both L_i and L_j are well defined, then $A(j, L_j)$ is contained in $A(i, L_j)$. The fact that $w(j, L_j) \geqslant L$ implies that $w(i, L_j) \geqslant L$ and so L_i must be at most L_j by definition. □

This monotonicity allows for the full set of L_i and U_i values to be precomputed in $O(n)$ time by a simple sweep-line technique. The precomputation of the U_i values is shown in Fig. 1; a similar technique can be used for computing L_i values. It is not difficult to verify the correctness and efficiency of these computations.

3. Right-skew segments

For the uniform model, Lin et al. [26] defined segment $A(i, k)$ to be *right-skew* if and only if $\mu(i, j) \leqslant \mu(j + 1, k)$ for all $i \leqslant j < k$. A partition of a sequence A into segments $A_1 A_2 \ldots A_m$ was termed a *decreasingly right-skew partition* if it is the case that each A_i is right-skew, and that $\mu(A_x) > \mu(A_y)$ for any $x < y$. Based on these definitions, they proved the following lemma.

Lemma 2. *Every sequence A has a unique decreasingly right-skew partition.*

We denote this unique partition as $DRSP(A)$. Within the proof of the above lemma, the authors implicitly demonstrated the following fact.

Lemma 3. *If segment $A(x, y)$ is not right-skew, then $DRSP(A(x, y))$ is precisely equal to the union of $A(x, k)$ and $DRSP(A(k + 1, y))$ where $A(x, k)$ is the longest possible right-skew segment beginning with index x.*

Because of this structural property, the decreasingly right-skew partitions of all suffixes of $A(1, n)$ can be simultaneously represented by keeping a *right-skew pointer*, $p[i]$, for each $1 \leqslant i \leqslant n$. The pointer is such that $A(i, p[i])$ is the first right-skew segment of $DRSP(A(i, n))$. They implicitly used dynamic programming to construct all such right-skew pointers in $O(n)$ time.

In order to find a maximum-density segment of width at least L, they searched independently for the "good partner" of each index i. The good partner of i is the index i' that maximizes $\mu(i, i')$ while satisfying $w(i, i') \geqslant L$. In order to find each good partner, they made use of versions of the following three lemmas.

Lemma 4 (*Atomic*). *Let B, C and D be sequences with $\mu(B) \leqslant \mu(C) \leqslant \mu(D)$. Then $\mu(BC) \leqslant \mu(BCD)$.*

Lemma 5 (*Bitonic*). *Let B be a sequence and let $DRSP(C) = C_1C_2 \cdots C_m$ for sequence C which immediately follows B. Let k be the greatest index $i \in [0, m]$ that maximizes $\mu(BC_1C_2 \cdots C_i)$. Then $\mu(BC_1C_2 \cdots C_i) > \mu(BC_1C_2 \cdots C_{i+1})$ if and only if $i \geqslant k$.*

Lemma 6. *Given a sequence B, let C denote the shortest segment of B realizing the maximum density for those segments of width at least L. Then the width of C is at most $2L - 1$.*

Without any upper bound on the desired segment length, the consequence of these lemmas is an $O(\log L)$-time algorithm for finding a good partner for arbitrary index *i*. Since only segments of width *L* or greater are of interest, the segment $A(i, L_i)$ must be included. If considering the possible inclusion of further elements, Lemma 4 assures that if part of a right-skew segment increases the density, including that entire segment is just as helpful (in the application of that lemma, *C* represents part of a right-skew segment *CD*). Therefore, the good partner for *i* must be L_i or else the right endpoint of one of the right-skew segments from $DRSP(A(L_i + 1, n))$. Lemma 5 shows that the inclusion of each successive right-skew segment leads to a bitonic sequence of densities, thus binary search can be used to locate the good partner. Finally, Lemma 6 assures that at most *L* right-skew segments need be considered for inclusion, and thus the binary search for a given *i* runs in $O(\log L)$ time. The result is an $O(n \log L)$-time algorithm for arbitrary *L*, with $U = n$.

Though presented in terms of the uniform model, the definition of a right-skew segment involves only the densities of segments and so it applies equally to our more general model. Lemmas 2–5 remain valid in the general model. A variant of Lemma 6 can be achieved with the additional restriction that $w_i \geqslant 1$ for all *i*, and thus their $O(n \log L)$-time algorithm applies subject to this additional restriction.

4. Improved algorithms

Our techniques are built upon the use of decreasingly right-skew partitions, as reviewed in Section 3. Our improvements are based upon the following observation. An exact good partner for an index *i* need not be found if it can be determined that such a partner would result in density no greater than that of a segment already considered. In particular, we make use of the following key lemma.

Lemma 7. *For a given j, assume $A(j, j')$ is a maximum-density segment of those starting with index j, having $L \leqslant w(j, j') \leqslant U$, and ending with index in a given range $[x, y]$. For a given $i < j$, assume $A(i, i')$ is a maximum-density segment of those starting with index i, having $L \leqslant w(i, i') \leqslant U$ and ending in range $[x, y]$. If $i' > j'$, then $\mu(j, j') \geqslant \mu(i, i')$.*

Proof. A typical such configuration is shown in Fig. 2. By assumption, both indices i' and j' lie within the range $[x, y]$. Since $L \leqslant w(j, j') < w(j, i') < w(i, i') \leqslant U$, the optimality of $A(j, j')$ guarantees that $\mu(j, j') \geqslant \mu(j, i')$. This implies that $\mu(j, j') \geqslant \mu(j, i') \geqslant \mu(j' + 1, i')$. Since $L \leqslant w(j, j') < w(i, j') < w(i, i') \leqslant U$, the optimality of $A(i, i')$ guarantees that $\mu(i, i') \geqslant \mu(i, j')$, which in turn implies

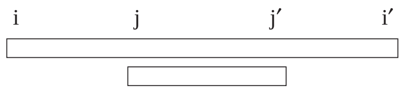


Fig. 2. Segments in proof of Lemma 7.

