Near linear time construction of an approximate index for all maximum consecutive sub-sums of a sequence

Ferdinando Cicalese[1], Eduardo Laber[2], Oren Weimann[3], and Raphael Yuster[4]

[1] Department of Computer Science, University of Salerno, Italy
cicalese@dia.unisa.it
[2] Department of Informatics, PUC-Rio, Rio de Janeiro, Brazil
laber@inf.puc-rio.br
[3] Department of Computer Science, University of Haifa, Israel, oren@cs.haifa.ac.il
[4] Department of mathematics, University of Haifa, Israel, raphy@math.haifa.ac.il

Abstract. We present a novel approach for computing all maximum consecutive subsums in a sequence of positive integers in near linear time. Solutions for this problem over binary sequences can be used for reporting existence (and possibly one occurrence) of Parikh vectors in a bit string. Recently, several attempts have been tried to build indexes for all Parikh vectors of a binary string in subquadratic time. However, to the best of our knowledge, no algorithm is know to date which can beat by more than a polylogarithmic factor the natural $\Theta(n^2)$ exhaustive procedure. Our result implies an approximate construction of an index for all Parikh vectors of a binary string in $O(n^{1+\eta})$ time, for any constant $\eta > 0$. Such index is approximate, in the sense that it leaves a small chance for false positives, i.e., Parikh vectors might be reported which are not actually present in the string. No false negative is possible. However, we can tune the parameters of the algorithm so that we can strictly control such a chance of error while still guaranteeing strong sub-quadratic running time.

Parikh vectors, maximum subsequence sum, approximate pattern matching, approximation algorithms

1 Introduction

Let $\mathbf{s} = s_1, \ldots, s_n$ be a sequence of non-negative integers. For each $\ell = 1, \ldots, n$, we denote with m_ℓ the maximum sum over a consecutive subsequence of $\mathbf{s}$ of size ℓ, in formulae:

$$m_\ell = \max_{i=1,\ldots,n-\ell+1} \sum_{j=i}^{i+\ell-1} s_j.$$

The Maximum Consecutive Subsums Problem (MCSP) asks for computing m_ℓ for each $\ell = 1, \ldots, n$.

Since an obvious implementation of the above formula allows to compute m_ℓ for a single value of ℓ in $O(n)$ time, it follows that there exists a trivial

$O(n^2)$ procedure to accomplish the above task. The interesting question is then about subquadratic procedure for solving MCSP. Notwithstanding quite some effort which has been recently devoted to the problem, surprisingly enough, no algorithm is known which is significantly better than the natural $\Theta(n^2)$.

In this paper we show that we can closely approximate the values m_ℓ (within an approximation factor as close to 1 as desired) with a procedure whose running time can be as close to linear as desired. More precisely, our main result is as follows.

Main Theorem. *For any $\epsilon, \eta > 0$, there exists an algorithm that computes values $\tilde{m}_1, \ldots, \tilde{m}_n$ such that $1 \leq \tilde{m}_j / m_j \leq 1 + \epsilon$, for each $j = 1, \ldots, n$, in time $O(k_{\epsilon,\eta} \cdot n^{1+\eta})$, where $k_{\epsilon,\eta}$ is a constant only depending on ϵ and η.*

MCSP arises in several scenarios of both theoretical and practical interest. Our main motivation for studying MCSP comes from the following Parikh vector matching problem.

An Index for constant time Parikh vector membership queries. Given a string t over an alphabet $[\sigma] = \{1, 2, \ldots, \sigma\}$, for each $c \in [\sigma]$, let x_c be the number of occurrences of character c in t. The vector $(x_1, \ldots, x_\sigma)$ is called the Parikh vector of t.

In the Parikh vector matching problem, given a string t (the text) over the alphabet $[\sigma]$, together with a vector of non-negative integers $p = (x_1, \ldots, x_\sigma)$ (the Parikh vector pattern) we ask for (all/one/existence) of occurrences of substrings s of t such that the Parikh vector of s is p.

