Constrained Heaviest Segments in a Number Sequence and Their Applications in Biomolecular Sequence Analysis (Working Draft)

Kun-Mao Chao[1,2]

[1]Department of Computer Science and Information Engineering
[2]Graduate Institute of Networking and Multimedia
National Taiwan University, Taipei, Taiwan 106
Email: kmchao@csie.ntu.edu.tw

October 5, 2005

Abstract

Keywords: algorithm, maximum-average segment, maximum-sum segment, sequence analysis

1 Introduction

With the rapid expansion in genomic data, the age of large-scale biomolecular sequence analysis has arrived. An important line of research in sequence analysis is to locate biologically meaningful segments, *e.g. conserved* segments (Stojonovic et al., 1999) and *GC-rich* regions (Bird, 1987; Gardiner-Garden and Frommer 1987; Huang 1994), non-coding RNA genes (Csűrös, 2004), and transmembrane segments (Fariselli et al., 2003).

A common approach is to assign a value to each residue, and then look for consectutive subsequences (substring) with high sum or average. In order to locate these interesting segments, many combinatorial and probabilistic techniques have been proposed. Perhaps the most popular ones are window-based. That is, a window of a fixed length is moved down the sequence/alignment and the content statistics are calculated at each position that the window is moved to (Nekrutenko and Li, 2000). Since an optimal region could span several windows, the window-based approach might fail in finding the exact locations of some interesting regions.

This chapter surveys recent developments in locating constrained heaviest segments in a number sequence. We start by introducing two basic algorithms for solving the maximum-sum substring problem and the maximum-average substring problem. Then we compile a list of recent application in this category. Finally, we discuss some possible extensions.

2 Two Basic Algorithms

2.1 The maximum-sum substring problem

Given a sequence of real numbers $A = \langle a_1, a_2, \ldots, a_n \rangle$, the maximum-sum substring problem is to find a consecutive subsequence (*i.e.*, a substring) in A with the maximum sum. For each position i, we can compute the maximum-sum substring ending at that position in $O(i)$ time. Therefore, a naive algorithm runs in $\sum_{i=1}^{n} O(i) = O(n^2)$ time.

Now let us describe a more efficient dynamic-programming algorithm for this problem (Bentley, 1986). Define $S(i)$ to be the maximum sum of substrings ending at position i of A. The value $S(i)$ can be computed by the following recurrence:

$$S(i) = \begin{cases} a_i + \max\{S(i-1), 0\} & \text{if } i > 1, \\ a_1 & \text{if } i = 1. \end{cases}$$

If $S(i-1) < 0$, concatenating a_i with its previous elements will give less sum than a_i itself. In this case, the maximum-sum substring ends at position i is a_i itself.

By a tabular computation, each $S(i)$ can be computed in constant time from $i = 1$ to $i = n$, therefore in total $O(n)$ time. During the computation, we also need to record the largest entry computed so far in order to report where

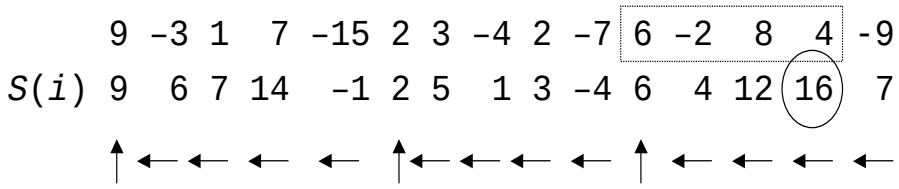

Figure 1: $A = \langle$ 9, -3, 1, 7, -15, 2, 3, -4, 2, 7, 6, -2, 8, 4, -9 $\rangle$. The maximum-sum substring of A is $\langle$ 6, -2, 8, 4 $\rangle$, whose sum is 16.

the maximum-sum substring ends. We also record the traceback information for each position i so that we can trace back from the end position of the maximum-sum substring to its start position. If $S(i-1) > 0$, we need to concatenate with previous elements for a larger sum, therefore the traceback symbol for position i is "$\leftarrow$." Otherwise, "$\uparrow$" is recorded. The traceback information can be used to quickly construct the maximum-sum substring by following the arrows until a "$\uparrow$" is reached. Figure 1 illustrates the process.