$\mu(j'+1, i') \geqslant \mu(i, i') \geqslant \mu(i, j')$. Combining these inequalities, $\mu(j, j') \geqslant \mu(j, i') \geqslant \mu(j'+1, i') \geqslant \mu(i, i')$, thus proving the claim that $\mu(j, j') \geqslant \mu(i, i')$. $\square$

Our high level approach is thus to find good partners for each left endpoint, considering those indices in decreasing order. However, rather than finding the true good partner for each i, our algorithm considers only matching indices which are less than or equal to all previously found good partners, in accordance with Lemma 7. With the use of sweep-line data structures, we can replace the $O(\log L)$-time binary searches used by Lin et al. [26] with sequential searches that run with an *amortized* time of $O(1)$.

4.1. Maximum-density segment with width at least L

We begin by considering the special case of finding a segment with the maximum possible density among those of width at least L, but not subject to any explicit upper bound. We first develop a sweep-line data structure which helps manage the search for good partners, then use such a data structure to implement an $O(n)$-time algorithm for this setting.

*4.1.1. The L-*Match *data structure*

Given a range $[x, y]$ specified upon initialization, the data structure developed in this section is designed to answer queries of the following type. For left index i, the goal is to return a matching right index i' such that $\mu(i, i')$ is maximized, subject to the constraints that $i' \in [x, y]$ and that $w(i, i') \geqslant L$. Yet, in order to achieve improved efficiency, the searches are limited in the following two ways:

(1) The structure can be used to find matches for many different left indices, however those indices must be queried in decreasing order.
(2) When asked to find the match for a left index, the structure only finds the true good partner in the case that the good partner has index less than or equal to all previously returned indices.

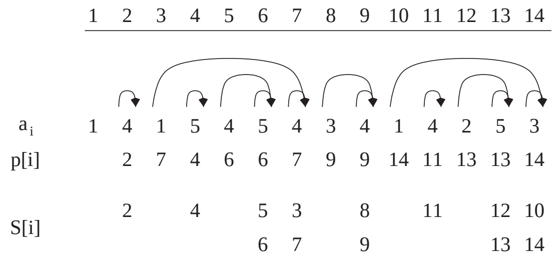
Our data structure augments the right-skew pointers for a given interval with additional information used to speed up searches for good partners. The structure contains the following state information, relative to given parameters $1 \leqslant x \leqslant y \leqslant n$:

- A (static) array, $p[k]$ for $x+1 \leqslant k \leqslant y$, where $A(k, p[k])$ is the *leftmost* segment of $DRSP(A(k, y))$.
- A sorted list, $S[k]$, for each $x+1 \leqslant k \leqslant y$, containing all indices j for which $p[j] = k$.
- Two indices ℓ and u (for "lower" and "upper"), whose values are non-increasing as the algorithm progresses.
- A variable, b (for "bridge"), which is maintained so that $A(b, p[b])$ is the segment of $DRSP(A(\ell, y))$ which contains index u.

The data structure is initialized with procedure `L-Match-Initialize`(x, y), given in Fig. 3. An example of an initialized structure is given in Fig. 4. Lines 1–8 of `L-Match-Initialize` set the

```
procedure L-Match-Initialize$(x, y)$        assumes $1 \le x \le y \le n$
    for $i \leftarrow y$ downto $x + 1$ do
        $S[i] \leftarrow \emptyset$
        $p[i] \leftarrow i$
        while $((p[i] < y)$ and $(\mu(i, p[i]) \le \mu(p[i] + 1, p[p[i] + 1])))$ do
            $p[i] \leftarrow p[p[i] + 1]$
        end while
        Insert $i$ at beginning of $S[p[i]]$
    end for
    $\ell \leftarrow y$; $u \leftarrow y$; $b \leftarrow y$
```

Fig. 3. The `L-Match-Initialize` operation, which sets the data structure's state information: $S[\,]$, $p[\,]$, ℓ, u, b.



1 2 3 4 5 6 7 8 9 10 11 12 13 14
a_i 1 4 1 5 4 5 4 3 4 1 4 2 5 3
p[i] 2 7 4 6 6 7 9 9 14 11 13 13 14
S[i] 2 4 5 3 8 11 12 10
6 7 9 13 14

Fig. 4. Example of data structure after `L-Match-Initialize`$(1, 14)$, when $w_i = 1$ for all i.

values $p[k]$ as was done in the algorithm of Lin et al. [26]. Therefore, we state the following lemma, proven in that paper.

Lemma 8. *After a call to* `L-Match-Initialize`(x, y), *for all* $x + 1 \leqslant k \leqslant y$, $p[k]$ *is set such that* $A(k, p[k])$ *is the leftmost segment of* $DRSP(A(k, y))$.

We also prove the following nesting property of decreasingly right-skew partitions.

Lemma 9. *Consider two segments* $A(x_1, y)$ *and* $A(x_2, y)$ *with a common right endpoint. Let* $A(k, k')$ *be a segment of* $DRSP(A(x_1, y))$ *and let* $A(m, m')$ *be a segment of* $DRSP(A(x_2, y))$. *It cannot be the case that* $k < m \leqslant k' < m'$.