Equivalently, we are asking for all the *jumbled* occurrences of a string s (the pattern) in a string t (the text), i.e., any occurrence of some permutation of s in t. Therefore, the Parikh vector matching problem is a type of approximate string matching [16, 10].

Typical applications of such model come from interpretation of mass spectrometry data analysis [5]. More generally, Parikh vector matching applies in scenarios where, notwithstanding the linearity (mono-dimensional) of the structure in which we perform the pattern search, it is not important the order in which what we are searching for, actually occurs. A typical example might be testing for bio-chemical characteristics of (part) of a macromolecule which only depends upon the presence of some substructures and their occurrence within a relatively short distance, whilst their relative order is not significant [1, 3, 15, 18].

Given an alphabet $[\sigma]$ and a string s it is not hard to see that the maximum number of Parikh vectors occurring in s is $O(n^2)$, since each Parikh vector is associated to at least one of the $\Theta(n^2)$ substrings of s. On the other hand, for any given length n there exist $\Omega(n^\sigma)$ distinct Parikh vectors.[1] This also motivates filtering algorithms based on executing membership queries before looking for the actual occurrences of the Parikh vector query.

[1] The number of Parikh vector of length n is equal to the number of ways we an split n elements into σ parts, which is exactly $\binom{n+\sigma-1}{\sigma-1}$.

For binary string t, knowing for each $\ell = 1, \ldots, n$ the minimum and maximum number of 1's found in a substring of size ℓ of t, we can answer membership queries in constant time. More precisely, the connection between MCSP and Parikh vector membership query problem is given by the following.

Lemma 1. *[9] Let s be a binary string and for each $i = 1, \ldots, n$ let $\mu_\ell^{\min}$ (resp. $\mu_\ell^{\max}$) denote the minimum (resp. maximum) number of ones in a substring of s of lenght ℓ. Then for any Parikh vector $p = (x_0, x_1)$ we have that there exists a substring in s with Parikh vector p if and only if $\mu_{x_0+x_1}^{\min} \leq x_1 \leq \mu_{x_0+x_1}^{\max}$.*

It follows that, after constructing the tables of $\mu^{\min}$'s and $\mu^{\max}$'s—which is equivalent to solving two instances of the MCSP—we can answer in constant time Parikh vector membership queries, i.e., questions asking: "Is there an occurrence of the Parikh vector (x_0, x_1) in s?" This is achieved by simply checking the condition in the previous Lemma.

Therefore, there has been significant interest in trying to solve the MCSP on binary sequences in subquadratic time.

To the best of our knowledge the best known constructions are as follows: Burcsi *et al.* in [9, 8] showed a $O(n^2/\log n)$-time algorithm which is based on the $O(n^2/\log n)$ algorithm of Bremner *et al.* [7, 12] for computing $(\min, +)$-convolution; Moosa and Rahman in [17] obtained the same result by a different use of $(\min, +)$-convolution, moreover, they show that an $O(n^2/\log^2 n)$ construction can be obtained assuming word-RAM operations.

Another interesting application of MCSP is in the following problem in the analysis of statistical data.

Finding large empty regions in data sets. Bergkvist and Damaschke in [4] used the index of all maximum consecutive subsums of a sequence of numbers for speeding up heuristics for the following problem: Given a sequence of positive real numbers $x_1, \ldots, x_n$, called items, and integers $s \geq 1$ and $p \geq 0$, find s pairwise disjoint intervals with total of $s + p$ items and maximum total length. Here an interval is a set of consecutive items $x_i, x_{i+1}, \ldots, x_j$ $(i \leq j)$ and its length is $x_i + x_{i+1} + \ldots x_j$. This problem, aka DIMAXL (Disjoint Intervals of Maximum Length) and its density variant where we are interested in interval of maximum density [12], rather than absolute length, are motivated by the problem of finding large empty regions (big holes) in data sets. Several other motivating applications can be found in [4] and [12].

By employing a geometric argument, in [4] a heuristic procedure is presented for the MCSP which can solve the problem in $O(n^{3/2})$ time in the best case, but whose worst case remains quadratic.

