7-2003

Fast and Space-Efficient Location of Heavy or Dense Segments in Run-Length Encoded Sequences

Ronald I. Greenberg
Rgreen@luc.edu

Recommended Citation
Ronald I. Greenberg. Fast and space-efficient location of heavy or dense segments in run-length encoded sequences. In COCOON: Ninth International Computing and Combinatorics Conference, volume 2697 of Lecture Notes in Computer Science, pages 528--536. Springer-Verlag, 2003.

Fast and Space-Efficient Location of Heavy or Dense Segments in Run-Length Encoded Sequences (Extended Abstract)

Ronald I. Greenberg

Loyola University, 6525 N. Sheridan Rd., Chicago, IL 60626, USA,
rig@cs.luc.edu,
http://www.cs.luc.edu/~rig

Abstract. This paper considers several variations of an optimization problem with potential applications in such areas as biomolecular sequence analysis and image processing. Given a sequence of items, each with a weight and a length, the goal is to find a subsequence of consecutive items of optimal value, where value is either total weight or total weight divided by total length. There may also be a specified lower and/or upper bound on the acceptable length of subsequences. This paper shows that all the variations of the problem are solvable in linear time and space even with non-uniform item lengths and divisible items, implying that run-length encoded sequences can be handled in time and space linear in the number of runs. Furthermore, some problem variations can be solved in constant space. Also, these time and space bounds suffice for certain problem variations in which we call for reporting of many "good" subsequences.

Keywords: maximum consecutive subsequence sum, maximum-density segments, biomolecular sequence analysis, bioinformatics, image processing, data compression

1 Introduction

Let S be a sequence comprised of n *runs*, where the ith run ($1 \leq i \leq n$) has weight w_i and length $l_i \geq 0$. Initially, we define a segment of S to be a consecutive subsequence of runs, i.e., segment $S(i,j)$ is comprised of runs i through j. The weight of $S(i,j)$ is

$$\mathit{weight}(i,j) = \sum_{k=i}^{j} w_k \ ,$$

the length of $S(i,j)$ is

$$\mathit{length}(i,j) = \sum_{k=i}^{j} l_k \ ,$$

and the density of $S(i, j)$ is

$$density(i, j) = weight(i, j)/length(i, j) \ .$$

Prior works on algorithms for biomolecular sequence analysis have considered the problem of finding a segment of S that is *heaviest* (maximizing $weight(i, j)$) or *densest* (maximizing $density(i, j)$), subject to constraints that the segment length must be at least L and/or at most U [1–4].[1] For brevity, we will refer to these as an L constraint and/or U constraint. In addition, the heaviest segment problem with no constraints on segment length is discussed by Bentley [5]. (This version of the problem was motivated by image processing tasks.) Note that for heaviest segments only we may consider empty segments, which may be represented by choosing $i > j$.

Most of the prior results were for the *uniform* version of the problem in which $l_i = 1$ for all i; exceptions will be noted below. (In the uniform case, the problem is often described as one of finding a subsequence of consecutive items of maximum sum or of maximum average.) We will also introduce below a new variation of the non-uniform version of the problem that we will refer to as the non-uniform case with breakable or non-atomic runs. With non-atomic runs, we will allow each end of a segment to include just a portion of the length of a run. When a run is partially included in a segment, a pro rata portion of its weight will be included in the weight of the segment.

While the non-uniform problem with atomic runs was considered by Goldwasser et al. [1] and an interesting application might be discovered, a particularly interesting use of the non-uniform model would be for working with sequences that have been compressed. That is, given a sequence under the uniform model, a simple compaction would be to replace any set of r consecutive items of weight w with a run of length r and weight wr under the non-uniform model, which corresponds to the standard compression technique of *run-length encoding*. Run-length encoding tends to be particularly useful in monochrome image processing. It could also have potential for such applications as DNA sequence analysis, since each item in a DNA sequence is chosen from just four different nucleotides. (Furthermore, in DNA sequence analysis, researchers may often be interested, on the first pass, in just a binary distinction between C/G and other [6–10].) To work with such a compressed sequence but be able to find a heaviest or densest segment in the uncompressed sequence, we must be able to break runs.

We begin by reviewing prior results for heaviest segments and then for densest segments. When discussing the problems together, we may use the term *optimal* to mean heaviest or densest, and we may refer to the weight or density of a segment as its *value*.

