On lower bounds for the Maximum Consecutive Subsums Problem and the (min,+)-convolution

Eduardo S. Laber and Wilfredo Bardales R.
Dept. Comp. Sc., PUC-Rio, Brazil
Email: {laber, wroncalla}@inf.puc-rio.br

Ferdinando Cicalese
Dept. Comp. Sc., Univ. Salerno, Italy
Email: cicalese@dia.unisa.it

***Abstract*—Given a sequence of n numbers, the MAXIMUM CONSECUTIVE SUBSUMS PROBLEM (MCSP) asks for the maximum consecutive sum of lengths ℓ for each $\ell = 1, \ldots, n$. No algorithm is known for this problem which is significantly better than the naive quadratic solution. Nor a super linear lower bound is known. The best known bound for the MCSP is based on the the computation of the $(min, +)$-convolution, another problem for which neither an $O(n^{2-\epsilon})$ upper bound is known nor a super linear lower bound. We show that the two problems are in fact computationally equivalent by providing linear reductions between them. Then, we concentrate on the problem of finding super linear lower bounds and provide empirical evidence for our conjecture that the solution of both problems requires $\Omega(n \log n)$ time in the decision tree model.**

I. INTRODUCTION

There exists a long line of research on identifying or selecting maximal consecutive subsums in a numerical sequence $A = (a_1, \ldots, a_n)$. The most classical example is perhaps the MAXIMUM SUM SEGMENT problem (MSS) asking for the subsequence attaining the maximum among all possible $\binom{n}{2} + n$ subsequences. The problem was introduced by Grenader [4] and has found application in pattern matching [20], biological sequence analysis [1], and data mining [18]. The MSS can be solved in linear time using Kadane's algorithm [4].

The k MAXIMUM SUM SEGMENTS (k-MSS), asking to find the k largest consecutive subsums in a sequence, has also been investigated [3], [11], [24] and eventually proved to be solvable in $O(n + k)$ in [7]. A linear time algorithm exists also for the case where the maximal disjoint segments of maximum sum have to be found [28], and for the variants of MSS and k-MSS where only segments whose length is within a given interval have to be considered.

All these problems admit a linear solution. In contrast, notwithstanding the apparent similarity, the following variant known as MAXIMUM CONSECUTIVE SUBSUMS PROBLEM has defied so far any attempt to provide a strongly subquadratic solution.

Given a sequence $A = (a_1, a_2, \ldots, a_n)$ of n numbers, the MAXIMUM CONSECUTIVE SUBSUMS PROBLEM (MCSP) asks to compute *for all* $\ell = 1, \ldots, n$, the maximum consecutive subsum of length ℓ, i.e., the sequence $m_1, \ldots, m_n$, where

$$m_\ell = \max_{i=1,\ldots n-\ell+1} a_i + \cdots + a_{i+\ell-1}. \quad (1)$$

The MCSP appears in several scenarios of both theoretical and practical interest like approximate pattern matching [8], mass spectrometry data analysis [14], and in the problem of locating large empty regions in data sets [5]. Most work has been done for the case where the input sequence is binary, since in this case the MCSP coincides with the problem of constructing membership query indexes for jumbled pattern matching [8], [9], [2], [19].

It is not difficult to come up with simple $O(n^2)$ solutions for the MCSP since each value m_ℓ in (1) can be easily computed in linear time by one pass over the input sequence. Surprisingly, despite the growing interest generated by this problem (see, e.g., [10], [8], [9], [12], [13], [26], [27], and references therein quoted), no solution is known with running time $O(n^{2-\epsilon})$ for some constant $\epsilon > 0$, nor is a lower bound better than the trivial $\Omega(n)$ known.

When the sequence compresses well, a better complexity can be achieved for the MCSP on 0/1 sequences [2], [19]. Algorithms that produce approximate solutions for the MCSP are also known [13]. However, for the general case, the best available algorithm in the real RAM model runs in $O(n^2/\log n)$ [10], [26], [27] and makes use of the algorithm proposed in [6] for computing a $(\min, +)$-convolution.

The $(min, +)$-convolution problem is a natural variation of the classical convolution problem: Given two sequences $X = (x_0, x_1, \ldots, x_n)$ and $Y = (y_0, y_1, \ldots, y_n)$ of real numbers, the $(min, +)$-convolution of X and Y is the sequence $z_k = \min_{i=0,\ldots,k} \{x_i + y_{k-i}\}$, for $k = 0, \ldots, 2n$.