2 The Approximate Index

Let $\mathbf{s} = s_1, \ldots, s_n$ be a sequence of non-negative integers. Our aim is to compute $m_\ell = \max_{i=1,\ldots,n-\ell+1} \sum_{j=i}^{i+\ell-1} s_j$ for each $\ell = 1, \ldots, n$. In this section we will prove the following result

Theorem 1. *For any constant $\epsilon, \eta \in (0,1)$, let k be the minimum positive integer such that the positive real solution $\tilde{\alpha}$ of the equation $\alpha = 1 + 1/\alpha^k$, satisfies $1 + \epsilon > \tilde{\alpha}$. Let $t = \lceil 1/\eta \rceil$.*

There exists an algorithm which in time $O(k^{t-1} \cdot n^{1+\eta})$ computes values $\tilde{m}_1, \ldots, \tilde{m}_n$ such that $1 \leq \tilde{m}_\ell / m_\ell \leq 1 + \epsilon$, for each $\ell = 1, \ldots, n$.

We shall start describing our approach by first presenting an algorithm which computes a $(\frac{1+\sqrt{5}}{2})$-approximation of the values m_ℓ's in $O(n^{3/2})$ time. Then we will show a variant of the approach which can be parametrized to achieve the desired $(1+\epsilon)$-approximation in time $O(n^{1.5})$. Finally, we will show how, by recursively applying such a strategy we can achieve the same approximation in time $O(n^{1+\eta})$, for any constant $\eta > 0$.

We will use the following simple facts.

Fact 1 *For each $1 \leq i < j \leq n$ it holds that $m_i \leq m_j$.*

Fact 2 *For each $\ell \in [n]$ and positive integers i, j such that $i + j = \ell$, it holds that $m_\ell \leq m_i + m_j$.*

Fact 1 directly follows by the non-negativity of the elements in the sequence. Fact 2 is a consequence of the following easy observation. For a fixed ℓ, let $s_r, \ldots, s_{r+\ell-1}$ be a subsequence achieving $s_r + \cdots + s_{r+\ell-1} = m_\ell$. By definition we have that, for any i, j such that $i + j = \ell$ it holds that $s_r + \cdots + s_{r+i-1} \leq m_i$ and $s_{r+i} + \cdots + s_{r+\ell-1} \leq m_j$, from which we obtain the desired inequality.

For ease of presentation, we shall usually neglect rounding necessary to preserve the obvious integrality constraints. The reader can assume that, when necessary, numbers are rounded to the closest integer. It will always be clear that these inaccuracies do not affect the asymptotic results. On the other hand, this way, we gain in terms of much lighter expressions.

2.1 Warm-up: A golden ratio approximation in $O(n^{3/2})$

Let $\alpha = \frac{1+\sqrt{5}}{2}$. The value α defines the approximation of our solutions. Fix an integer $t \geq 2$ and set $g = n^{1/2}$.

The basic idea is to compute via exhaustive search the value of $m_{j \times g}$ for each $j = 1, \ldots, n/g$ and then to use these values for approximating all the others.

For each $j = 1, \ldots, n/g$, we set $\tilde{m}_{j \times g} = m_{j \times g}$, i.e., our approximate index will contain the exact value.

Let ℓ be such that $j \times g < \ell < (j+1) \times g$ for some $j = 1, \ldots, n/g - 1$. By Fact 1 we have that $m_{j \times g} \leq m_\ell \leq m_{(j+1) \times g}$. Therefore, if $m_{(j+1) \times g} / m_{j \times g} \leq \alpha$, by setting $\tilde{m}_\ell = m_{(j+1) \times g}$ we also have that $\tilde{m}_\ell$ is an α-approximation of the real value m_ℓ.

What happens if the gap between $m_{(j+1) \times g}$ and $m_{j \times g}$ is large?