The first result was an unpublished result of Kadane [5] for the unconstrained heaviest segment problem. This solution uses $O(n)$ time and constant space (be-

[1] Note that Lin et al. [3] use the term "heaviest" to mean what we define as "optimal" below.

yond the space used to represent the input)[2]. With an L constraint, an algorithm of Huang [4] can be used to find a heaviest segment in $O(n)$ time and $O(n)$ space as noted by Lin et al. [3]. Lin et al. further showed how to obtain the same time and space bounds with both an L constraint and a U constraint [3].

In the case of finding a densest segment, the problem is trivially solvable in $O(n)$ time and constant space if there is no L constraint; just find the single run of maximum density. Lin et al. [3] showed that with an L constraint, a densest segment can be found in time and space $O(n \lg L)$. Goldwasser et al. [2, 1] improved the time and space bounds to $O(n)$. They further showed that the same bounds hold with both an L constraint and a U constraint [1]. In addition, they showed that with only an L constraint, the results could be extended to the non-uniform version of the problem [1]. Finally, they showed that with both constraints, $O(n+n\lg(U-L+1))$ time suffices when $l_i \geq 1$ for all i. Goldwasser and I have, however, observed that the analysis in [1] can be modified to yield $O(n)$ time and space for any lengths satisfying $l_i \geq 0$ for all i.

This paper explains in Sect. 2 why all the results mentioned so far can be extended to the non-uniform model with atomic runs. In Sect. 3, we show that the space usage for finding a heaviest segment with an L constraint can be reduced to constant space. In Sect. 4, we show that all the results can be maintained even if we allow breakable runs. As indicated above, the use of breakable runs is of particular interest in connection with run-length encoded sequences, but the results of Sect. 4 apply even when l_i values are allowed to be nonintegral and when runs can be broken into any fraction. In Sect. 5, we consider variations on the problem in which we seek not just one optimal segment but all optimal segments or all optimal segments of maximal or minimal length, or even a more general concept as considered by Huang [4].

2 The Non-Uniform Model

Most of the prior algorithms for finding a heaviest segment or finding a densest segment may essentially be cast into the following basic framework. (The approach of Huang [4] is somewhat exceptional, and we show in Section 3 that it can be greatly simplified when we seek only a heaviest segment with L constraint.) We sweep left to right across the given sequence considering each position in turn as a possibility for the right endpoint of an optimal segment. At each such step we determine a best choice of the left endpoint, called a "good partner" for the current right endpoint, that is at least as far to the right as the prior good partner. It is relatively easy to see that the good partner should never "back up" when seeking a heaviest segment. For densest segments, the

[2] This measure of space usage is analogous to the concept of algorithms that sort in-place (e.g., [11]) by using only a constant amount of storage outside the input array. Interestingly, the algorithm of Kadane has an even stronger property that one need not store the entire input array at one time; rather one may read the input piecemeal and never use more than constant storage in a strict sense.

correctness of this approach is based on the following lemma that is essentially the same as one proven by Goldwasser et al. [2, Lemma 9]:

Lemma 1. *Let $S(i, j)$ be a densest segment among those ending at index j and having length at least L. Similarly, let $S(i', j')$ be a densest segment among those ending at index j' and having length at least L. If $j' > j$ and $i' < i$, then density$(i, j) \geq$ density(i', j').* □

Except in the unconstrained heaviest segment problem, the existing algorithms make use of a cleverly precomputed data structure of size $O(n)$ to determine how far to move the left endpoint at each step without overshooting the proper location for the good partner of the right endpoint. (In the unconstrained heaviest segment problem, no such data structure is necessary, because a simple check indicates whether the left endpoint should stay at the same position as in the last step or move to the same position as the right endpoint.)

For the most part, the l_i values of the input sequence are irrelevant to the operation of the algorithms that find an optimal segment. The main place they have an effect is in providing an additional constraint (beside the constraint that the good partner does not back up) on the range of indices to consider for the current good partner. In the uniform case, the additional constraint is trivial; a good partner of position j must be in the range $j - U$ to $j - L$. In the non-uniform case, however, these constraints are easily precomputed. In $O(n)$ time and space, a simple scan through the input sequence allows us to calculate U_j and L_j for all j, such that the good partner for j is between U_j and L_j. Instead of precomputing, these constraints can actually be managed on the fly, so that constant space will suffice for finding a heaviest sequence with an L constraint as shown in Section 3.