$(min, +)$-convolution has important applications in a variety of areas, including signal processing, pattern recognition, computer vision, and mathematical programming. According to [6], this problem has appeared frequently in the literature since Bellman's early work on dynamic programming. Like for the MCSP, no strongly subquadratic algorithm appears to be known to compute the min-convolution. The best known algorithm for computing $(min, +)$-convolution runs in $O(n^2/\log n)$ [6].[1]

Taking into account the apparent difficulty to devise an $O(n^{2-\epsilon})$ algorithm for the MCSP and for computing the $(min, +)$-convolution, a natural question to ask is whether there exists a non-trivial lower bound for these problems. This work describes our findings in the quest for a super-linear

[1]Very recently, Williams [29] announced the computation of (min,+)-convolution in $O(n^2/2^{(\log n/\log\log n)^{1/2}})$, which is better than there result of [6] by more than any polylog factor, though, it is still $\omega(n^{2-\epsilon})$ for any ϵ.

lower bound in the decision tree model of computation.

Our Contributions. We provide computational evidence that the running time of both MCSP and (min,+)-convolution problem is $\Omega(n \log n)$ in the decision tree model of computation.

We start by showing linear reductions between the two problems. As a result of this equivalence, any bound for one problem also holds for the other. Then, in the following, we only concentrate on the MCSP. In Section III, we argue that a lower bound for the MCSP in the decision tree model can be obtained by generating a large set of inputs sequences such that no pair of them produce a common output. We prove constructively that there exists such a sets of exponential size, although, this is still not enough to prove a superlinear lower bound on MCSP.

In Section IV, by using a deterministic approach we show empirically that for $n \leq 14$ there exists a set of inputs, with the above property, and cardinality larger than $(n/2)!$ so that $n/2\log(n/2)$ is a lower bound on the depth of any decision tree that solves the MCSP for instances of size $n \leq 14$. This required 27 hours of CPU time in our computational environment. In order to address larger values of n, we employed sampling strategies. We devised a hypothesis test and showed that the $n/2\log(n/2)$ lower bound also holds for any $n \leq 100$ with confidence much larger than 99.999%.

We believe that our results bring new insight on the complexity of both the MCSP and the (min,+)-convolution problem and represent a significant initial step towards proving a superlinear lower bound for this problems. Moreover, the techniques employed might be useful in the investigation of lower bounds for other computational problems with the same flavor.

II. MCSP and the $(min,+)$-Convolution Problem have the same complexity

We will show a linear time reduction between the MCSP and the $(min,+)$-convolution and vice versa.

Let $A = (a_1, \ldots, a_n)$ be an input sequence for the MCSP and let $I(A) = (X, Y)$ be the instance for the $(min,+)$-convolution problem defined by $x_0 = y_n = 0$ and $x_i = -\sum_{k=1}^{i} a_k$ and $y_{n-i} = \sum_{k=1}^{i} a_k$, for $i = 1, \ldots, n$. It is easy to verify that p_k is the starting position of a maximum consecutive sum of length k in A if and only if $z_{n+k} = x_{p_k+k-1} + y_{n-p_k+1}$.

Vice versa, let $I = (X, Y)$ be an input for the $(min,+)$-convolution, where $X = (x_0, \ldots, x_n)$ and $Y = (y_0, \ldots, y_n)$. Let S be a large enough number and define the input sequence $A = (a_1, \ldots, a_{2n+4})$ for the for the MCSPas follows: $a_{n+1} = a_{n+4} = S$; $a_{n+3} = -y_0$; $a_{n+2} = -x_0$; for each $i < n+1$ set $a_i = x_{n-i} - x_{n+1-i}$ and for $i > n+4$ set $a_i = y_{i-n-5} - y_{i-n-4}$.

Denoting by $z_0, \ldots, z_2 n - 1$ the result of the convolution of X and Y, one can verify that

$$z_k = \left(\sum_{j=p_k}^{p_k+k+1} a_i \right) - 2S,$$

where p_k is the starting position of the subsequence in A of length $k+2$ which achieves the maximum sum.

Due to these linear time reductions we can conclude that both problems have the same time complexity. In the following, we will focus our discussion on lower bounds for MCSP and any conclusion reached will also hold for the $(min,+)$-convolution.

III. Towards a Lower Bound for the MCSP

In this section we discuss our approach to prove a lower bound for the MCSP. It will be convenient to employ the following alternative formulation of the MCSP

Input. A sequence $A = (a_1, \ldots, a_n)$ of n real numbers;

Output. A sequence $P = (p_1, \ldots, p_n)$, where p_ℓ, for $\ell = 1, \ldots, n$, is the starting position of a consecutive subsequence of A that has maximum sum among the consecutive subsequences of A with length ℓ.