2.2 The maximum-average substring problem

Given a sequence of real numbers, $A = \langle a_1, a_2, \ldots, a_n \rangle$, the maximum-average substring problem is to find, for each position i, a consecutive subsequence of A starting at that position such that the average value of the numbers in the subsequence is maximized. Lin, Jiang and Chao (2002) gave a very interesting linear-time algorithm for solving this problem.

Let $w(A) = \sum_{i=1}^{n} a_i$ be the sum of elements of A. Furthermore, let $d(A) = |A| = n$, be the length of the sequence A. The *average* of A is defined as $\mu(A) = w(A)/d(A)$.

Definition 1 *A sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$ is* right-skew *if and only if the average of any prefix $\langle a_1, a_2, \ldots, a_i \rangle$ is always less than or equal to the average of the remaining suffix subsequence $\langle a_{i+1}, a_{i+2}, \ldots, a_n \rangle$. A partition $A = A_1 A_2 \cdots A_k$ is* decreasingly right-skew *if each segment A_i of the partition is right-skew and $\mu(A_i) > \mu(A_j)$ for any $i < j$.*

The following are some useful properties of right-skew segments and their averages.

Lemma 1 (Combination) *Let A, B be two sequences with $\mu(A) < \mu(B)$. Then $\mu(A) < \mu(AB) < \mu(B)$.*

REPORT-DRS-PART$(i, p[\cdot])$
Input: i denoting the suffix sequence $\langle a_i, a_{i+1}, \ldots, a_n\rangle$; $p[\cdot]$: right-skew pointers of A.
Output: The decreasingly right-skew partition of the suffix.
1 **while** $i \leq n$ **do** $\triangleright$ Reports $\langle a_i, \ldots, a_j\rangle$ as a right-skew segment.
2 OUTPUT $(i, p[i]); i \leftarrow p[i] + 1$

Figure 2: Reporting the decreasingly right-skew partition of a suffix sequence.

Proof. Let $\lambda = d(A)/d(AB)$. We have $\mu(AB) = \lambda\mu(A) + (1-\lambda)\mu(B)$. The result is true because $0 < \lambda < 1$. □

Lemma 2 *Let A, B be two right-skew sequences with $\mu(A) \leq \mu(B)$. Then the sequence AB is also right-skew.*

Proof. Consider a prefix P of AB. Clearly, $\mu(P) \leq \mu(B)$ if $P = A$. If P is a proper prefix of A, *i.e.* $A = PY$ for some nonempty sequence Y, then we have $\mu(P) \leq \mu(A) \leq \mu(Y)$ by the last lemma. Hence, $\mu(P) \leq \mu(YB)$ since $\mu(P) \leq \mu(B)$.

On the other hand, if P contains a proper prefix of B, *i.e.* $B = CD$ and $P = AC$ for some nonempty sequences C and D, then $\mu(C) \leq \mu(B) \leq \mu(D)$. Hence, $\mu(P) = \mu(AC) \leq \mu(D)$ since $\mu(A) \leq \mu(B) \leq \mu(D)$. □

For a sequence $A = \langle a_1, a_2, \ldots, a_n\rangle$, each suffix of A, $\langle a_i, \ldots, a_n\rangle$, defines a decreasingly right-skew partition, denoted as $A_1^{(i)} A_2^{(i)} \cdots A_k^{(i)}$, for some $k \geq 1$. Suppose that $A_1^{(i)} = \langle a_i, \ldots, a_{p[i]}\rangle$, where $p[i]$ is called the *right-skew pointer* of index i. Note that the right-skew pointers of A implicitly encode the decreasingly right-skew partitions for each suffix $\langle a_i, \ldots, a_n\rangle$ of A. Given the right-skew pointers, one can easily report the decreasingly right-skew partitions of a suffix as illustrated in Figure 2. Interestingly, we can compute all right-skew pointers in linear time.