There is one more complication involved in finding a densest segment with L and U constraints. This problem is actually solved by dividing the input sequence into contiguous blocks of length $U - L$ before proceeding with any other operations. Thus, the problem of finding a good partner breaks down into a problem of comparing a good partner found in a specific block with no explicit U constraint to a good partner found in the next farther block with no explicit L constraint. Whereas Goldwasser et al. [1] proposed dividing the sequence into blocks of $U - L$ runs, Goldwasser and I have observed that dividing into blocks of length at least $U - L$ and as close as possible to $U - L$ yields $O(n)$ time and space for finding a densest segment.

The above observations are encapsulated in the following theorem:

Theorem 2. *$O(n)$ time suffices to find a length-constrained heaviest segment or densest segment even with non-uniform run lengths.* □

All results given later in this paper will also be applicable to the non-uniform model.

3 Constant-Space Location of a Heaviest Segment with an L Constraint

In this section, we show that a heaviest segment with an L constraint but no U constraint can be found in $O(n)$ time and constant space, improving on the $O(n)$ space result that follows from the approach of Huang [4].

As in the approaches discussed in Sect. 2, we make a scan left to right across the input sequence, considering each position in turn as a possible location for the right endpoint of a heaviest segment. As we do so, we keep track of the good partner (a best left endpoint for the current right endpoint), which also moves rightward. Since $weight(i, j)$ is just $weight(i, j-1) + w_j$, a good partner p of $j-1$ serves as a good partner of j unless a segment of higher weight than $S(p, j)$ is obtained by considering left endpoints p' with $length(p', j) \geq L \geq length(p', j-1)$. By keeping track of the location that is length L away from j, as well as keeping track of the current good partner and the heaviest segment seen so far, we can find a heaviest segment in $O(n)$ time and constant space. We present the algorithm in Fig. 1, but, for simplicity, we find only the *weight* of a heaviest segment; it should be clear that we could easily keep track of an actual heaviest segment as well. The pseudocode in Fig. 1 also incorporates the use of non-uniform lengths as discussed in Sect. 2.

```
1   Lherestart ← 1
2   maxsofar ← maxendinghere ← Lherewt ← Lherelength ← 0
3   for j ← 1 to n do
4       Lherelength ← Lherelength + l_j
5       Lherewt ← Lherewt + w_j
6       maxendinghere ← maxendinghere + w_j
7       while Lherestart ≤ j and Lherelength − l_Lherestart ≥ L do
8           Lherelength ← Lherelength − l_Lherestart
9           Lherewt ← Lherewt − w_Lherestart
10          Lherestart ← Lherestart + 1
11          maxendinghere ← max{maxendinghere, Lherewt}
12      endwhile
13      if Lherelength ≥ L
14      then maxsofar ← max{maxsofar, maxendinghere}
15      endif
16  endfor
```

Fig. 1. The algorithm to find the weight of a heaviest segment with L constraint and non-uniform lengths in $O(n)$ time and constant space.

We summarize with the following theorem:

Theorem 3. *A heaviest segment with length greater than or equal to L can be found in $O(n)$ time and constant space.* □

4 Non-Atomic Runs

In this section, we show that all the results so far can be extended to work with non-atomic runs. Note that all the running times remain linear in the number of runs and that L, U, and the l_i values may even be nonintegral. We make use of the following two lemmas:

Lemma 4. *To find an optimal (heaviest or densest) segment, we need only consider a segment containing a partial run if its length is exactly L or U.*

Proof. Consider a segment of length strictly between L and U that contains a partial run. Then the length constraints allow us to use more of this run or trim off part of this run. One of these changes must not decrease the value (weight or density) of the segment, since the density of a run is considered to be uniform. □

Lemma 5. *To find an optimal segment, we may limit attention to segments with a partial run on at most one end.*

Proof. Consider a segment with partial runs on each end. Without loss of generality, suppose that that the run truncated on the left has lower density than the run truncated on the right. If the utilized portion of the run on the left is shorter than the unutilized portion of the run on the right, we can completely eliminate the run on the left and add a corresponding portion of the run on the right. On the other hand, if the utilized portion of the run on the left is longer than the unutilized portion of the run on the right, we can completely include the run on the right at the expense of the run on the left. Either way, neither the weight nor density of the segment will decrease. □