Using this representation, the subsequence $A[p_i \ldots p_i + i - 1]$ is a maximum consecutive subsum of length i. We call the sequence P an *output configuration* or simply a *configuration*.

For example, for the input sequence $A = (3, 0, 5, 0, 2, 4)$ the only output configuration is given by the sequence $P = (5, 5, 1, 3, 2, 1)$, which says, e.g., that there is a maximum consecutive subsum of length 4 starting at position 3.

We note that there are $n!$ possible configurations because p_i, for $i = 1, \ldots, n$, can assume any value in the set $\{1, \ldots, n - i + 1\}$. In particular for the input $A = (a_1, \ldots, a_n)$, with $a_1 = a_2 = \cdots = a_n = 1$, all the $n!$ possible configurations are output configurations for A.

A. An approach based on unique configurations

The previous example shows that some input sequences have more than one output configuration.

Let $\mathcal{A} = \{A_1, \ldots, A_k\}$ be an arbitrarily chosen set of inputs for the MCSP and let $\mathcal{P}(A) = \{P \mid P \text{ is an output configuration for } A\}$.

We say that a configuration P is *unique* if and only if there exists an input A for the MCSP for which $\mathcal{P}(A) = \{P\}$.

Our approach to prove a lower bound on MCSP consists on finding a large set of distinct unique configurations.

In fact, let $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ be a set of distinct unique configurations. Moreover, let A_i, for $i = 1, \ldots, k$, be an instance for MCSP such that P_i is its unique configuration. Then, the instances in the set $\mathcal{A} = \{A_1, \ldots, A_k\}$ are distinct. In order to be able to solve the instances in $\mathcal{A}$ an algorithm must be able to uniquely distinguish the configurations in $\mathcal{P}$. This requires at least $\lceil \log k \rceil$ bits of information. Hence, $\lceil \lg k \rceil$ is a lower bound to the problem in the decision tree model. In fact, the above considerations shows that we need at least that number of steps, or decisions, to uniquely identify any solution from the set $\mathcal{P}$ using the decision tree model. We have proved the following.

Theorem 1: If $\mathcal{P}$ is a subset of distinct unique configurations, then $\log |\mathcal{P}|$ is a lower bound for the running time of the MCSP in the decision tree model.

We shall observe that if every configuration were unique, then we could prove a lower bound of $\Omega(n \log n)$ since there exists $n!$ configurations of size n. However, there are configurations that are not unique like the configuration $P = (1, 2, 1)$. In fact, assume that $A = (a_1, a_2, a_3)$ is an input for which P is its unique configuration. Then, we would have both $a_1 > a_3$ and $a_2 + a_3 > a_1 + a_2$, which is not possible.

The following result shows that the number of unique configurations is indeed very large.

Theorem 2: There exist $\Omega\left(2^{\frac{n}{2}}\right)$ unique configurations.

Proof: Fix a subset $S \subseteq \left\{\left\lceil \frac{n}{2} \right\rceil + 1, \ldots, n\right\}$ and define an input $A^S = (a_1^S, \ldots, a_n^S)$ as follows:

$$a_i^S = \begin{cases} 0 & \text{if } 1 \leq i < \lceil n/2 \rceil - 1 \\ 2 & \text{if } i = \lceil n/2 \rceil - 1 \\ 4n & \text{if } i = \lceil n/2 \rceil \\ 3 & \text{if } \lceil n/2 \rceil < i \leq n \text{ and } i \notin S \\ 1 & \text{if } \lceil n/2 \rceil < i \leq n \text{ and } i \in S \end{cases}$$

It is not hard to see that the unique output for A^S is the configuration $P^S = (p_1^S, \ldots, p_n^S)$ given by

$$p_j^S = \begin{cases} \lceil n/2 \rceil & \text{if } j \leq \lceil n/2 \rceil \text{ and } (j + \lceil n/2 \rceil - 1) \notin S \\ \left\lceil \frac{n}{2} - 1 \right\rceil & \text{if } j \leq \lceil n/2 \rceil \text{ and } (j + \lceil n/2 \rceil - 1) \in S \\ n - j + 1 & \text{if } \lceil n/2 \rceil < j \leq n \end{cases}$$

If $\lceil n/2 \rceil < j \leq n$ it is easy to realize that the maximum sum of length j in A^S has to start at the rightmost feasible position so that $p_j^S = n - j + 1$.

