Tracking Maximum Ascending Subsequences in Sequences of Partially Ordered Data

Ton Kloks
Richard B. Tan
Jan van Leeuwen

Technical Report UU-CS-2017-010
May 2017

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Tracking Maximum Ascending Subsequences in Sequences of Partially Ordered Data*

Ton Kloks[1] Richard B. Tan[2] Jan van Leeuwen[3]

[1] Department of Computer Science, National Tsing Hua University
101 Kuang Fu Rd, Sec.2, Hsinchu, Taiwan 300 (R.O.C.)
antonius@ntub.edu.tw

[2] Dept. of Computer Science, University of Sciences & Arts of Oklahoma
Chickasha, OK 73018, USA
rbtan@usao.edu

[3] Dept. of Information and Computing Sciences, Utrecht University
Princetonplein 5, 3584 CC Utrecht, The Netherlands.
j.vanleeuwen1@uu.nl

Abstract. We consider scenarios in which long sequences of data are analyzed and subsequences must be traced that are monotone and maximum, according to some measure. A classical example is the online *Longest Increasing Subsequence Problem* for numeric and alphanumeric data. We extend the problem in two ways: (a) we allow data from any partially ordered set, and (b) we maximize subsequences using much more general measures than just length or weight. Let P be a poset of finite width w, and let δ be any data sequence over P. We show that the measure of the maximum monotone subsequences in δ can be maintained in at most $O(w \log \min(\frac{n}{w}, D_n))$ time and $O(\min(n, wD_n))$ memory when the n-th data item is processed, where D_n is the 'depth' of the measure at position n ($n \geq 1$). The result generalizes all earlier $O(\log n)$ time-per-input results for the corresponding longest or heaviest increasing subsequence problems.

Keywords: data sequences, partially ordered sets, monotone subsequences, online algorithms, sequence measures.

1 Introduction

In the processing of long sequences of data, it is a major problem to find and maintain analytic information about specific patterns. Patterns can be any scattered subsequences of interest that occur in a sequence, with or without a pre-specified characteristic. A classical example is the *Longest Increasing Subsequence Problem* [16]: given a sequence of distinct numbers in a streamlike fashion, keep track of (the length of) the longest increasing subsequence 'from left to right'. In [21] a more general *weighted* version of the problem is considered. We study the problem in the framework of *online algorithms*.

We extend the problem in two ways. First of all, we allow sequences of data from any *partially ordered set* P and want to compute (the length of) the subsequences that are maximum and monotone (ascending or descending) as the sequence unfolds. Secondly, we want subsequences to be maximized using a much wider *class of measures* than just length or weight.

* Version: May 31, 2017.

This leads to the following general problem, for which we seek a solution that is both time- and space efficient. Let $\delta_i = \sigma_1, \cdots, \sigma_i$ $(i \geq 1)$ denote the prefix of any data sequence δ as observed up to position i.

> **Online Maximum Ascending Subsequence Problem**: Given a sequence δ of distinct elements from a partially ordered set P and a sequence measure H, compute the function $F_H(\delta_i)$ for $i = 1, 2, \cdots$ such that for all i with $i \geq 1$, $F_H(\delta_i)$ = '*the maximum H-value over all ascending subsequences of δ_i*'.

We will define later what measures H are allowed. F_H will be termed the 'maximizer' associated with measure H. We assume that the *width* of the posets P we consider is bounded by w, for some finite $w \geq 1$. (The width of a poset is the size of its largest antichain.)

1.1 Problem Background

The archetypical Longest Increasing Subsequence Problem is well-studied in the theory of Young tableaux [33, 35] and permutation graphs [17], and in application areas like computational biology [19]. The problem also arises naturally in the analysis of 'patience sorting' [4, 26, 27].

The (expected) length of longest increasing subsequences is a well-studied entity. By the 'subsequence theorem' of Erdős and Szekeres [14], every sequence of n distinct numbers must have an increasing or decreasing subsequence of at least $\lceil \sqrt{n} \rceil$ terms. For random permutations, the problem of determining the expected length of a longest increasing subsequence is known as the *Ulam-Hammersley problem* [33]. By the Baik-Deift-Johansson theorem [5], the expected length equals $2\sqrt{n} - \alpha n^{\frac{1}{6}} + o(n^{\frac{1}{6}})$ with $\alpha = 1.77108..$ and $n \to \infty$. For more mathematical details we refer to [33].