If $m_{(j+1) \times g} / m_{j \times g} > \alpha$, our idea is to compute exhaustively m_ℓ for each $\ell = j \times g + 1, \ldots, (j+1) \times g - 1$. The critical point here is that the above "large" gap can only happen once! This is formalized in the following argument: Let

$j > 0$ be the minimum integer such that $m_{(j+1)\times g}/m_{j\times g} > \alpha$. Therefore, by Fact 1 we also have

$$m_{(j+1)\times g}/m_g > \alpha. \tag{1}$$

Now, for each $i > j$, we have

$$\frac{m_{(i+1)\times g}}{m_{i\times g}} \leq \frac{m_{i\times g} + m_g}{m_{i\times g}} = 1 + \frac{m_g}{m_{i\times g}} \leq 1 + \frac{m_g}{m_{(j+1)\times g}} < 1 + 1/\alpha = \alpha, \tag{2}$$

where the first inequality follows by Fact 2; the second inequality follows by Fact 1 together with $i \geq (j+1)$, respectively; the third inequality follows from (1) and the last equality because $\alpha = (1+\sqrt{5})/2$.

Therefore, after encountering the first large gap between $m_{j\times g}$ and $m_{(j+1)\times g}$ all the following gaps will be "small", hence we can safely set $\tilde{m}_\ell = m_{(i+1)\times g}$ for each $i > j$ and $i \times g < \ell \leq (i+1) \times g$. In fact, using again Fact 1 we have $\tilde{m}_\ell/m_\ell \leq m_{(i+1)\times g}/m_{i\times g}$ and then by (2) we are guaranteed that $\tilde{m}_\ell = m_{(i+1)\times g}$ is indeed an α-approximation of the exact m_ℓ.

2.2 As closed to 1 as wished

In this section we prove our main result which follows by refining the algorithm described in the previous section. Here we use the expression "*compute m_ℓ exhaustively*" (for some fixed ℓ) to indicate the linear time computation attained by scanning the sequence $\mathbf{s}$ from left to right and computing the sum of all consecutive subsequences of size ℓ. This exhaustive computation for a single ℓ can be clearly achieved in $\Theta(n)$ time.

Let k be the minimum positive integer such that $\alpha \leq 1 + \epsilon$, where α is the positive real solution of the equation $\alpha = 1 + 1/\alpha^k$. In an explicit way, $k = \lceil -\frac{\ln(\epsilon)}{\ln(1+\epsilon)} \rceil$.

The value α defines the approximation of our solutions. Let us also set $g = n^{1/2}$.

We partition the list of values m_ℓ ($\ell = 1, \ldots, n$) into n/g intervals, each of them with g consecutive values. Then, we proceed as follows:

1. Compute exhaustively the exact value for each m_ℓ in the first interval, i.e., for m_ℓ such that $\ell = 1, \ldots, g$.
2. Compute exhaustively the exact value for the extremes of all intervals (i.e., for m_ℓ such that $\ell = 2g, 3g, \ldots$).
3. Let $m_g, m_{2g}, \ldots$ be the extremes of the n/g intervals. We say that an extreme $m_{i\times g}$ is relevant if $m_{i\times g}/m_{(i-1)\times g} > \alpha$. Compute exhaustively m_ℓ for every ℓ such that the rightmost extreme of its interval is among the first k relevant extremes.
4. The remaining points are approximated by the value of the rightmost point in the interval where they lie (i.e., for $\ell \in \{jg+1, \ldots, (j+1)g-1\}$ such that $m_{(j+1)g}/m_{jg} \leq \alpha$ we set $\tilde{m}_\ell = m_{(j+1)g}$).

We have to prove that the values $\tilde{m}$'s satisfy the desired $(1+\epsilon)$-approximation.

Let $j_1, \ldots, j_r$, with $r \leq k$, be the values of j for which the algorithm verified $m_{j\times g}/m_{(j-1)\times g} > \alpha$ and hence computed exhaustively m_ℓ in the interval $\ell = (j-1)g, \ldots, jg$.

It is easy to see that $\tilde{m}_\ell/m_\ell \leq \alpha$ for each $\ell \leq j_r \times g$.

Moreover, if $r < k$, it also means that $m_{j\times g}/m_{(j-1)\times g} \leq \alpha$ for each $j > j_r$. Hence, we also have that $\tilde{m}_\ell/m_\ell \leq \alpha$ for each $\ell > j_r \times g$.