If $j \leq \left\lceil \frac{n}{2} \right\rceil$ we claim that the maximum sum of length j either starts at position $\left\lceil \frac{n}{2} \right\rceil$ or at position $\left\lceil \frac{n}{2} \right\rceil - 1$ of A^S. In fact, if $p_j^S > \left\lceil \frac{n}{2} \right\rceil$ then we would have the maximum sum of length j smaller than $3n$ which is not possible because the value of A^S at position $\left\lceil \frac{n}{2} \right\rceil$ is $4n$. Moreover, we cannot have $p_j^S < \left\lceil \frac{n}{2} \right\rceil - 1$ for otherwise the sum of the subsequence of length j starting at position $p_j^S + 1$ of A^S would be larger than the maximum sum. Hence, our claim is correct.

To decide whether the maximum of length j starts at position $\left\lceil \frac{n}{2} \right\rceil$ or at position $\left\lceil \frac{n}{2} \right\rceil - 1$ we just need to compare the value of A^S at position $\left\lceil \frac{n}{2} \right\rceil + j - 1$ (which is either 1 or 3) with that of A^S at position $\left\lceil \frac{n}{2} \right\rceil - 1$ (which is 2). If the former is larger, what happens when $\left\lceil \frac{n}{2} \right\rceil + j - 1 \notin S$, then we must have $p_j^S = \left\lceil \frac{n}{2} \right\rceil$. Otherwise, we must have $p_j^S = \left\lceil \frac{n}{2} \right\rceil - 1$.

Clearly, for any two distinct $S_1, S_2 \subseteq \{\lceil n/2 \rceil + 1, \ldots, n\}$ we have $P^{S_1} \neq P^{S_2}$. Thus the set $\left\{P^S | S \subseteq \{\lceil n/2 \rceil + 1, \ldots, n\}\right\}$ is a set of unique configurations of cardinality $2^{n/2}$, which establishes or result. ■

The existence of at least exponentially many (in n) configurations supports our approach and is an indication of its potential. However, this result combined with Theorem 1 is still not enough to obtain a non trivial (superlinear) lower bound for the MCSP.

This motivated us to enrich our analysis by empirically exploring the number of unique configurations.

IV. Empirical evidences that MCSP requires $\Omega(n \log n)$ time

In order to count the number of unique configurations it is important to decide whether a given configuration P is unique or not. For that we test whether there exists an input sequence $A = (a_1, \ldots, a_n)$ for which P is its unique output configuration. For $i \in \{1, \ldots, n-1\}$ and a configuration P, let $\mathcal{Q}(P, i)$ be the following set of inequalities:

$$\sum_{k=p_i}^{p_i+i-1} a_k > \sum_{k=j}^{j+i-1} a_k \qquad \text{for } j = 1, \ldots, n-i+1 \text{ and } j \neq p_i$$

It is easy to realize that the contiguous subsequence of length i that starts at position p_i of A has sum larger than the sum of any other contiguous subsequence of length i if and only if the point $A = (a_1, \ldots, a_n) \in R^n$ satisfies the above set of inequalities. Thus, P is a unique configuration if and only if the set of inequalities $\mathcal{Q}(P, 1) \cup \mathcal{Q}(P, 2) \cup \cdots \cup \mathcal{Q}(P, n-1)$ has a feasible solution. In our experiments we employed a linear programming solver to perform this verification.

In order to speed up our computation we also employ a sufficient condition for the non-uniqueness of a configuration, which is provided by the following proposition whose proof is deferred to the appendix.

Proposition 1: Let $P = (p_1, \ldots, p_n)$ be an output configuration. If there exist $1 \leq i < j \leq n$ such that $p_j = p_i + i$, then P is not unique.

For instance, consider the configuration $P = (5, 1, 3, 4, 1)$. By taking $i = 2$ and $j = 3$, we have $p_i + i = p_j$ and, can conclude that P is not unique, as can be easily verified.

Unfortunately, this non-adjacency property does not completely characterize the set of unique configurations because there exist configurations with no adjacent maximums that are not unique. For example, the configuration $P = (2, 4, 2, 1, 2, 1)$ has no adjacent maximums and is not unique. In fact, if $A = (a_1, \ldots, a_6)$ is an input sequence for which P is its unique configuration then we must have simultaneously: (i) $a_2 > a_4$ because of $p_1 = 2$; (ii) $a_4 + a_5 > a_1 + a_2$ because of $p_2 = 4$; and (iii) $a_1 + a_2 + a_3 + a_4 > a_2 + a_3 + a_4 + a_5$ because of $p_4 = 1$. However, this is impossible, since by summing the first two inequality and adding a_3 on both sides, we obtain a contradiction to the third inequality.

Nonetheless, we can effectively use the condition to speed up our algorithms.