The extension to posets comes from tuple data. By Dilworth's theorem [13], any poset P of finite width w decomposes into precisely w chains. However, monotone sequences in P need not be confined to a single chain, they can 'meander'. We also allow measures other than length. This makes the Online Maximum Ascending Subsequence Problem a nontrivial extension of the Longest Increasing Subsequence Problem. For sequences of n distinct elements from P, the expected length of a longest ascending subsequence is at least $\lceil \sqrt{\frac{n}{w}} \rceil$.

1.2 Algorithmic Background

The Longest Increasing Subsequence Problem can be solved in $O(n \log L)$ time, where L is the length of the longest increasing subsequence [16, 31]. The algorithm can be extended to produce all longest increasing subsequences, in an additional $O(L)$ time per sequence [6]. The $O(n \log L)$-bound is optimal in common models [16, 32]. If n is known at the outset, $O(n \log \log n)$ implementations exist [20, 25]. In a more powerful RAM model an $O(n \log \log L)$-time algorithm can be achieved [10]. We will not use this model here.

A relevant extension of the Longest Increasing Subsequence Problem was distinguished in [21]. There, similar bounds as above were shown for the *weighted*

case, in which each term of δ has a (position-dependent) non-negative weight and one looks for a *maximum weight* (i.e. 'heaviest') increasing subsequence. A special case is the problem to determine a heaviest monotone subsequence of a permutation of n numbers in which each number has a weight equal to itself. In [34] it is shown that these subsequences must weigh at least $\sqrt{n/3}$.

Many variants of the Longest Increasing Subsequence Problem have been considered. These include e.g. computing longest increasing subsequences with constraints, longest *common* (increasing) subsequences of several sequences, and results depending on domain size [7, 11, 41, 42]. The longest increasing subsequence problem has been studied in parallel computing [2, 8, 22, 29, 30, 36, 37] and it has been used as a benchmark in the study of streaming algorithms [24, 40], in combination with reporting over *sliding windows* [3, 9, 12, 23]. Many of these variants may be studied for the Online Maximum Ascending Subsequence Problem as well.

1.3 Results

In this paper we develop an approach to the Online Maximum Ascending Subsequence Problem, for arbitrary posets of finite width and a very general class of sequences measures. After the precise definitions have been given, we aim for the following.

Let H be a measure and $\delta = \sigma_1, \sigma_2, \cdots$ a sequence of elements from poset P. Let the width of P be bounded by w. We want to compute the values $F_H(\delta_1), \cdots$ such that the *update complexity* when stepping from $F_H(\delta_i)$ to $F_H(\delta_{i+1})$ is small, for every $i \geq 1$. One easily recognizes the Longest Increasing Subsequence Problem as the special case with $w = 1$.

We show that for any maximizer F_H and sequence δ given online, the value of $F_H(\delta_i)$ can be determined using at most $O(w \log \min(\frac{i}{w}, D_i))$ time *per element* while maintaining a data structure in $O(w \min(\frac{i}{w}, D_i))$ memory, as i increases. Here D_i is the 'depth' of F_H at position i, a characteristic we define later. In case H is the normal 'length' measure, D_i equals L_i, the length of the longest ascending subsequence of δ_i.

Aggregating the result for sequences of n distinct poset elements, it follows that the ultimate value of $F_H(\delta_n)$ can be computed using $O(wn \log \min(\frac{n}{w}, D_n))$ time and $O(\min(n, D_n))$ auxiliary storage. This generalizes all known $O(n \log L)$ bounds for the Longest and Heaviest Increasing Subsequence Problems, respectively [16, 21]. Some further applications will be discussed.

1.4 Outline

In Section 2 we define the class of measures we admit as generalizations of measures like length and weight for sequences. We derive several useful properties. We also list the basic assumptions for computing in posets of finitely bounded width.

In Section 3 we describe the online algorithm and basic data structure for computing the values $F_H(\delta_i)$ for consecutive values of i. In Section 3.1 we define the notion of *depth* for measures, and in Section 3.2 we prove the main result that

provides worst-case bounds on the time- and space complexity of the algorithm (Theorem 1).

In Section 4 we comment on the construction of concrete maximum ascending subsequences, and prove a bound on the number of deletions which are involved in maintaining the online data structure, if unused memory would not be automatically reclaimed.

In Section 5 we give examples and argue that our results extend and generalize the earlier results for computing 'longest' or 'heaviest' increasing subsequences[16, 21]. In Section 6 we give some concluding remarks.