Proof. We assume for contradiction that $k < m \leqslant k' < m'$, and consider the following three non-empty segments, $A(k, m - 1)$, $A(m, k')$ and $A(k' + 1, m')$. Since $A(k, k')$ is right-skew, it must be that $\mu(k, m - 1) \leqslant \mu(m, k')$. Since $A(m, m')$ is right-skew, it must be that $\mu(m, k') \leqslant \mu(k' + 1, m')$. In this case, it must be that the combined segment $A(k, m')$ is right-skew (this fact can be explicitly proven by application of Lin et al.'s Lemma 4 [26]).

Since $A(k, k')$ is a segment of $DRSP(A(x_1, y))$, a repeated application of Lemma 3 assures that $A(k, k')$ is the leftmost segment of $DRSP(A(k, y))$ and that $A(k, k')$ is the longest possible right-skew segment of those starting with index k. Yet the existence of the longer, right-skew segment $A(k, m')$ forms a contradiction. $\square$

```
    procedure L-Match-Find($i$)
1       while ($\ell > 1 + \max(x, L_i)$) do                   // decrease $\ell$
2           $\ell \leftarrow \ell - 1$
3           if ($p[\ell] \geq u$) then
4               $b \leftarrow \ell$
5           end if
6       end while
7       while ($u \geq \ell$) and ($\mu(i, b-1) > \mu(i, p[b])$) do     // bitonic search
8           $u \leftarrow b - 1$
9           if ($u \geq \ell$) then
10              $b \leftarrow$ minimum $j \in S[u]$ such that $j \geq \ell$,
11              removing all $k > b$ from $S[u]$
12          end if
13      end while
14      return $u$
```

Fig. 5. The `L-Match-Find`(i) operation. Recall that $S[\,]$, $p[\,]$, ℓ, u, b and x are maintained as state information for the L-Match data structure.

Corollary 10. *For indices k and m, it cannot be that $k < m \leqslant p[k] < p[m]$.*

Proof. A direct result of Lemmas 8–9. □

Lemma 11. *If b is the minimum value satisfying $\ell \leqslant b \leqslant u \leqslant p[b]$, then $A(b, p[b])$ is the segment of $DRSP(A(\ell, y))$ which contains index u.*

Proof. By Lemma 8, $A(b, p[b])$ is the leftmost segment of $DRSP(A(b, y))$, and as $b \leqslant u \leqslant p[b]$, $A(p, p[b])$ contains index u.

By repeated application of Lemma 3, $DRSP(A(\ell, y))$ equals $A(\ell, p[\ell])$, $A(p[\ell]+1, p[p[\ell]+1])$, and so on, until reaching right endpoint y. We claim that $A(b, p[b])$ must be part of that partition. If not, there must be some other $A(m, p[m])$ with $m < b \leqslant p[m]$. By Lemma 9, it must be that $p[m] \geqslant p[b]$, yet then $m < b \leqslant u \leqslant p[b] \leqslant p[m]$. Such an m violates the assumed minimality of b. □

The data structure's query routine, `L-Match-Find`, is introduced in Fig. 5.

Lemma 12. *Whenever line 7 of* `L-Match-Find()` *is evaluated, b is the minimum value satisfying $\ell \leqslant b \leqslant u \leqslant p[b]$, if such a value exists.*

Proof. We show this by induction over time. When initialized, $\ell = b = u = p[b] = y$, and thus b is the only satisfying value. The only time this invariant can be broken is when the value of ℓ or u changes. ℓ is changed only when decremented at line 2 of `L-Match-Find`. The only possible violation of the invariant would be if the new index ℓ satisfies $\ell \leqslant u \leqslant p[\ell]$. This is exactly the condition handled by lines 3–4.

Second, u is modified only at line 8 of `L-Match-Find`. Immediately before this line is executed, the invariant holds. At this point, we claim that $p[k] \leqslant b - 1$ for any values of k such that $\ell \leqslant k < b$. For $k < b$, Corollary 10 implies that either $p[k] < b$ or $p[k] \geqslant p[b]$. If it were the case that $p[k] \geqslant p[b] \geqslant u$ this would violate the minimality of b assumed at line 7. Therefore, it must be that $p[k] \leqslant b - 1$ for all $\ell \leqslant k \leqslant b - 1$. As u is reset to $b - 1$, the only possible values for the new bridge b are those indices j with

$p[j] = u$. The set $S[u]$ considered at line 10 of `L-Match-Find` ensures that b is the minimum such $j \geqslant \ell$. □

Lemma 13. *Assume* `L-Match-Find`(i) *is called with a value i less than that of all previous invocations and such that $L_i < y$. Let r be the most recently returned value from* `L-Match-Find`$()$ *or y if this is the first such call. Let $A(i, i')$ be the widest maximum-density segment of those starting with i, having width at least L, and ending in $[x, y]$. Then* `L-Match-Find`(i) *returns the value* $\min(i', r)$.

Proof. All segments which start with *i*, having width at least *L* and ending in $[x, y]$ must include interval $A(i, \max(x, L_i))$. The loop starting at line 1 ensures that variable $\ell = 1 + \max(x, L_i)$ upon the loop's exit. As discussed in Section 3, the optimal such i' must either be $\ell - 1$ or else among the right endpoints of $DRSP(A(\ell, y))$.

Since u is only set within `L-Match-Find`, it must be that $u = r$ upon entering the procedure. By Lemmas 11–12, $A(b, p[b])$ is the right-skew segment containing index u in $DRSP(A(\ell, y))$. If $\mu(i, b-1) \leq \mu(i, p[b])$, Lemma 5 assures that the good partner must have index at least $p[b] \geqslant u$. In this case, the while loop of line 7 is never entered, and the procedure returns $r = \min(i', r)$.