Algorithm 1 shows the pseudo-code of our method that recursively constructs all unique configurations of length n. When DETERMINISTIC-COUNTING(P, i) is called, the values of the positions $1, 2, \ldots, i-1$ of the configuration P, under construction, are already fixed. Then, for each $j \in \{1, \ldots, n-i+1\}$, the procedure ISFEASIBLE(P, i, j), explained below, is called to verify whether it is possible to extend the partial configuration P by setting the value of position i to j. If this is the case, the algorithm set $p_i = j$ and it recursively calls DETERMINISTIC-COUNTING$(P, i+1)$ to construct all unique configurations that are extensions of P. As soon as the algorithm set the values of all positions of P, it increments the global variable *count* of unique configurations. The procedure has to be initially called with $i = 1$ and the variable *count* shoud be initially set to 0.

The procedure IS-FEASIBLE(P, i, j) first verifies if the subsequence of A starting at position j is adjacent to some maximum subsequence that has already been fixed. This test can be done in $O(1)$ time by using a suitable data structure. If this test is positive it rules out j as a value for p_i due to Proposition 1. Otherwise, the procedure verifies whether the set of inequalities $Q(P,1) \cup \cdots \cup Q(P,i)$ is feasible and it returns TRUE or FALSE, accordingly.

Algorithm 1 DETEMINISTIC-COUNTING(P, i)

```
1: if i = n then
2:     count ← count + 1
3:     return
4: end if
5: for j ← 1 to n − i + 1 do
6:     if ISFEASIBLE(P, i, j) then
7:         p_i ← j
8:         DETERMINISTIC-COUNTING(P, i + 1)
9:     end if
10: end for
```

Algorithm 2 ISFEASIBLE(P, i, j)

```
1: if the subsequence of length i starting at position j is adajcent to
   the subsequence of length k starting at p_k for some k < i then
2:     return FALSE
3: else
4:     p_i = j    % this is only to verify if this extension if feasible.
5:     if the set of inequalities Q(P,1) ∪ ··· ∪ Q(P,i)is feasible
6:       then return TRUE
7:       else return FALSE
8: end if
```

Table I presents the results obtained by the deterministic approach. We were able to determine the number of unique configurations up to $n = 14$. The results suggest a super exponential growth. Indeed, notice the growth of the ratio between the number of unique configurations and $(n/2)!$.

All the executions required 27 hours of CPU time under the following hardware and software specifications: Main Hardware Platform: Intel® Core™ i7 3960X, 3.30GHz CPU, 32GB RAM, 64-bit; OS: Windows 7 Professional x64; Compiler: Microsoft®Visual C# 2010 Compiler version 4.0.30319.1; Solver: Gurobi Optimizer.

TABLE I
THE NUMBER OF UNIQUE CONFIGURATIONS FOR $n = 1, \ldots, 14$ COMPARED TO THE VALUE $n/2!$.

n	U(n) = Nº Unique Config.	$\frac{n}{2}!$	Ratio $\frac{U(n)}{(n/2)!}$
1	1	0.8	1.25x
2	2	1.0	2.00x
3	4	1.3	3.07x
4	12	2.0	6.00x
5	36	3.3	10.90x
6	148	6.0	24.66x
7	586	11.6	50.51x
8	2,790	24.0	116.25x
9	13,338	52.3	255.02x
10	71,562	120.0	596.35x
11	378,024	287.9	1,313.03x
12	2,222,536	720.0	3,086.85x
13	12,770,406	1,871.3	6,824.34x
14	78,968,306	5,040.0	15,668.31x

In order to extend our analysis to larger instances we then employed a probabilistic approach.

A. A Probabilistic Approach

The first idea for estimating the number of unique configurations is to sample a large number M of configurations and then employ Procedure IS-FEASIBLE to decide whether each of them is unique or not. The number of unique configurations found over M is an unbiased estimator for the number of unique configurations. With this approach we managed to obtain strong evidence of the super linear lower bound for n up to 28. To extend our range we followed a different approach.

In the deterministic case, we explore the configuration space via a depth first search over the back-tracking tree of all possible configurations. In our probabilistic approach, presented in Algorithm 3, we randomly traverse a path in the back-tracking tree that corresponds to a unique configuration. Assume that we have already fixed the values for the positions $1, 2, \ldots, i-1$ of the configuration P that is under construction. Then, in order to set the value of p_i, we construct a list S of all values $j \in \{1, \ldots, n-i+1\}$ such that ISFEASIBLE(P, i, j) returns TRUE. Let $b_i = |S|$ be the *branching factor* of our path at level i. Then, we randomly choose one of the values in S for p_i and continue to set the values of p_j for $j > i$; if the method observes the branching factors $b_1, b_2, \ldots, b_n$, in a root to leaf path, then it outputs $X = \prod_{i=1}^{n} b_i$. as a guess for the number of unique configurations.