Assume now that $r \geq k$. Let us write j^* for j_r. Since for k times we had that $m_{j\times g}/m_{(j-1)\times g} > \alpha$, we have in particular that

$$\frac{m_{j^*\times g}}{m_g} > \alpha^k. \tag{3}$$

Now, let us consider an integer ℓ such that $i \times g < \ell < (i+1) \times g - 1$ for some $i \geq j^*$. In such case we have that $\tilde{m}_\ell = m_{(i+1)\times g}$. Therefore, for our purposes, it is enough to show that $\frac{m_{(i+1)\times g}}{m_\ell} \leq \alpha$.

Indeed we have

$$\frac{m_{(i+1)\times g}}{m_\ell} \leq \frac{m_{(i+1)\times g}}{m_{i\times g}} \leq \frac{m_{i\times g} + m_g}{m_{i\times g}} = 1 + \frac{m_g}{m_{i\times g}} \leq 1 + \frac{m_g}{m_{j^*\times g}} < 1 + \frac{1}{\alpha^k} = \alpha.$$

The first inequality follows by Fact 1, since $\ell > i \times g$. The second inequality follows by Fact 2 yielding $m_\ell \leq m_{i\times g} + m_{\ell - i\times g}$ together with $m_{\ell - i\times g} \leq m_g$ (by Fact 1 and $\ell - i \times g < g$).

The third inequality follows by $m_{i\times g} \geq m_{j^*\times g}$ ($i \geq j^*$ and Fact 1). The fourth inequality follows from (3) and the last equality by the definition of α.

This concludes the proof that for each ℓ the values $\tilde{m}_\ell$ is indeed an α-approximation (and hence a $(1+\epsilon)$-approximation) of m_ℓ.

For the time bound, we observe that the number of times in the above procedure we compute with the exhaustive linear procedure some value m_ℓ is at most $(k+1) \times n^{1/2}$ for the full intervals of size g and $n^{1-1/2}$ for the extremes of the intervals. Therefore the algorithm runs in time $O(k \times n^{3/2})$.

2.3 The last piece: a recursive argument

In the above procedure, for $g = n^{1/2}$ we first compute $m_g, m_{2g}, m_{3g}, \ldots, m_n$ in time $O(n^{3/2})$. Then, we identify (up to) $k+1$ relevant intervals. The first interval is $[m_1, m_g]$ and the other k intervals are all the ones of the form $[m_{jg}, m_{(j+1)g}]$ where $m_{(j+1)g}/m_{jg} > \alpha$. For each one of the $k+1$ relevant intervals we compute *exhaustively* all m_ℓ values inside the interval, hence in total requiring time $O(n^{3/2})$.

In order to reduce the time complexity, instead of computing all the values exhaustively in the relevant intervals we can recursively use in each interval the argument we use for the full "interval" $m_1, \ldots, m_n$. In particular, this means subdividing each relevant interval into subintervals and compute exhaustively the

m_ℓ values only at the subintervals' boundaries, but for $k+1$ relevant subintervals, where we recurse again.

Let us first consider an example giving a $(1+\epsilon)$-approximation in time $O(n^{1+1/3})$. For this we choose $g = n^{2/3}$ and compute exhaustively $m_g, m_{2g}, \ldots, m_n$, so spending in total $O(n^{1+1/3})$ time. Then, we identify $k+1$ relevant intervals just as before. Suppose that $[m_{jg}, m_{(j+1)g}]$ is one of the $k+1$ relevant intervals, i.e., $m_{(j+1)g}/m_{jg} > \alpha$. We partition this interval into $n^{1/3}$ sub-intervals each of size $n^{1/3}$. Now, we first compute m_ℓ for every sub-interval endpoint, which requires $n^{1/3} \times n = O(n^{1+1/3})$ time. Then, we compute m_ℓ exhaustively for each ℓ in the $k+1$ relevant sub-intervals, i.e., the first sub-interval and the ones for which the ratio of the values m_ℓ at the boundaries is greater than α. As done in the previous subsection, it can be easily shown that at most k sub-intervals can exists for which such condition is verified.