With these lemmas in mind, we see that we need not deviate too far from working with atomic runs to obtain an optimal segment when we are allowed to break runs. We need only consider small adjustments in addition to each of the segments considered as a possible optimal segment when working with atomic runs. For example, to update the algorithm of Fig. 1 to work with breakable runs, we can just add the following code after Line 11:

if $Lherelength - L < l_j$
then $Lhereadjwt \leftarrow Lherewt - (Lherelength - L)w_j/l_j$
$maxendinghere \leftarrow \max\{maxendinghere, Lhereadjwt\}$
endif

and the following code after Line 14:

$Lhereadjwt \leftarrow Lherewt - (Lherelength - L)w_{Lherestart}/l_{Lherestart}$
$maxsofar \leftarrow \max\{maxsofar, Lhereadjwt\}$

(Note that this modification is correct under the assumption that $L \geq 0$, which places no restriction on the utility of the algorithm, since $l_i \geq 0$ for all i.)

Incorporating other variations of the problem as well, we can state the following theorem:

Theorem 6. *$O(n)$ time and space suffices to find a length-constrained heaviest segment or densest segment even with non-atomic runs. Furthermore, constant space suffices to find a heaviest segment with no upper bound on segment length.* □

5 Finding "All" Optimal Segments

Huang's algorithm [4] actually generates not only a heaviest segment (of length at least L) but all heaviest segments that cannot be extended, i.e., that are of maximal length. In fact, Huang's algorithm finds all segments of maximal length that are at least as heavy as any overlapping segment.[3] In this section, we show that if one seeks to report even all optimal segments, the run time may be $\Theta(n^2)$ in the worst case, but reporting all optimal segments of *minimal* length does not change any of the time and space bounds discussed so far. We also show that Huang's goal can be achieved for heavy segments with a simpler algorithm and that the space usage can be reduced from two numeric arrays of length n to a single boolean array of length n.

We begin with the worst-case lower bound for finding all optimal segments. This result is actually quite trivial, since an input sequence with $w_i = 0$ for all i makes every segment optimal with $L = 0$ and no U constraint.

Theorem 7. *Finding all optimal segments requires $\Theta(n^2)$ time in the worst case.* □

(It is also possible to construct more interesting sequences with many optimal segments. Furthermore, while a U constraint will keep the number of optimal segments below n^2, the number of optimal segments may still be $\Theta(nU)$.)

We now note that finding all optimal segments of minimal length can be done with the same time and space bounds as finding one optimal segment:

Theorem 8. *Reporting all optimal segments of minimal length can be done in $O(n)$ time and space. For heaviest segments with no U constraint, the space can be reduced to $O(1)$.*

Proof. We can begin by simply finding the optimal value (weight or density of an optimal segment) using the algorithms discussed so far. Then we can essentially rerun the same algorithm but report an optimal segment each time that we find a good partner of a right endpoint for which the value of the corresponding segment equals the optimal value. The only remaining detail is that where there is a tie among candidates for the good partner, we must choose the rightmost good partner. This choice is easy to make; for example in the algorithm for heaviest sequences with an L constraint (Fig. 1), we could maintain good partner information as well as best value information at Lines 11 and 14. When there is a tie in the two values to which we are applying the max operator, we would retain the good partner corresponding to *Lherewt* in preference to *maxendinghere*, and *maxendinghere* in preference to *maxsofar*. □

[3] Note that Huang uses the term "optimal" for a different meaning than in this paper.

Finally, we give a simpler method than Huang's [4] to find all segments of maximal length (with no U constraint) that are at least as heavy as any overlapping segment, and we reduce the space usage. For simplicity, we stick to the uniform model ($l_i = 1 \forall i$) and $L = 0$. The enhancements of non-uniform lengths, breakable runs, and $L > 0$ can be incorporated by combining ideas presented in earlier sections of this paper. In the case we focus on now, however, our statement about space usage can be particularly strong; it includes even space used to store input data as long as we are allowed to read the input twice.