The value X can be used to estimate the number of unique configurations because X is a sample of a random variable $\mathbf{X}$ whose expected value $\mathrm{E}[\mathbf{X}]$ is equal to the number of the unique configurations. In fact, let ℓ be a leaf located at depth n of the backtracking tree, that is, ℓ corresponds to a unique configuration. The probability of reaching ℓ in our random walk is $1/B(\ell)$, where $B(\ell)$ is the product of the branching factors in the path from the root of the tree to ℓ. In addition,

Algorithm 3 BRANCHING-PRODUCT(n)

```
X ← 1
for i ← 1 to n do
    S ← ∅
    for j ← 1 to n − i + 1 do
        if IsFeasible(P, i, j) then
            Add j to the list S of possible branchings
        end if
    end for
    if S is empty then
        return 0
    else
        X ← X × |S|
        p_i ← value randomly selected from list S
    end if
end for
return X
```

if ℓ is reached, the method outputs $B(\ell)$. Let L be the set of leaves located at level n in the backtracking tree. Thus we have that

$$\mathrm{E}[\mathbf{X}] = \sum_{\ell \in L} \frac{1}{B(\ell)} \times B(\ell) = |L|.$$

After coming up with this approach, we found out that it had already been proposed to study heuristics for backtracking algorithms [23], [22].

We do not use directly the observed value X to estimate the number of unique configurations; instead, we assume that $\mathrm{E}[\mathbf{X}]$, the number of unique configurations, is smaller than or equal to $(\frac{n}{2}!)$ and use the sampled value of $\mathbf{X}$ to reject this hypothesis with some level of confidence.

Under the hypothesis that $\mathrm{E}[\mathbf{X}] \leq (\frac{n}{2})!$, using Markov's inequality [25] it follows that:

$$\Pr\left[\mathbf{X} \geq c\frac{n}{2}!\right] \leq \Pr[X \geq c\,\mathrm{E}[\mathbf{X}]] \leq \frac{1}{c}$$

which implies that $\Pr\left[\mathbf{X} < c\frac{n}{2}!\right] \geq 1 - \frac{1}{c}$.

Therefore, if we sample $\mathbf{X}$ and find a value larger than $c\frac{n}{2}!$, we can reject the hypothesis and conclude that the number of unique configurations is $> \frac{n}{2}!$ with confidence of $1 - \frac{1}{c}$.

We can extend this approach by taking the maximum of k samples. Let $X_1, \ldots, X_k$ be the values for k samples of the random variable $\mathbf{X}$. Then, with the hypothesis $\mathrm{E}[\mathbf{X}] \leq \frac{n}{2}!$ and using the above inequality we have

$$\Pr\left[\max\{X_1, \ldots, X_k\} < c\frac{n}{2}!\right] = \Pr\left[\bigwedge_{i=1}^{k}\left(X_i < c\frac{n}{2}!\right)\right]$$
$$= \prod_{i=1}^{k} \Pr\left[X_i < c\frac{n}{2}!\right] \geq \left(1 - \frac{1}{c}\right)^k.$$

In other words, assuming that $\mathrm{E}[\mathbf{X}] \leq \frac{n}{2}!$, and using the $\max\{X_1, X_2, \ldots, X_k\}$, the probability of all the samples being less or equal than $c\frac{n}{2}!$, for a large enough c, is very high, and if in those k values we find a value greater or equal than $c\frac{n}{2}!$, then we can reject the hypothesis and conclude that $\mathrm{E}[\mathbf{X}] \geq \frac{n}{2}!$ with confidence $(1 - 1/c)^k$.

TABLE II
$\Pr\left[\max\{X_1, \ldots, X_k\} < c_n\frac{n}{2}!\right]$ FOR $k = 1{,}000$

n	c_n	$\Pr[\max_1^k\{X_i\} < c_n\frac{n}{2}!]$
10	6,048	99.98346560846560%
11	23,760	99.99579124579120%
12	38,880	99.99742798353910%
13	439,296	99.99977236305360%
14	558,835	99.99982105636870%
20	372,252,672	99.99999973136530%
30	102,827,922,078	99.99999999902750%
40	4,680,410,711,674	99.99999999997860%
50	69,590,788,615,385	99.99999999999860%
60	562,841,769,233,371	99.99999999999980%
70	136,904,322,455,757	99.99999999999930%
80	87,399,891,508,924	99.99999999999890%
90	73,279,283,017	99.99999999863540%
100	204,252,401	99.99999951040970%