Overall, the number of values of ℓ for which we compute m_ℓ exhaustively are:

- the $n^{1/3}$ boundaries $g, 2g, \ldots$. This requires $O(n^{1/3})$ exhaustive computations of some m_ℓ.
- the $n^{1/3}$ boundaries of the sub-interval in the $k+1$ relevant intervals, i.e., in total $O(kn^{1/3})$ exhaustive computations of some m_ℓ.
- for each one of the $k+1$ relevant interval, we may have up to $k+1$ relevant sub-intervals and each relevant sub interval requires to compute $n^{1/3}$. In total this gives additional $O(k^2 n^{1/3})$ exhaustive computations of some m_ℓ.

Therefore, we have that the time complexity becomes $O(k^2 n^{1+1/3})$.

If we want to get $O(n^{1+1/4})$ running time we can choose $g = n^{3/4}$ and add one more level of recursion, i.e., partition the sub-intervals to sub-sub-intervals. This way the running time becomes $O(k^3 n^{1+1/4})$.

In general, given any fixed $\eta > 0$ we can set $t = \lceil 1/\eta \rceil$ and $g = n^{1-1/t}$. By using $t-1$ levels of recursion we then get a (1+epsilon) approximation in $O(k^t n^{1+1/t}) = O(k_{\epsilon,\eta} n^{1+\eta})$ time, where $k_{\epsilon,\eta}$ is a constant depending only on ϵ and η. This provides the proof of our main theorem. The algorithm is described in the pseudocode below.

3 Applying the index to the Parikh vector matching problem

We can now analyze the consequences of our result with respect to the connection between MCSP and Parikh vector membership query problem.

An immediate corollary of Theorem 1 is the following.

Corollary 1. *Let s be a binary string and for each $i = 1, \ldots, n$ let $\mu_\ell^{\min}$ (resp. $\mu_\ell^{\max}$) denote the minimum (resp. maximum) number of ones in a substring of s of lenght ℓ. For any $\epsilon, \eta \in (0,1)$, we can compute in $O(n^{1+\eta})$ approximate values $\tilde{\mu}_\ell^{\min}$ (resp. $\tilde{\mu}_\ell^{\max}$) such that*

$$\mu_\ell^{\min} \geq \tilde{\mu}_\ell^{\min} \geq (1-\epsilon)\mu_\ell^{\min} \qquad \mu_\ell^{\max} \leq \tilde{\mu}_\ell^{\max} \leq (1+\epsilon)\mu_\ell^{\max}$$

Algorithm 1 Approximate index for all maximum consecutive subsums

Input: A string s, an approximation value ϵ and a time threshold η s.t. $\epsilon, \eta \in (0, 1)$.
Output: $\tilde{m}_1, \ldots, \tilde{m}_n$ such that $m_j/\tilde{m}_j \leq \epsilon$, for each $j \in [n]$.

Set k to the minimum integer s.t. $1+\epsilon$ is not smaller than the positive real solution of $\alpha = 1 + 1/\alpha^k$.
Set $t = \lceil 1/\eta \rceil$
Recursive-Approx$(1, n, 1)$
return $\tilde{m}_1, \ldots, \tilde{m}_n$.