Terminology and Notation Let P be a poset. The partial order of P is denoted by $\preceq$. Let P^+ be the set of all finite non-empty sequences of elements of P. Let $P^* = P^+ \cup \{\lambda\}$, where λ is the empty sequence. For every sequence δ of distinct elements ('data items') from P, we assume that *the occurring items are indexed by their position in* δ.

Let $\delta = \sigma_1, \cdots$ be a sequence. A subsequence of length $t \geq 1$ in δ is any sequence $x = \sigma_{i_1}, \cdots, \sigma_{i_t}$ with $i_1 < \cdots < i_t$. An *ascending subsequence* of length $t \geq 1$ in δ is any subsequence $\sigma_{i_1}, \cdots, \sigma_{i_t}$ of δ with $i_1 < \cdots < i_t$ such that $\sigma_{i_1} \preceq \cdots \preceq \sigma_{i_t}$. A *descending subsequence* of length $t \geq 1$ in δ is defined likewise. By δ_i we denote the prefix $\sigma_1, \cdots, \sigma_i$ of δ of length i ($i \geq 1$).

For finite subsequences $\alpha, \beta \in P^*$ we write $\alpha \lhd \beta$ if α is a subsequence of β. We write $\alpha \lhd\!\!\upharpoonright \beta$ in case α is a non-empty subsequence of β with the last term of α *coinciding* with the last term of β.

By $||S||$ we denote the cardinality of S. For $a, b \in \mathbb{R}_+$ with $a < b$, $(a, b]$ denotes the half-open interval from a to b. By convention, we set $\log a = 1$ for any $0 \leq a \leq 2$.

2 Preliminaries

In this section we define the class of online measures for subsequences we consider. Then we define the 'maximizers' associated with them and prove some basic properties. Finally, we describe our assumptions for computing in posets of finitely bounded width.

2.1 Measures

Before defining sequence measures, we define their base functions. Let $\delta = \sigma_1, \sigma_2, \cdots$ be a fixed but arbitrary sequence of elements from poset P.

Definition 1. *A function $B : \mathbb{R}_+ \times \mathbb{N} \times P \to \mathbb{R}_+$ is called a* measure base *(or, a* base*) if it satisfies the following properties for all $s, s' \in \mathbb{R}_+$, $i \in \mathbb{N}$ and $\sigma \in P$:* (a) $B(s, i, \sigma) > s$, *and* (b) $B(s, i, \sigma) \leq B(s', i, \sigma)$ *whenever* $s \leq s'$.

The definition of $B(s, i, \sigma)$ tells us how the measure 's' of a subsequence of δ_{i-1} increases when an item σ arrives in the i-th position of δ. We assume that B-values can be computed in unit time.

Definition 2. *The* sequence measure *associated with a given base function B is the function $H : P^* \to \mathbb{R}_+$ such that* (a) $H(\lambda) = 0$ *and* (b) *for any $i \geq 1$ and $u \lhd \sigma_1, \cdots, \sigma_{i-1}$, $H(u\sigma_i) = B(H(u), i, \sigma_i)$.*

Clearly H is uniquely determined by B and 'easy' to compute, given that B is. For all $i \geq 1$, $H(\sigma_i) = B(0, i, \sigma_i)$ and hence $H(\sigma_i) > 0$.

Example 1. Let $w : \mathbb{N} \times P \to \mathbb{R}_+$ be a weight function. Let B be defined by $B(s, k, \sigma) = s + w(k, \sigma)$. B is a measure base, and its associated measure is $H(u) =$ '*the 'positioned' weight of u*', where symbols are weighed by their position in δ. This is the measure used in [21]. A special case arises when P has an additive structure, and $w(k, \sigma) = \sigma$. Then $H(u) =$ '*the 'sum' of the elements in u*'.

Lemma 1. *Let H be any sequence measure. Then for all finite subsequences y, x of δ we have: if $y \lhd x$, then $H(y) \leq H(x)$.*

Proof. Let B be the measure base of H. Consider y, x with $y \lhd x$ and $x \lhd \sigma_1, \cdots, \sigma_i$. We prove the lemma by induction on i. For $i = 1$ we observe that $(0 =)\, H(\lambda) \leq H(y) \leq H(\sigma_1)$ for $y = \lambda$ and $y = \sigma_1$, hence for all $y \lhd \sigma_1$, and we are done. Assume the lemma holds for $i - 1$, for some $i - 1 \geq 1$.