Theorem 9. *All segments of maximal length that are at least as heavy as any overlapping segment can be reported in $O(n)$ time using just one boolean array of length n plus constant additional space.*

Proof. The key observation is that $S(i, j)$ satisfies the criterion if and only if (1) the heaviest non-empty segment ending with position $i-1$ and the heaviest non-empty segment beginning with position $j+1$ each have negative weight (if they exist), and (2) for any position from i to j, the heaviest segment beginning there and the heaviest segment ending there have nonnegative weight. The algorithm in Fig. 2 completes the proof. (For simplicity, we report only non-empty segments.) □

6 Conclusion

We have seen that finding a length-constrained heaviest segment or densest segment in a sequence composed of either atomic or non-atomic items, each of arbitrary weight and nonnegative length, can be accomplished in time and space linear in the number of items. Furthermore, constant space suffices to find a *heaviest* segment when there is no upper bound on segment length. In addition, the same results apply to finding all optimal segments of minimal length. A remaining open problem is to improve on the linear space requirement for variations of the problem that involve an upper bound on segment length or finding a *densest* segment.

7 Acknowledgments

Thank you to Michael Goldwasser of Loyola University for helpful discussions.

References

1. Goldwasser, M.H., Kao, M.Y., Lu, H.I.: Linear-time algorithms for computing maximum-density sequence segments with bioinformatics applications. Manuscript available at http://www.cs.luc.edu/ mhg/publications/DensityPreprint.pdf (2002)
2. Goldwasser, M.H., Kao, M.Y., Lu, H.I.: Fast algorithms for finding maximum-density segments of a sequence with applications to bioinformatics. In: Proc. of the Second Annual Workshop on Algorithms in Bioinformatics, Springer-Verlag (2002) 157–171

```
1   maxbeghereneg[n + 1] ← TRUE
2   maxbeginninghere ← 0
3   for j ← n downto 1 do
4       if maxbeginninghere + w_j < 0 then
5           maxbeginninghere ← 0
6           maxbeghereneg[j] ← TRUE
7       else
8           maxbeginninghere ← maxbeginninghere + w_j
9           maxbeghereneg[j] ← FALSE
10      endif
11  endfor
12  i ← 1
13  maxendinghere ← 0
14  for j ← 1 to n do
15      if maxendinghere + w_j < 0 then
16          maxendinghere ← 0
17          i ← j + 1
18      else
19          maxendinghere ← maxendinghere + w_j
20          if maxbeghereneg[j + 1] then Output (i, j) endif
21      endif
22  endfor
```

Fig. 2. The algorithm to report all segments of maximal length that are at least as heavy as any overlapping segment. In this code, *maxendinghere* and *maxbeginninghere* represent weights of possibly empty segments, whereas *maxbeghereneg* is an boolean array for flagging positions from which the heaviest non-empty interval is negative.

3. Ling Lin, Y., Jiang, T., Mao Chao, K.: Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. In: 27th International Symposium on Mathematical Foundations of Computer Science. Volume 2420 of Lecture Notes in Computer Science., Springer-Verlag (2002) 459–470 To appear in Journal of Computer and System Sciences.
4. Huang, X.: An algorithm for identifying regions of a DNA sequence that satisfy a content requirement. Computer Applications in the Biosciences **10** (1994) 219–225
5. Bentley, J.: Programming Pearls. Second edn. Addison-Wesley (2000)
6. Hannenhalli, S., Levy, S.: Promoter prediction in the human genome. Bioinformatics **17** (2001) S90–96
7. Nekrutenko, A., Li, W.H.: Assessment of compositional heterogeneity within and between eukaryotic genomes. Genome Research **10** (2000) 1986–1995
8. Larsen, F., Gundersen, R., Lopez, R., Prydz, H.: CpG islands as gene marker in the human genome. Genomics **13** (1992) 1095–1107
9. Hardison, R.C., Krane, D., Vandenbergh, C., Cheng, J.F.F., Mansberger, J., Taddie, J., Schwartz, S., Huang, X., Miller, W.: Sequence and comparative analysis of the rabbit alpha-like globin gene cluster reveals a rapid mode of evolution in a C+G rich region of mammalian genomes. Journal of Molecular Biology **222** (1991) 233–249
10. Gardiner-Garden, M., Frommer, M.: CpG islands in vertebrate genomes. Journal of Molecular Biology **196** (1987) 261–282
11. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. Second edn. McGraw-Hill (2001)