Recursive-Approx$(start, end, depth)$

if $depth = t - 1$ **then**
  for all $\ell = start, \ldots, end$ **do**
    compute m_ℓ exaustively and set $\tilde{m}_\ell = m_\ell$
  end for
else
  Set $size = (end - start + 1)$ and $g = size/n^{1/t}$.
  Recursive-Approx$(start, start + g, depth + 1)$
  Set $j = 2$ and $\kappa = 0$.
  while $j \leq n^{1/t}$ and $\kappa < k$ **do**
    compute $m_{start+jg}$ exaustively and set $\tilde{m}_{start+jg} = m_{start+jg}$
    if $m_{start+j\times g} > \alpha\, m_{start+(j-1)\times g}$ **then**
      Recursive-Approx$(start + (j-1)g, start + jg, depth + 1, t, k)$
      $\kappa = \kappa + 1$
    else
      for all $\ell = start + (j-1)g + 1, \ldots, start + jg - 1$ **do**
        Set $\tilde{m}_\ell = m_{start+jg}$
      end for
    end if
    $j = j + 1$
  end while
  for all $i = j, \ldots, n^{1/t}$ **do**
    compute $m_{start+ig}$ exaustively and set $\tilde{m}_{start+ig} = m_{start+ig}$
    for all $\ell = start + (i-1)g + 1, \ldots, start + ig - 1$ **do**
      Set $\tilde{m}_\ell = m_{start+ig}$
    end for
  end for
end if

Let $\mathbf{s}$ be a binary string of length n. Fix a tolerance threshold $\epsilon > 0$, and let $\tilde{\mu}_\ell^{\min}$ and $\tilde{\mu}_\ell^{\max}$ $(\ell = 1, \ldots, n)$ be as in Corollary 1. In terms of Parikh vector membership queries, we have the following necessary condition for the occurrence of a Parikh vector in the string $\mathbf{s}$.

Corollary 2. *For any $p = (x_0, x_1)$ such that there exists a substring of $\mathbf{s}$ whose Parikh vector equals p, we have that*

$$\tilde{\mu}_{x_0+x_1}^{\min} \leq x_1 \leq \tilde{\mu}_{x_0+x_1}^{\max}.$$

We also have an "almost matching" sufficient condition, which also follows from Lemma 1 and Corollary 1.

Corollary 3. *Fix a Parikh vector $p = (x_0, x_1)$. If*

$$\frac{\tilde{\mu}^{\min}_{x_0+x_1}}{1-\epsilon} \leq x_1 \leq \frac{\tilde{\mu}^{\max}_{x_0+x_1}}{1+\epsilon}$$

then p occurs in **s**.

As a consequence, if we use the values $\tilde{\mu}^{\min}$ and $\tilde{\mu}^{\max}$ we can answer correctly to any membership query involving a Parikh vector which occurs in s. Moreover, we can also answer correctly any membership query involving a Parikh vector satisfying the condition in Corollary 3. In contrast, it might be that the approximate index makes us report false positives, when the membership query is about a Parikh vectors $p = (x_0, x_1)$ such that

$$\tilde{\mu}^{\min}_{x_0+x_1} \leq x_1 \leq \frac{\tilde{\mu}^{\min}_{x_0+x_1}}{1-\epsilon} \quad \text{or} \quad \frac{\tilde{\mu}^{\max}_{x_0+x_1}}{1+\epsilon} \leq x_1 \leq \tilde{\mu}^{\max}_{x_0+x_1}.$$

4 Some final observations and open problems

We presented a novel approach to to approximating all maximum consecutive subsums of a sequence of non-negative integers in time $O(n^{1+\eta})$, for any constant $\eta > 0$. This can be directly used for obtaining a linear size index for binary string, which allows to answer in constant time Parikh vector membership queries. The existence of a $o(n^2)$ time solution for the exact case remains open. Some observations are in order regarding our approach.

A first observation is that we do not need to store $\tilde{m}_j$ for each $j = 1, \ldots, n$ as it is sufficient to store only the distinct values computed exhaustively. This brings down also the space complexity of our approximate index to $O(n^{\epsilon})$.

The computation of the approximate index is extremely easy and fast to implement, and, in contrast to the previous *exact* solution present in the literature [9, 8, 17], it does not require any tabulation or convolution computation. In order to get the flavor of the quantities involved, notice that for $k = 30$ we can already guarantee an approximation ratio $\alpha = 1.085$ and with $k = 500$, we get $\alpha = 1.009$.