Consider $y \lhd x$ for some $x \lhd \sigma_1, \cdots, \sigma_i$. If $x \lhd \sigma_1, \cdots, \sigma_{i-1}$, then $H(y) \leq H(x)$ by induction. If $x \lhd\!\restriction \sigma_1, \cdots, \sigma_i$, write $x = x'\sigma_i$. Now two cases can arise:

- $y \lhd x'$. By the induction hypothesis and the properties of B we have: $H(y) \leq H(x') \leq B(H(x'), i, \sigma_i) = H(x'\sigma_i) = H(x)$.
- $y \lhd\!\restriction x'\sigma_i$. Then y is of the form $y = y'\sigma_i$ with $y' \lhd x'$. By the induction hypothesis we have $H(y') \leq H(x')$ and consequently by the properties of B: $H(y) = H(y'\sigma_i) = B(H(y'), i, \sigma_i) \leq B(H(x'), i, \sigma_i) = H(x'\sigma_i) = H(x)$.

This completes the induction. □

By Lemma 1, the maximum value of H over all subsequences of δ_n is equal to $H(\delta_n)$, for all $n \geq 0$.

2.2 Maximizers

Let H be a sequence measure, with base B. Let Asc be the predicate on subsequences defined by $Asc(y) \equiv$ '*y is an ascending subsequence of δ*.' For consistency we assume that $Asc(\lambda)$.

Definition 3. *A function $F_H : P^* \to \mathbb{R}_+$ is called the* maximizer *of H if for any $i \geq 1$ and $x \lhd \sigma_1, \cdots, \sigma_i$ we have $F_H(x) = \max\{H(u) \mid u \lhd x \text{ and } Asc(u)\}$.*

A function F is called a *maximizer* if there is a sequence measure H such that $F = F_H$. We omit subscripts if H is understood from the context. Maximizers F are everywhere defined and monotone: if $y \lhd x$, then one sees from Lemma 1 that $F(y) \leq F(x)$. Note that $F(\lambda) = 0$, and thus we have $F(\lambda) < F(x)$ for all $x \neq \lambda$.

Example 2. Continuing Example 1, define F by $F(\lambda) = 0$ and $F(x) =$ '*the maximum positioned weight of any ascending subsequence of* x', for any $x \lhd \delta$ with $x \neq \lambda$. Then F is the maximizer of H. The special case in Example 1 leads to the *Maximum Sum Ascending Subsequence Problem*.

Lemma 2. *Let F be the maximizer of H. Then for all $x \lhd \delta$, $F(x) = H(x)$ whenever $Asc(x)$. Also, $F(x) = \max\{F(u) \mid u \lhd x \text{ and } Asc(u)\}$.*

Proof. By definition, $F(x) = \max\{H(u) \mid u \lhd x \text{ and } Asc(u)\}$ for all subsequences x of δ. From this and with the help of Lemma 1 it follows that for all u with $Asc(u)$, necessarily $F(u) = H(u)$. Consequently $F(x) = \max\{F(u) \mid u \lhd x \text{ and } Asc(u)\}$. □

Lemma 3. *Let F be the maximizer of H. Then for all $i \geq 1$ and $x \lhd \sigma_1, \cdots, \sigma_i$ we have $F(x\sigma_{i+1}) = \max\{F(x), H_{x,i+1}\}$, where $H_{x,i+1} = \max\{H(y) \mid y \lhd^{\uparrow} x\sigma_{i+1}$ and $Asc(y)\}$.*

Proof. By definition, $F(x\sigma_{i+1}) = \max\{H(y) \mid y \lhd x\sigma_{i+1} \text{ and } Asc(y)\}$. For all y, $y \lhd x\sigma_{i+1}$ if and only if $y \lhd x$ or $y \lhd^{\uparrow} x\sigma_{i+1}$. It follows that $F(x\sigma_{i+1})$ is the maximum of $\max\{H(y) \mid y \lhd x \text{ and } Asc(y)\} = F(x)$ and $\max\{H(y) \mid y \lhd^{\uparrow} x\sigma_{i+1} \text{ and } Asc(y)\} = H_{x,i+1}$. □

Given a sequence δ and maximizer F, the values $F(\delta_1), F(\delta_2), \cdots$ may thus be computed as follows. At every stage, remember the most recent F-value. When the next input $\sigma_{i+1} \in P$ arrives, compute $H_{\delta_i,i+1}$ and set $F(\delta_{i+1}) = \max\{F(\delta_i), H_{\delta_i,i+1}\}$. Setting $y = z\sigma_{i+1}$ in the definition of $H_{\delta_i,i+1}$, we obtain $H_{\delta_i,i+1} = \max\{B(H(z), i+1, \sigma_{i+1}) \mid z \lhd \delta_i \text{ and } Asc(z\sigma_{i+1})\}$. Thus, $H_{\delta_i,i+1}$ could be computed by going through all subsequences z of δ_i and determining the maximum $B(H(z), i+1, \sigma_{i+1})$-value for those z's for which $Asc(z\sigma_{i+1})$. (The maximum exists, as at least $z = \lambda$ satisfies the required properties.) We examine in Section 3 how to do this 'fast' in posets of finitely bounded width.