In any other case, a true good partner for *i* is less than or equal to *r*, and is found by the while loop of line 7, in accordance with Lemmas 4–5 and 11–12. □

Corollary 14. *If* `L-Match-Find`(i) *fails to return i', as defined in the statement of Lemma 13, it must be the case that for some $j > i$, a previous call to* `L-Match-Find`(j) *returns a j' such that* $\mu(j, j') \geqslant \mu(i, i')$.

Proof. By Lemma 13, the returned value is $\min(i', r)$. For the first call, $r = y$ and so the returned value must be $i' \leqslant y$. If $r < i'$, we consider the largest index $j > i$ for which the returned value, j', is strictly less than i'. When `L-Match-Find`(j) was invoked, the respective value of *u* must have been greater than or equal to i'. Therefore, $A(j, j')$ must truly be the maximum-density segment, of those starting with *j*, of width at least *L*, and ending in $[x, y]$. We can thus apply Lemma 7, with *U* unbounded, concluding that $\mu(j, j') \geqslant \mu(i, i')$. □

Lemma 15. *The L-Match data structure supports its operations with amortized running times of $O(y - x + 1)$ for* `L-Match-Initialize`(x, y)*, and $O(1)$ for* `L-Match-Find`(i).

Proof. With the exception of lines 2, 7 and 9, the initialization procedure is simply a restatement of the algorithm given by Lin et al. [26] for constructing the right-skew pointers. An $O(y - x + 1)$-time worst-case bound was proven by those authors.

In analyzing the cost of `L-Match-Find`, an $O(1)$ cost accounts for first evaluation of the loop condition at lines 1 and 7, as well as the return statement at line 14. The additional costs incurred during iterations of either of the while loops will be amortized against the $O(y - x + 1)$ cost of the initialization process. First, we claim that the loop of lines 1–6 of `L-Match-Find` iterates at most $y - x + 1$ times. This is so because variable ℓ is initialized to value *y* at line 9 of `L-Match-Initialize`, remains at least $x + 1$ due to the condition at line 1, and modified only when decremented at line 2 during each iteration. Second, we claim that the loop of lines 7–13 iterates at most $y - x + 1$ times. This is so because

```
   procedure MaximumDensitySegmentL($A, L$)
1      calculate values, $L_i$, as discussed in Section 2
2      call L-Match-Initialize($1, n$) to create data structure
3      $i_0 \leftarrow$ maximum index such that $L_{i_0}$ is well-defined
4      for $i \leftarrow i_0$ downto 1 do
5          if ($L_i = n$) then              // only one feasible right index
6              $g[i] \leftarrow n$
7          else
8              $g[i] \leftarrow$ L-Match-Find($i$)
9          end if
10     end for
11     return $(k, g[k])$ which maximizes $\mu(k, g[k])$ for $1 \le k \le i_0$
```

Fig. 6. Algorithm for finding maximum-density segment with width at least L.

variable u is initialized to value y at line 9 of `L-Match-Initialize`, modified only at line 8. By Lemma 12, $x < \ell \leqslant b \leqslant u \leqslant p[b]$, and so this line results in a strict decrease in the value of u yet u remains at least x. The only operations in either loop which cannot be bounded by $O(1)$ in the worst case are those of lines 10–11. Because $S[u]$ is sorted by construction, the cost of these operations is proportional to one plus the number of removals. As each k is inserted into list $S[u]$ for a distinct value of u, the overall number of removals is bounded by $O(y - x + 1)$. □

4.1.2. An $O(n)$-time algorithm

In Fig. 6, we present a linear-time algorithm for the maximum-density segment problem subject only to a lower bound, L, on the segment width. The algorithm makes use of a single L-Match data structure, for the range $[1, n]$.

Theorem 16. *Given a sequence A, the algorithm* `MaximumDensitySegmentL` *identifies a maximum-density segment of those with width at least L.*

Proof. First, we note that $w(k, g[k]) \geqslant L$ for all $k \leqslant i_0$, as a result of Lemma 13, and thus the segment returned by line 11 is indeed of adequate width. To complete the proof, let i denote the greatest index which begins such a maximum-density segment and (i, i') the widest such optimal segment beginning at i. We show that $g[i] = i'$. As L_i must be well defined, we consider the pass of the loop starting at line 4 for such an i. If $L_i = n$, then this is the only feasible right index for i and $g[i] = i' = n$. Alternatively, if `L-Match-Find` is invoked, Corollary 14 assures that if $g[i] \neq i'$, then we previously considered a segment $A(j, g[j])$ with $j > i$ such that $\mu(j, g[j]) \geqslant \mu(i, i')$. This contradicts the assumption that i is the greatest index beginning such an optimal segment of width at least L. Therefore, $g[i] = i'$ and such a segment is returned by the procedure. □

Theorem 17. *The algorithm* `MaximumDensitySegmentL` *runs in $O(n)$ time, given a sequence A of length n.*

Proof. Line 1 runs in $O(n)$ time as per discussion in Section 2, line 2 in $O(n)$ as per Lemma 15, and there are at most n calls to `L-Match-Find` at line 8, running in combined $O(n)$ time as per Lemma 15. □

```
    procedure U-Match-Initialize$(x, y)$          assumes $1 \le x \le y \le n$
1       for $i \leftarrow x+1$ to $y$ do
2           $q[i] \leftarrow i$
3           while $((q[i] > x)$ and $(\mu(q[q[i]-1], q[i]-1) \le \mu(q[i], i)))$ do
4               $q[i] \leftarrow q[q[i]-1]$
5           end while
6       end for
7       $u \leftarrow y$
```

Fig. 7. The `U-Match-Initialize` operation, which sets the data structure's state information: $q[\,]$, u.