With respect to the Parikh vector indexing problem, one can clearly modify the algorithm in order to store together with the value $\tilde{m}_k$ also the position where the corresponding maximizing substring occurs. This way the algorithm can report a substring whose Parikh vector is an $(1 + \epsilon)$-approximation of the desired Parikh vector. Moreover, this can also serve as a starting point for examining, in time proportional to ϵk the surrounding part of s in order to check whether the reporting position actually leads to an exact match. Along this line, an alternative is to associate to each $\tilde{m}_s$ both $m_{(j-1)\times g}$ and $m_{j\times g}$ and the corresponding positions.

It would be interesting to investigate whether it is possible to obtain Las Vegas algorithms for the Parikh vector problem based on our approximation perspective. Another direction for future investigation regards the extension of our approach to cover the case of Parikh vector membership queries in strings over non-binary alphabets.

References

1. A. Amir, A. Apostolico, G. M. Landau, and G. Satta. Efficient text fingerprinting via Parikh mapping. *J. Discrete Algorithms*, 1(5-6):409–421, 2003.
2. L. Babai and P. F. Felzenszwalb. Computing rank-convolutions with a mask. *ACM Trans. Algorithms*, 6(1):1–13, 2009.
3. G. Benson. Composition alignment. In *Proc. of the 3rd International Workshop on Algorithms in Bioinformatics (WABI'03)*, pages 447–461, 2003.
4. A. Bergkvist and P. Damaschke. Fast algorithms for finding disjoint subsequences with extremal densities. *Pattern Recognition*, 39:2281–2292, 2006.
5. S. Böcker. Sequencing from compomers: Using mass spectrometry for DNA de novo sequencing of 200+ nt. *Journal of Computational Biology*, 11(6):1110–1134, 2004.
6. S. Böcker and Zs. Lipták. A fast and simple algorithm for the Money Changing Problem. *Algorithmica*, 48(4):413–432, 2007.
7. D. Bremner, T. M. Chan, E. D. Demaine, J. Erickson, F. Hurtado, J. Iacono, S. Langerman, and P. Taslakian. Necklaces, convolutions, and $X + Y$. In *14th Annual European Symposium on Algorithms (ESA'06)*, pages 160–171, 2006.
8. P. Burcsi, F. Cicalese, G. Fici, and Zs. Lipták. On Approximate Jumbled Pattern Matching. *Theory of Computing Systems*, 50(1):35-51, 2012.
9. P. Burcsi, F. Cicalese, G. Fici, and Zs. Lipták. Algorithms for Jumbled Pattern Matching in Strings. In *5th International Conference FUN with Algorithms (FUN 2010)*, pages 89-101.
10. A. Butman, R. Eres, and G. M. Landau. Scaled and permuted string matching. *Inf. Process. Lett.*, 92(6):293–297, 2004.
11. T. M. Chan. All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. *Algorithmica*, 50(2):236–243, 2008.
12. Y.H. Chen, H.I. Lu, C.Y. Tang. Disjoint segments with maximum density. In *Proc. of the International Workshop on Bioinformatics Research and Applications (IWBRA 2005)*, LNCS 3515, pages 845–850, 2005.
13. F. Cicalese, G. Fici, and Zs. Lipták. Searching for jumbled patterns in strings. In *Proc. of the Prague Stringology Conference 2009 (PSC'09)*, pages 105–117, 2009.
14. M. Cieliebak, T. Erlebach, Zs. Lipták, J. Stoye, and E. Welzl. Algorithmic complexity of protein identification: Combinatorics of weighted strings. *Discrete Applied Mathematics*, 137(1):27–46, 2004.
15. R. Eres, G. M. Landau, and L. Parida. Permutation pattern discovery in biosequences. *Journal of Computational Biology*, 11(6):1050–1060, 2004.
16. P. Jokinen, J. Tarhio, and E. Ukkonen. A comparison of approximate string matching algorithms. *Software Practice and Experience*, 26(12):1439–1458, 1996.
17. T.M. Moosa and M.S. Rahman. Sub-quadratic time and linear size data structures for permutation matching in binary strings. *J. Discrete Algorithms*, 10(1):5–9, 2012.
18. L. Parida. Gapped permutation patterns for comparative genomics. In *Proc. of WABI 2006*, pages 376–387, 2006.