For example, for $n = 10$ and $k = 10$, suppose that $\max\{X_1, \ldots, X_{10}\} = 378000$ which is equal to $3{,}150\frac{10}{2}!$. In this case,

$$\Pr\left[\max\{X_1, \ldots, X_{10}\} < c\frac{10}{2}!\right] \geq \left(1 - \frac{1}{3{,}150}\right)^{10} \approx 99.68\%$$

This implies that we can reject the hypothesis with a confidence of 99.68% because we've found a value of $3{,}150\frac{10}{2}!$. But, if the number of samples were 1000 and the maximum remains the same, the confidence level would drop to 72.79%, which clearly shows the trade-off between the number values sampled and the confidence level.

In our experiments we sampled 1000 values of $\mathbf{X}$ for different values of n. Let $max(n, 1000)$ be the maximum value found in the $1{,}000$ samples and let $c_n = \lfloor max(n, 1000)/\frac{n}{2}!\rfloor$. Table II shows the probability of $\Pr[\max\{X_1, \ldots, X_{1000}\} < c_n\frac{n}{2}!]$ assuming that $\mathrm{E}[\mathbf{X}] \leq \frac{n}{2}!$ for configurations up to size $n = 100$. This probability also expresses our confidence to reject the hypothesis because in fact we've found a value greater or equal than $c_n\frac{n}{2}!$. Based on these results we state the following conjecture.

Conjecture 1: The running time of MCSP is $\Omega(n \log n)$ in the decision tree model.

V. CONCLUSIONS

We have described an approach to prove lower bounds for the MCSP and the $(min, +)$-convolution problem in the decision tree model. Using this approach we gave probabilistic evidence supporting the conjecture that both problems require $\Omega(n \log n)$ computational steps in the decision tree model of computation.

Moreover, we believe that the techniques employed here can be of independent interest in investigating theoretical lower bounds on other related computational problems.

REFERENCES

[1] L. Allison, Longest biased interval and longest non-negative sum interval, Bioinformatics 19(10),1294-1295, 2003.

[2] G. Badkobeh, G. Fici, S. Kroon, Z. Lipták, Binary jumbled string matching for highly run-length compressible texts, IPL 113:604-608,2013.

[3] S.E. Bae and T.Takaoka, Improved algorithms for the k-maximum subarray problem, Comput. J. 49(3), 358-374, 2006.