4.2. Maximum-density segment with width at least L and at most U

In the previous section, we considered the problem with the width subject to a lower bound, L, but no explicit upper bound. In this section, we consider the addition of an explicit upper bound, U, on the width, presenting an $O(n)$-time algorithm for this setting.

This algorithm is based upon a generalization of the techniques in Section 4.1, however those techniques cannot be directly applied. The L-Match data structure enforced a lower bound on the width of considered segments, but no upper bound. At first glance, the sweeping of variable u in that structure may appear similar to placing an explicit upper bound on the width of the segments considered. The bridge segment $A(b, p[b])$ is used in performing the bitonic search, however that bridge is a segment of $DRSP(A(L_i+1, n))$. If the right endpoint of the bridge is strictly greater than U_i, the bridge cannot be considered atomically. To properly apply Lemmas 4 and 5, we must consider segments of $DRSP(A(L_i+1, U_i))$ as opposed to $DRSP(A(L_i+1, n))$.

Unable to develop a data structure which simultaneously enforces both an upper and lower bound on the segment width, our solution for the general setting is two-fold. We develop a second data structure, analogous though not strictly symmetrical to the L-Match, which locates good partners subject to an upper bound on the width yet no explicit lower bound. Then, rather than build each data structure over the entire range, $[1, n]$, we break the range into many "blocks" and maintain both types of data structures independently for each block. The blocks are designed so that a good partner for a given index need only be identified from two adjacent blocks.

4.2.1. The U-Match data structure

We develop another sweep-line data structure, U-Match, used to locate segments beginning with i, ending in $[x, y]$ and subject to an *upper* bound on the resulting segment width (but with no explicit lower bound). For a given i, the decomposition of interest is $DRSP(A(x+1, U_i))$. As $x+1$ is fixed yet U_i decreases with i, we choose to represent the decreasingly right-skew partitions for all *prefixes* of $A(x+1, y)$, rather than all *suffixes* as with the L-Match structure. We assign values $q[k]$ for $x+1 \leqslant k \leqslant y$ such that $A(q[k], k)$ is the *rightmost* segment of $DRSP(A(x+1, k))$.

There exists a clear symmetry between the L-Match and U-Match structures, though the symmetry is not perfect as the concept of right-skew segments, used in both structures, is oriented. In fact the U-Match structure is considerably simpler, with the only state information being the array $q[\,]$ and a non-increasing index u. The initialization routine for this new structure is presented in Fig. 7. An example of an initialized structure is given in Fig. 8. The redesign of the initialization routine relies on a simple duality when compared with the corresponding routine of Section 4.1.1. One can easily verify that an execution of this

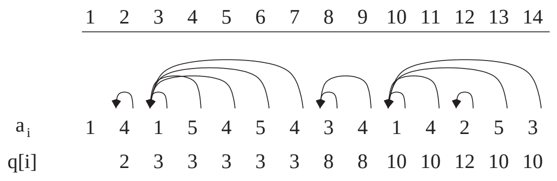

1 2 3 4 5 6 7 8 9 10 11 12 13 14
a_i 1 4 1 5 4 5 4 3 4 1 4 2 5 3
q[i] 2 3 3 3 3 3 8 8 10 10 12 10 10

Fig. 8. Example of data structure after `U-Match-Initialize`(1, 14), with $w_i = 1$ for all i.

```
    procedure U-Match-Find($i$)
1       while ($u > U_i$) do                                  // decrease $u$
2           $u \leftarrow u - 1$
3       end while
4       while ($u > x$) and ($\mu(i, q[u] - 1) > \mu(i, u)$) do      // bitonic search
5           $u \leftarrow q[u] - 1$
6       end while
7       return $u$
```

Fig. 9. `U-Match-Find`(i) operation. Recall that $q[\,]$, u and x are maintained as state information for the U-Match data structure.

routine on a segment $A(x, y)$ sets the values of array $q[\,]$ precisely as the original version would set the values of array $p[\,]$ if run on a reversed and negated copy of $A(x, y)$. Based on this relationship, we claim the following dual of Lemma 8 without further proof.

Lemma 18. *After a call to* `U-Match-Initialize`(x, y), *the segment* $A(q[k], k)$ *is the rightmost segment of* $DRSP(A(x + 1, k))$, *for all* $x + 1 \leqslant k \leqslant y$.

The data structure's query routine, `U-Match-Find`, is introduced in Fig. 9.

Lemma 19. *Assume* `U-Match-Find`(i) *is called with a value i less than that of all previous invocations and such that* $x \leqslant U_i \leqslant y$. *Let r be the most recently returned value from* `U-Match-Find`() *or y if this is the first such call. Let* $A(i, i_r)$ *be the widest maximum-density segment of those starting with i, having width at most U, and ending with* $i_r \in [x, r]$. *Then* `U-Match-Find`(i) *returns the value* i_r.

Proof. Combining the constraints that $w(i, i_r) \leqslant U$ and that $i_r \in [x, r]$, it must be that $i_r \leqslant \min(U_i, r)$. When entering the procedure, the variable u has value r. The loop starting at line 1 ensures that variable $u = \min(U_i, r)$ upon the loop's exit. The discussion in Section 3 assures that the optimal $i_r \in [x, u]$ must either be x or else among the right endpoints of $DRSP(A(x + 1, u))$. Based on Lemma 18, $A(q[u], u)$ is the rightmost segment of $DRSP(A(x + 1, u))$ and so the loop condition at line 4 of `U-Match-Find` is a direct application of Lemma 5. □

Corollary 20. *Let* $A(i, i')$ *be the widest maximum-density segment of those starting with i, having width at most U, and ending with* $i \in [x, y]$. *In the statement of Lemma 19,* $A(i, i_r)$ *was defined similarly, however with* $i_r \in [x, r]$ *rather than* $[x, y]$. *If* $i_r \neq i'$, *it must be the case that for some* $j > i$, *a previous call to* `U-Match-Find`(j) *returns a* j' *such that* $\mu(j, j') \geqslant \mu(i, i')$.