Lemma 3 *The algorithm* DRS-POINT *given in Figure 3 computes all right-skew pointers for a length n sequence in $O(n)$ time.*

Proof. Consider the algorithm DRS-POINT shown in Figure 3. The working pointer i scans the elements of A from right to left. By Lemma 2, two increasingly right-skew segment can be grouped into one right-skew segment and hence, the pair $(i, p[i])$ always represents a segment of A that is right-skew

```
DRS-Point(A)
Input: A sequence A = ⟨a_1, a_2, ..., a_n⟩.
Output: n right-skew pointers of A, encoded by array p[·].
1 for i ← n downto 1 do
2     p[i] ← i; w[i] ← w(a_i); d[i] ← d(a_i);        ▷ Each ⟨a_i⟩ alone is right-skew.
3     while (p[i] < n) and (w[i]/d[i] ≤ w[p[i] + 1]/d[p[i] + 1]) do
4         w[i] ← w[i] + w[p[i] + 1]
5         d[i] ← d[i] + d[p[i] + 1]
6         p[i] ← p[p[i] + 1]
```

Figure 3: Setting up the right-skew pointers in $O(n)$ time.

throughout the entire algorithm. The correctness of the algorithm follows from the fact that the right-skew pointers found by the algorithm encode a partition of each suffix of A with strictly decreasing averages.

We can analyze the time complexity of the algorithm by an amortized argument. We conclude that the amortized cost of each iteration of the for-loop is just a constant.

In short, we deposit a credit whenever a correct value of the right-skew pointer $p[\cdot]$ is found. Later on, when the algorithm needs to advance the $p[\cdot]$ pointer in the while-loop, the skipping cost can be charged to the pre-deposited credits. Since exactly n credits are deposited in the entire process, the while-loop spends at most overall $O(n)$ time. □

It should be noted that $(i, p[i])$ is the maximum-average substring of A starting at position i. Readers are encouraged to define the "left-skew decomposition", and locate the maximum-average substrings ending at each position.

3 Applications

This section describes the applications in details to put the problems and some related results in proper perspective.

3.1 GC-Rich Regions

In all organisms, the GC base composition of DNA varies between 25–75%, with the greatest variation in bacteria. Mammalian genomes typically have a GC content of 45-50%. Nekrutenko and Li (2000) showed that the extent of the compositional heterogeneity in a genomic sequence strongly correlates with its GC content. Genes are found predominantly in the GC-richest isochore classes. Hence, finding GC-rich regions is an important problem in gene recognition and comparative genomics.

Huang (1994) used the expression $x - p \cdot l$ to measure the GC richness of a region, where x is the C+G count of the region, p is a positive constant ratio, and l is the length of the region. In other words, each of nucleotides C and G is given a reward of $1 - p$, and each of nucleotides A and T is penalized by p. Similar expression was used by Sellers (1984) for recognizing patterns by mismatch density. A length cutoff L is given to avoid reporting extremely short optimal regions. Huang extended the well-known recurrence relation used by Bentley (1986) for solving the maximum sum consecutive subsequence problem, and derived a linear-time algorithm for computing the optimal segments with lengths at least L.

Here we explain briefly Huang's idea for computing the maximum sum consecutive subsequence of length at least L. Let $A = \langle a_1, a_2, \ldots, a_n \rangle$ be a DNA sequence of length n. Use $w(X)$ to denote the *score* of nucleotide X, *i.e.* $w(G)$=$w(C)$=$1 - p$, and $w(A)$=$w(T)$=$-p$. Define $S(i)$ to be the maximum score of regions ending at position i of A, which include the empty region. The scores $S(i)$ can be computed by the following recurrence:

$$S(i) = \begin{cases} \max\{S(i-1) + w(a_i), 0\} & \text{if } i > 0, \\ 0 & \text{if } i = 0. \end{cases}$$

Now let us shift along the sequence with a window of size L. For each fixed window, we can compute its score, and then the maximum score of regions ending at the front of the window with the help of the vector S. This results in a linear-time method for computing the maximum sum consecutive subsequence of length at least L.