[4] J. L. Bentley, Programming pearls (Addison-Wesley, 1986).
[5] A. Bergkvist, P. Damaschke, Fast algorithms for finding disjoint subsequences with extremal densities, Proc. ISAAC'05, LNCS 3827, 714-723.
[6] D. Bremner, T.M. Chan, E.D. Demaine, J. Erickson, F. Hurtado, J. Iacono, S. Langerman, M. Patrascu, P. Taslakian, Necklaces, convolutions, and x+y. CoRR, abs/1212.4771, 2012. See also Proc. of ESA'06, LNCS 4168, pp. 160-171, 2006.
[7] G. S. Brodal and A. G. Jorgensen, A linear time algorithm for the k maximal sums problem, Proc. of MFCS07, pp. 442-453, 2007.
[8] P. Burcsi, F. Cicalese, G. Fici and Zs. Lipták, Algorithms for jumbled pattern matching in strings, IJFCS, 23, 357-374, 2012.
[9] P. Burcsi, F. Cicalese, G. Fici and Zs. Lipták, On approximate jumbled pattern matching in strings, Th. of Comp. Systems 50(1), 35-51, 2012.
[10] P. Burcsi, F. Cicalese, G.Fici and Zs. Lipták, On table arrangements, scrabble freaks, and jumbled pattern matching, Proc. of Fun with Algorithms, LNCS 6099, pp. 89-101, 2010.
[11] C.-H. Cheng, K.-Y. Chen, W.-C. Tien and K.-M. Chao, Improved algorithms for the *k* maximum-sums problems, Proc. of ISAAC05, LNCS 3827, pp. 799-808.
[12] F. Cicalese, G. Fici and Zs. Lipták, Searching for jumbled patterns in strings, Proc. of the Prague Stringology Conference, pp. 105-117, 2009.
[13] F. Cicalese, E. Laber, O. Weimann and R. Yuster, Near linear time construction of an approximate index for all maximum consecutive subsums of a sequence, Proc. of CPM2012, LNCS 7354, pp. 149158, 2012.
[14] M. Cieliebak, T. Erlebach, Z. Lipták, J. Stoye and E. Welzl, Algorithmic complexity of protein identification: combinatorics of weighted strings, Discrete Applied Mathematics 137(1), 27-46, 2004.
[15] D. Eppstein, Efficient algorithms for sequence analysis with concave and convex gap costs, PhD thesis, Comp. Sc. Dept., Columbia Univ., 1989.
[16] T.H. Fan, S. Lee, H.I. Lu, T.S. Tsou, T.C. Wang and A. Yao, An optimal algorithm for maximum-sum segment and its application in bioinformatics, Proc. of CIAA03, pp. 251-257, 2003.
[17] P.F. Felzenszwalb, D.P. Huttenlocher and J.M. Kleinberg, Fast algorithms for large- state-space HMMs with applications to web usage analysis, Proc. of NIPS 2003.
[18] T. Fukuda, Y. Morimoto, S. Morishita and T. Tokuyama, Data mining with optimized two-dimensional association rules, ACM Trans. Database Syst. 26, 179-213, 2001.
[19] T. Gagie, D. Hermelin, G. M. Landau, O. Weimann, Binary Jumbled Pattern Matching on Trees and Tree-Like Structures, Proc. of ESA 2013, LNCS 8125, pp. 517-528, 2013
[20] U. Grenander, Pattern Analysis (New York : Springer-Verlag, 1978).
[21] X. Huang, An algorithm for identifying regions of a dna sequence that satisfy a content requirement, Bioinformatics 10(3), 219-225, 1994.
[22] D. E. Knuth, Estimating the efficiency of backtrack programs, Mathematics of computation 29(129), 122-136, 1975.
[23] O. Kullmann, Fundaments of branching heuristics, Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications 185, pp. 205-244, 2009.
[24] T.-C.Lin and D.T. Lee, Randomized algorithm for the sum selection problem, Theor. Comput. Sci. 377 (May 2007) 151-156.
[25] M. Mitzenmacher and E. Upfal, Probability and Computing: Randomized Algorithms and Probabilistic Analysis (Cambridge University Press, New York, NY, USA, 2005).
[26] T. M. Moosa and M. S. Rahman, Indexing permutations for binary strings, Inf. Process. Lett. 110, 795-798, 2010.
[27] T.M. Moosa and M.S.Rahman, Sub-quadratic time and linear space data structures for permutation matching in binary strings, J. of Discrete Algorithms 10, pp. 5-9, 2012.
[28] W. L. Ruzzo and M. Tompa, A linear time algorithm for finding all maximal scoring subsequences., ISMB, (AAAI, 1999), pp. 234-241, 1999.
[29] R. Williams, Faster all-pairs shortest paths via circuit complexity, arXiv:1312.6680, 2013.

APPENDIX

A. The Proof of Proposition 1

Proof: Let i, j be such that $p_j = p_i + i$. In addition, assume that there is an input A for which P is its unique configuration.

By definition, the subsequence of length j starting at p_j has sum greater than any other subsequence of length j, so that:

$$\sum_{k=p_i}^{p_i+j-1} a_k < \sum_{k=p_j}^{p_j+j-1} a_k$$

By decomposing the previous inequality

$$\sum_{k=p_i}^{p_i+i-1} a_k + \sum_{k=p_i+i}^{p_i+j-1} a_k < \sum_{k=p_j}^{p_j+(j-i)-1} a_k + \sum_{k=p_j+(j-i)}^{p_j+j-1} a_k$$

Replacing p_j by $p_i + i$ in the first summand of the right hand side

$$\sum_{k=p_i}^{p_i+i-1} a_k + \sum_{k=p_i+i}^{p_i+j-1} a_k < \sum_{k=p_i+i}^{p_i+j-1} a_k + \sum_{k=p_j+(j-i)}^{p_j+j-1} a_k \quad \text{so that}$$

$$\sum_{k=p_i}^{p_i+i-1} a_k < \sum_{k=p_j+(j-i)}^{p_j+j-1} a_k \tag{2}$$

By definition, for p_i we also have that:

$$\sum_{k=p_j+(j-i)}^{p_j+j-1} a_k < \sum_{k=p_i}^{p_i+i-1} a_k \tag{3}$$

Adding (2) and (3) we get

$$\sum_{k=p_i}^{p_i+i-1} a_k + \sum_{k=p_j+(j-i)}^{p_j+j-1} a_k < \sum_{k=p_j+(j-i)}^{p_j+j-1} a_k + \sum_{k=p_i}^{p_i+i-1} a_k$$

which is a contradiction. ■