Proof. Since i' is the optimal such match in the range $[x, y]$, it would so be the optimal such match over the range $[x, r]$ so long as $r \geqslant i'$. If $r < i'$, we consider the largest index $j > i$ for which the returned value, j', is strictly less than i'. When `U-Match-Find`(j) was invoked, the previously returned value was at least i'. So the segment $A(j, j')$ must be the maximum density segment, starting with j, having width at most U, and ending in the range $[x, i']$. We can thus apply Lemma 7 constrained to the range $[x, i']$, concluding that $\mu(j, j') \geqslant \mu(i, i')$. □

Lemma 21. *The U-Match data structure supports its operations with amortized running times of $O(y - x + 1)$ for* `U-Match-Initialize`(x, y), *and $O(1)$ for* `U-Match-Find`(i), *so long as $U_i \geqslant x$ for all i.*

Proof. The initialization procedure has an $O(y - x + 1)$-time worst-case bound, as was the case for the similar routine in Section 4.1.1.

To account for the cost of `U-Match-Find`, we note that u is initialized to value y at line 7 of `U-Match-Initialize`. It is only modified by lines 2 and 5 of the routine, and we claim that both lines strictly decrease the value. This is obvious for line 2; for line 5 it follows since $q[u] \leqslant u$ in accordance with Lemma 18. We also claim that u is never set less than x. Within the loop of lines 1–3, this is due to the assumption that $U_i \geqslant x$. For the loop of lines 4–7, it is true because $q[u] \geqslant x + 1$ in accordance with Lemma 18. Therefore, these loops execute at most $O(y - x + 1)$ times combined and this cost can be amortized against the initialization cost. An $O(1)$ amortized cost per call can account for checking the initial test condition before entering either loop. □

4.2.2. An $O(n)$-time algorithm

In this section, we present a linear-time algorithm for the maximum-density segment problem subject to both a lower bound L and an upper bound U on the segment width, with $L < U$. Our strategy is as follows. We preprocess the original sequence by breaking it into smaller, disjoint blocks, maintaining an L-Match and U-Match data structure for each such block. Let i_0 be the maximum index such that L_{i_0} is defined. We partition the range $[L_1, U_{i_0}]$ into a collection of disjoint blocks based upon the following recursive definition. We let block $[x_1, y_1] = [L_1, U_1]$. Then, so long as $y_{c-1} \neq U_{i_0}$, we add a block $[x_c, y_c]$ to the collection, where $x_c = y_{c-1} + 1$ and $y_c = U_i$, where i is the minimum index such that $L_i \geqslant x_c$ or i_0 if no such index exists. These blocks can be constructed in $O(n)$ time, by the `ConstructBlocks` algorithm, shown in Fig. 10. The boundaries are defined precisely to guarantee the following lemma.

Lemma 22. *For a given range, $[L_i, U_i]$, let $[x, y]$ be the block containing L_i. Then it is either the case that $U_i = y$ or that U_i is contained in the adjacent block, $[x', y']$.*

Proof. Let h be the minimum index such that L_h lies in block $[x, y]$. As $h \leqslant i$, Lemma 1 implies that $U_i \geqslant U_h = y$. If $U_i > y$ then there must exist an adjacent block $[x', y']$, since the blocks continue until reaching $U_m \geqslant U_i$.

We show that $U_i \leqslant y'$. If $y' = U_m$, then this is trivially true. Otherwise, there exists a j, which is the minimum index such that L_j lies in block $[x', y']$. A typical such configuration is shown in Fig. 11. Since $i < j$, Lemma 1 assures that $U_i \leqslant U_j = y'$, thereby proving the lemma. □

```
procedure ConstructBlocks($A, L, U$)
    calculate values $L_i$ and $U_i$ as discussed in Section 2
    $i \leftarrow 1$; $i_0 \leftarrow$ maximum index such that $L_{i_0}$ is well-defined
    $x \leftarrow L_1$, $y \leftarrow U_1$
    Add block $[x, y]$ to the collection
    while ($y < U_{i_0}$) do
        $x \leftarrow y + 1$
        while ($i < i_0$) and ($L_i < x$) do    // find min $L_i \geq x$ if exists
            $i \leftarrow i + 1$
        end while
        $y \leftarrow U_i$
        Add block $[x, y]$ to the collection
    end while
```

Fig. 10. Algorithm for constructing the collection of blocks.

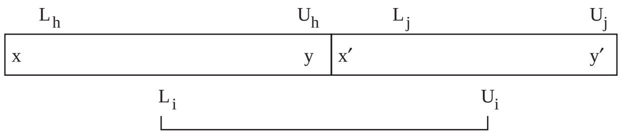


Fig. 11. An example of a typical search range, $[L_i, U_i]$, that lies across two adjacent blocks, $[x, y]$ and $[x', y']$.

For a given left index i, a valid good partner must lie in the range $[L_i, U_i]$. Assuming that block $[x, y]$ contains L_i, we search for a potential partner in the range $[L_i, y]$ by querying the L-Match structure for that block, and, if $U_i > y$, search for a potential partner in the range $[y + 1, U_i]$ by querying the U-Match structure for the adjacent block. Though we are not assured of finding the true good partner for each i, we again find the global optimum, in accordance with Lemma 7. Our complete algorithm is given in Fig. 12.