As noted by Huang, the lengths of the regions reported by the algorithm are usually much greater than the cutoff L. An immediate implication is that they might contain some very poor and irrelevant regions. It is therefore natural to consider bounding the target regions with additional upper bound. Lin, Jiang and Chao (2002) gave an algorithm that can be combined

with Huang's algorithm to yield a linear-time algorithm for computing the maximum sum consecutive subsequence of length between lower bound L and upper bound U.

Huang (2002) has also proposed an interesting alternative measure for finding GC-rich regions. Namely, given a DNA sequence, one would now attempt to find segments of length at least L with the highest C+G ratio. Specifically, each of nucleotides C and G is assigned a score of 1, and each of nucleotides A and T is assigned a score of 0.

```
DNA sequence:     ATGACTCGAGCTCGTCA
Binary sequence:  00101011011011010
```

The maximum-average segments of the binary sequence correspond to segments with the highest GC ratio in the DNA sequence.

He noted that such an optimal segment is of length at most $2L - 1$. This observation yields an $O(nL)$-time algorithm for computing a segment of length at least L with the highest C+G ratio, where n is the length of the input sequence. A linear-time algorithm for this problem was given by Chung and Lu (2004).

3.2 CpG Islands

CpG islands are defined as regions of DNA of at least 200bp (*i.e.* base pairs) in length with G+C content above 50%, and a ratio of observed vs. expected CpGs (CG di-nucleotides) at least 0.6 (Gardiner-Garden and Frommer, 1987). Most of the CpG islands are between 200 and 1400bp with a majority of them being 200–400bp.

CpG islands often occur in the $5'$ regions of genes (Bird, 1987). They are typically a few hundred to a few thousand bases long. Though the widely accepted definition of what constitutes a CpG island was proposed by Gardiner-Garden and Frommer (1987), new definitions and methods for a CpG island are still in progress (Takai and Jones, 2002).

A Markov chain model was introduced by Durbin *et al.* (1998) to decide if a short DNA sequence comes from a CpG island or not. The model consists of a dinucleotide table, which, for each of the 16 different dinucleotides, gives the log likelihood ratio of the frequencies of the dinucleotide in CpG islands and in non-CpG regions. The numbers in the table range from -1.169 for the dinucleotide TA to 1.812 for the dinucleotide CG. The *log-odds score* of a

DNA sequence is the sum of the log-odds scores of every dinucleotide in the sequence. The average score (also called normalized score) of the sequence is obtained by dividing its score by its length. It is known that CpG islands and non-CpG regions can be better discriminated by using average scores than using raw scores. The histogram of the average scores of CpG islands and non-CpG regions in (Durbin et al., 1998) shows that all non-CpG regions have average scores less than 0.1 and most of the CpG islands have average scores greater than 0.1. The average score value of 0.1 could thus be used as a cutoff in deciding if a sequence comes from a CpG island.

The above Markov chain model can be used to locate CpG islands in a long genomic sequence by computing the average score for a window of constant size around every position in the sequence and plotting the scores. However, this approach is not very effective because CpG islands have sharp boundaries and variable lengths (Durbin et al., 1998). We consider an alternative approach to identify CpG islands based on the Markov chain model and program MAVG.

The input genomic sequence is converted into a sequence of real numbers using the dinucleotide table mentioned above. The average (score) of a segment of the number sequence is the sum of the numbers in the segment divided by the length of the segment. The core of a CpG island is defined as a region of length at least 250 bp with the maximum average score. The full extent of a CpG island is a longest region that does not contain any sufficiently long (it i.e. 250 bp or longer) subregion with average score below the cutoff. Lin et al. (2003) implemented an algorithm for enumerating k maximum-average segments with lengths at least L, where L is given parameter, as a C program called MAVG. The empirical tests suggest that the programs MAVG and NEWCPGSEEK, which is a popular existing program for finding CpG islands, are complementary in the sense that a combination of their may provide a more accurate predication of CpG islands.