Theorem 23. *Given a sequence A and parameters $L < U$, the* `MaximumDensitySegmentLU` *algorithm identifies a maximum-density segment of those with width at least L and at most U.*

Proof. For a given $k \leqslant i_0$, assume L_k lies in block $[x, y]$. Lemma 22 assures us that either $U_k = y$ or U_k lies within the adjacent block. Therefore, a good partner must lie in the range $[L_k, y]$ or possibly $[y + 1, U_k]$.

We first note that $L \leqslant w(k, g[k]) \leqslant U$ for any $k \leqslant i_0$ and thus that the segment returned by the algorithm is of adequate width. For the value $g[k]$ determined at line 13, the lower bound on the width is due to Lemma 13 and the upper bound since $y \leqslant U_k$. Similarly, if the adjacent block is queries, the lower bound is due to the fact that $L_k < y + 1$ and the upper bound due to Lemma 19.

To complete the proof, let i denote the greatest index which begins such a maximum-density segment and (i, i') the widest such optimal segment beginning at i. We show that $g[i] = i'$. As L_i must be well defined, we consider the pass of the loop starting at line 9 for such an i. As discussed earlier, i' must lie in the range $[L_i, y]$ or $[y + 1, U_i]$ where $[x, y]$ is the block containing L_i. If i' lies within range $[L_i, y]$, the invocation of `L-Match-Find` at line 13 will return i', or else, by Corollary 14, we would have already considered a segment $A(j, g[j])$ with $j > i$ such that $\mu(j, g[j]) \geqslant \mu(i, i')$, contradicting the assumption

```
procedure MaximumDensitySegmentLU($A, L, U$)
    calculate values $L_i$ and $U_i$ as discussed in Section 2
    call ConstructBlocks($A, L, U$)
    foreach block $[x, y]$ in collection
        call L-Match-Initialize($x, y$)
        call U-Match-Initialize($x, y$)
    end foreach
    $i_0 \leftarrow$ maximum index such that $L_{i_0}$ is well-defined
    $[x, y] \leftarrow$ rightmost block of collection
    for $i \leftarrow i_0$ downto $1$ do
        if $L_i \notin [x, y]$ do
            $[x, y] \leftarrow$ preceeding block in the collection
        end if
        $g[i] \leftarrow$ L-Match-Find($i$) invoked on block $[x, y]$
        if $U_i > y$ do
            $alt \leftarrow$ U-Match-Find($i$) invoked on *succeeding* block $[x', y']$
            if $(\mu(i, alt) \geq \mu(i, g[i]))$ then
                $g[i] \leftarrow alt$
            end if
        end if
    end for
    return $(k, g[k])$ which maximizes $\mu(k, g[k])$ for $1 \leq k \leq i_0$
```

Fig. 12. Algorithm for finding maximum-density segment with width at least L and at most U.

that i is the greatest index beginning such an optimal segment of width at least L. Similarly, if i' lies in a range $[y+1, U_i]$, the invocation of `U-Match-Find` at line 15 will return i', in similar accordance with Corollary 20. Therefore $g[i] = i'$ and such a segment is returned by the procedure. □

Theorem 24. *The* `MaximumDensitySegmentLU` *algorithm runs in $O(n)$ time, given a sequence A of length n.*

Proof. Because the blocks are disjoint, the overall time in initializing and querying the L-Match and U-Match data structures is $O(n)$, as per Lemmas 15 and 21. □

Note: The original extended abstract of this work [15], provides the $O(n)$-time algorithm for unbounded U, and an $O(n + n\log(U - L + 1))$-time algorithm for general U, in both the uniform and weighted cases. The $O(n)$-time algorithm of Section 4.2.2 was originally presented for the *uniform* model, in an earlier submission of the current article. In that case, the sequence is partitioned precisely into blocks of cardinality $(U - L + 1)$. The application of this technique to the non-uniform case, namely in relying on blocks of non-uniform cardinality as per Fig. 10, was first suggested to the authors by Greenberg [16].

Independently, Chung and Lu [6,7] present an $O(n)$-time algorithm for general U, using techniques which do not rely on the right-skew decomposition. Kim [25] presents an algorithm based on a geometric interpretation of the problem, claiming an $O(n)$ running time. Unfortunately, the analysis of the algorithm appears fatally flawed (see footnote 1 of [7] for further discussion).

Acknowledgements

We thank Yaw-Ling Lin and Ron Greenberg for helpful discussions. We also appreciate the helpful comments of the anonymous referees.

References

[1] N.N. Alexandrov, V.V. Solovyev, Statistical significance of ungapped sequence alignments, in: R.B. Altman, A.K. Dunker, L. Hunter, T.E. Klein (Eds.), Proceedings of the Pacific Symposium on Biocomputing, World Scientific, Maui, Hawaii, 1998, pp. 463–472.

[2] J.L. Bentley, Programming Pearls, Addison-Wesley, Reading, MA, 1986.

[3] G. Bernardi, Isochores and the evolutionary genomics of vertebrates, Gene 241 (1) (2000) 3–17.

[4] G. Bernardi, G. Bernardi, Compositional constraints and genome evolution, J. Mol. Evol. 24 (1–2) (1986) 1–11.

[5] B. Charlesworth, Genetic recombination: patterns in the genome, Curr. Biol. 4 (2) (1994) 182–184.

[6] K.-m. Chung, H.-I. Lu, An optimal algorithm for the maximum-density segment problem, in: G.D. Battista, U. Zwick (Eds.), Proceedings of the 11th Annual European Symposium on Algorithms, Lecture Notes in Computer Science, vol. 2832, Springer, Budapest, Hungary, 2003, pp. 136–147.

[7] K.-m. Chung, H.-I. Lu, An optimal algorithm for the maximum-density segment problem, SIAM J. Comput., to appear.