3.3 Annotating Multiple Sequence Alignments

Conserved regions in biomolecular sequences are strong candidates for functional elements. The most popular methods to compute conserved regions all start with a given multiple sequence alignment (Stojanovic et al., 1999; Stojanovic and Dewar, 2004). Stojanovic et al. (1999) gave several methods for finding highly conserved regions within previously computed multiple alignments. Three of the methods are based on assigning a numerical score

to each column of a multiple alignment and then looking for runs of columns with high cumulative scores. Since the assigned scores may be all positive (*e.g.* in the information content case), each examined column could increase the cumulative score. It follows that the entire alignment could be reported erroneously as a conserved region. Therefore, it is imperative that each column score is adjusted by subtracting a positive *anchor* value. Determining such an anchor value appropriately for each dataset could make the use of a program based on the above approach very complicated. An alternative solution to the above problem is to look for runs of sufficiently many columns in the multiple alignment with the maximum *average* (or *normalized*) score instead. This can be efficiently computed by the algorithm for the length-constrained maximum average consecutive subsequence problem.

3.4 Post-Processing Sequence Alignments

A new popular approach to gene prediction in the human genome is based on comparative analysis of human and mouse DNA. The rationale behind this approach is that similarity between corresponding human and mouse exons is 85% on average, while similarity between introns is 35% on average (Arslan, Eğecioğlu, and Pevzner, 2001). Though the ingenious Smith-Waterman (Smith and Waterman, 1981) local alignment approach has been very successful in revealing highly conserved regions by discarding poorly conserved surrounding regions, a potential drawback of the method is that it may lead to the inclusion of arbitrarily poor internal regions (called the *mosaic effect*).

In an attempt to fix the mosaic effect problem, Zhang et al. (1999) suggested to first run Smith-Waterman type of alignment algorithms and then post-process the computed alignments. They developed an elegant linear-time algorithm that decomposes a long alignment into sub-alignments to avoid the mosaic effect. The method for computing the length-constrained maximum average consecutive subsequences can be used to locate within an alignment the region that is sufficiently long and has the maximum degree of normalized similarity.

3.5 All Maximal Scoring Segments

3.6 Maximum-Scoring Segment Sets

4 Discussions

longest and shortest heaviest segments ...
MaxSubSeq

Acknowledgements

Kun-Mao Chao was supported in part by NSC grants 94-2213-E-002-018 and 94-2213-E-002-091 from the National Science Council, Taiwan.

References

[1] Allison, L. (2003). Longest biased interval and longest non-negative sum interval. Bioinformatics *19*, 1294–1295.

[2] Arslan A., Eğecioğlu, Ö and Pevzner, P. (2001). A new approach to sequence comparison: normalized sequence alignment. Bioinformatics *17*, 327–337.

[3] Bae, S.E. and Takaoka, T. (2004). Algorithms for the problem of k maximum sums and a VLSI algorithm for the k maximum subarrays problem. Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks, 247–253.

[4] Bengtsson, F. and Chen, J. (2004). Efficient algorithms for k maximum sums. Proceedings of the 15th International Symposium on Algorithms And Computation, LNCS *3341*, 137–148.

[5] Bentley, J. (1986). Programming Pearls (Reading: Addison-Wesley).

[6] Bird, A. (1987). CpG islands as gene markers in the vertebrate nucleus. Trends in Genetics *3*, 342-347.

[7] Chen, K.-Y. and Chao, K.-M. (2004). On the range maximum-sum segment query problem. Proceedings of the 15th International Symposium on Algorithms And Computation, LNCS *3341*, 294–305.