[8] L. Duret, D. Mouchiroud, C. Gautier, Statistical analysis of vertebrate sequences reveals that long genes are scarce in GC-rich isochores, J. Mol. Evol. 40 (3) (1995) 308–317.

[9] 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 Res. 20 (7) (1992) 1497–1501.

[10] A. Eyre-Walker, Recombination and mammalian genome evolution, Proc. Roy. Soc.: Biol. Sci. 252 (1335) (1993) 237–243.

[11] C.A. Fields, C.A. Soderlund, gm: a practical tool for automating DNA sequence analysis, Comput. Appl. Biosci. 6 (3) (1990) 263–270.

[12] J. Filipski, Correlation between molecular clock ticking codon usage fidelity of DNA repair, chromosome banding and chromatin compactness in germline cells, FEBS Lett. 217 (2) (1987) 184–186.

[13] M.P. Francino, H. Ochman, Isochores result from mutation not selection, Nature 400 (6739) (1999) 30–31.

[14] S.M. Fullerton, A.B. Carvalho, A.G. Clark, Local rates of recombination are positively correlated with GC content in the human genome, Mol. Biol. Evol. 18 (6) (2001) 1139–1142.

[15] M.H. Goldwasser, M.-Y. Kao, H.-I. Lu, Fast algorithms for finding maximum-density segments of a sequence with applications to bioinformatics, in: R. Guigó, D. Gusfield (Eds.), Proceedings of the Second Annual Workshop on Algorithms in Bioinformatics, Lecture Notes in Computer Science, vol. 2452, Springer, Rome, Italy, 2002, pp. 157–171.

[16] R.I. Greenberg, Fast and space-efficient location of heavy or dense segments in run-length encoded sequences (extended abstract), in: T. Warnow, B. Zhu (Eds.), Proceedings of the Ninth Annual International Conference Computing and Combinatorics, Lecture Notes in Computer Science, vol. 2697, Springer, Big Sky, MT, 2003, pp. 528–536.

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

[18] R.C. Hardison, D. Drane, C. Vandenbergh, J.-F.F. Cheng, J. Mansverger, J. Taddie, S. Schwartz, X. Huang, 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, J. Mol. Biol. 222 (2) (1991) 233–249.

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

[20] G.P. Holmquist, Chromosome bands, their chromatin flavors, and their functional features, Amer. J. Hum. Genet. 51 (1) (1992) 17–37.

[21] X. Huang, An algorithm for identifying regions of a DNA sequence that satisfy a content requirement, Comput. Appl. Biosci. 10 (3) (1994) 219–225.

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

[23] R.B. Inman, A denaturation map of the 1 phage DNA molecule determined by electron microscopy, J. Mol. Biol. 18 (1966) 464–476.

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

[25] S.K. Kim, Linear-time algorithm for finding a maximum-density segment of a sequence, Inform. Process. Lett. 86 (6) (2003) 339–342.

[26] Y.-L. Lin, T. Jiang, K.-M. Chao, Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis, J. Comput. System Sci. 65 (3) (2002) 570–586.

[27] G. Macaya, J.-P. Thiery, G. Bernardi, An approach to the organization of eukaryotic genomes at a macromolecular level, J. Mol. Biol. 108 (1976) 237–254.

[28] C.S. Madsen, C.P. Regan, G.K. Owens, Interaction 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, J. Biol. Chem. 272 (47) (1997) 29842–29851.

[29] S.-i. Murata, P. Herman, J.R. Lakowicz, Texture analysis of fluorescence lifetime images of AT- and GC-rich regions in nuclei, J. Hystochem. Cytochem. 49 (11) (2001) 1443–1452.

[30] A. Nekrutenko, W.-H. Li, Assessment of compositional heterogeneity within and between eukaryotic genomes, Genome Res. 10 (12) (2000) 1986–1995.

[31] P. Rice, I. Longden, A. Bleasby, EMBOSS: The European molecular biology open software suite, Trends Genet. 16 (6) (2000) 276–277.

[32] L. Scotto, 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, Mol. Cell. Biol. 13 (6) (1993) 3588–3597.

[33] P.H. Sellers, Pattern recognition in genetic sequences by mismatch density, Bull. Math. Biol. 46 (4) (1984) 501–514.

[34] P.M. Sharp, M. Averof, A.T. Lloyd, G. Matassi, J.F. Peden, DNA sequence evolution: the sounds of silence, Philos. Trans. Roy. Soc. London Ser. B: Biol. Sci. 349 (1329) (1995) 241–247.

[35] P. Soriano, M. Meunier-Rotival, G. Bernardi, The distribution of interspersed repeats is nonuniform and conserved in the mouse and human genomes, Proc. Natl. Acad. Sci. (USA) 80 (7) (1983) 1816–1820.

[36] N. Stojanovic, L. Florea, C. Riemer, D. Gumucio, J. Slightom, M. Goodman, W. Miller, R. Hardison, Comparison of five methods for finding conserved sequences in multiple alignments of gene regulatory regions, Nucleic Acids Res. 27 (19) (1999) 3899–3910.

[37] N. Sueoka, Directional mutation pressure and neutral molecular evolution, Proc. Natl. Acad. Sci. (USA) 85 (8) (1988) 2653–2657.

[38] Z. Wang, E. Lazarov, M. O'Donnel, 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, J. Biol. Chem. 277 (6) (2002) 4446–4454.

[39] K.H. Wolfe, P.M. Sharp, W.-H. Li, Mutation rates differ among regions of the mammalian genome, Nature 337 (6204) (1989) 283–285.

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

[41] S. Zoubak, O. Clay, G. Bernardi, The gene distribution of the human genome, Gene 174 (1) (1996) 95–102.