[8] Chen, K.-Y. and Chao, K.-M. (2005). Optimal algorithms for locating the longest and shortest segments satisfying a sum or an average constraint. Information Processing Letters, accepted.

[9] Cheng, C.-H., Chen, K.-Y., Tien, W.-C., and Chao, K.-M. (2005). Improved algorithms for the k maximum-sum problems. Proceedings of the 16th International Symposium on Algorithms And Computation, accepted.

[10] Chung, K.-M., and Lu, H.-I. (2004). An optimal algorithm for the maximum-density segment problem. SIAM Journal on Computing *34*, 373–387.

[11] Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C. (1999). Introduction to Algorithms (The MIT Press: 2nd Edition).

[12] Csűrös, M. (2004). Maximum-scoring segment sets. IEEE/ACM Transactions on Computational Biology and Bioinformatics *1*, 139–150.

[13] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). Biological sequence analysis (Cambridge University Press).

[14] Fan, T.-H., Lee, S., Lu, H.-I., Tsou, T.-S., Wang, T.-C., and Yao, A. (2003). An optimal algorithm for maximum-sum segment and its application in bioinformatics. Proceedings of the Eighth International Conference on Implementation and Application of Automata, LNCS *2759*, 251–257.

[15] Fariselli, P., Finelli, M., Marchignoli, D., Martelli, P.L., Rossi, I., and Casadio, R. (2003). MaxSubSeq: an algorithm for segment-length optimization. The case study of the transmembrane spanning segments. Bioinformatics *19*, 500–505.

[16] Fukuda, T., Morimoto, Y., Morishita, S., and Tokuyama, T. (1996). Data mining using two-dimensional optimized association rules: scheme, algorithms, and visualization. Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, 13–23.

[17] Gardiner-Garden, M. and Frommer, M. (1987). CpG islands in vertebrate genomes. J. Mol. Biol. *196*, 261–282.

[18] Grenander, U. (1978). Pattern Analysis (New York: Springer-Verlag).

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

[20] Lin, R.-R., Kuo, W.-H., and Chao, K.-M. (2005). Finding a length-constrained maximum-density path in a tree. Journal of Combinatorial Optimization *9*, 147–156.

[21] Lin, Y.-L., Huang, X., Jiang, T., and Chao, K.-M. (2003). MAVG: locating non-overlapping maximum average segments in a given sequence. Bioinformatics *19*, 151–152.

[22] Lin, Y.-L., Jiang, T., and Chao, K.-M. (2002). Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. Journal of Computer and System Sciences *65*, 570–586.

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

[24] Perumalla, K. and Deo, N. (1995). Parallel algorithms for maximum subsequence and maximum subarray. Parallel Processing Letters *5*, 367–373.

[25] Ruzzo, W.L. and Tompa, M. (1999). A linear time algorithm for finding all maximal scoring subsequences. Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology, 234–241.

[26] Smith, T.F. and Waterman, M.S. (1981). Identification of common molecular sequences. J. Mol. Biol. *147*, 195–197.

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

[28] Stojanovic, N. and Dewar, K. (2005). Identifying multiple alignment regions satisfying simple formulas and patterns. Bioinformatics *20*, 2140–2142.

[29] Takaoka, T. (2002). Efficient algorithms for the maximum subarray problem by distance matrix multiplication. Electronic Notes in Theoretical Computer Science *61*, 1–10.

[30] Tamaki, T. and Tokuyama, T. (1998). Algorithms for the maximum subarray problem based on matrix multiplication. Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 446–452.

[31] Takai, D. and Jones, P.A. (2002). Comprehensive analysis of CpG islands in human chromosomes 21 and 22. PNAS *99*, 3740–3745.

[32] Wang, L. and Xu, Y. (2003). SEGID: identifying interesting segments in (multiple) sequence alignments. Bioinformatics *19*, 297–298.

[33] Zhang, Z., Berman, P., Wiehe, T., and Miller, W. (1999). Post-processing long pairwise alignments. Bioinformatics *15*, 1012–1019.