2.3 Computing in Posets

We need a few more assumptions in order to be able to process the elements of δ. Let P be a poset of finite width (at most) w, By Dilworth's theorem [13], we may as well assume that P is given by a *decomposition* into w disjoint and ascending chains $C_1, \cdots, C_w$. (NB. Elements that belong to different chains may still be comparable in P.)

With suitable effectiveness conditions on this kind of representation, it becomes possible to compute with the elements of P. In particular, as in [15], we assume two computable predicates, $\mathbf{in}(x, C)$ and $\mathbf{comp}(x, y)$, that can be tested in *unit time*. Given C and $x, y \in P$, the predicate $\mathbf{in}(x, C)$ tells whether x belongs to chain C or not and the predicate $\mathbf{comp}(x, y)$ tells whether $x \preceq y$ or not. Predicate **comp** enables one to make comparisons in and across chains, and decide whether x and y are comparable in P at all.

3 Computing Maximizers Online

For sequences over totally ordered sets it is known that one can efficiently compute the length of longest increasing subsequences online [16]. In this section we consider the general problem, for sequences over any poset P of finitely bounded width and using any permissible sequence measure H.

Let P and H (with base B) be given, let F be the associated maximizer and $\delta = \sigma_1, \sigma_2, \cdots$ a sequence of (distinct) elements from P. Let $C_1, \cdots, C_w$ be a Dilworth decomposition of P that fulfils the requirements above. In Subsection 3.1 we give the basic definitions we need, in Subsection 3.2 we develop the general online algorithm to compute the values $F(\delta_i)$ for $i \geq 1$.

3.1 Preparations

Our aim will be to compute $F(\delta_i)$ using only a small amount of stored information for every $i \geq 1$. We begin by observing the following property.

Lemma 4. *Let subsequences $u, v, w \lhd \delta$ be such that $u\sigma_j w$ and $v\sigma_k w$ are ascending and $F(u\sigma_j) \leq F(v\sigma_k)$. Then $F(u\sigma_j w) \leq F(v\sigma_k w)$.*

Proof. By induction. The assertion trivially holds for $w = \lambda$. Assume it holds for w. Let σ_l be such that $u\sigma_j w\sigma_l$ and $v\sigma_k w\sigma_l$ are both ascending. By Lemma 2 we have $F(u\sigma_j w\sigma_l) = H(u\sigma_j w\sigma_l) = B(H(u\sigma_j w), l, \sigma_l) \leq B(H(v\sigma_k w), l, \sigma_l) = H(v\sigma_k w\sigma_l) = F(v\sigma_k w\sigma_l)$. This proves the induction step. □

Definition 4. $T_i = \{(F(y\sigma_j), \sigma_j) \mid y\sigma_j \lhd \delta_i \text{ and } y\sigma_j \text{ is ascending}\}$.

Note that $F(\delta_i)$ is equal to the largest F-value of any pair in T_i. All we need is to filter it out. In order to do so efficiently, we generalize the approach as used in the Longest Increasing Subsequence Problem.

It can be seen from Lemma 4 that, if $(F(u\sigma_j), \sigma_j), (F(v\sigma_k), \sigma_k) \in T_i$ with $\sigma_k \prec \sigma_j$ but $F(v\sigma_k) \geq F(u\sigma_j)$, then $(F(u\sigma_j), \sigma_j)$ can be omitted from further consideration when maximizing F at all later stages. The same can be said if $\sigma_k = \sigma_j$ and $F(v\sigma_k) > F(u\sigma_j)$. In all these cases we say that $(F(v\sigma_k), \sigma_k)$ *cancels* $(F(u\sigma_j), \sigma_j)$. This motivates the following definition.

Definition 5. *A set of pairs $S_i = \{(f_{j_1}, \sigma_{j_1}), \cdots, (f_{j_k}, \sigma_{j_k})\}$ is said to be* extremal *in T_i if* (a) *$S_i \subseteq T_i$, and* (b) *for every pair $(F(y\sigma_j), \sigma_j) \in T_i$ there is a pair $(f_{j_p}, \sigma_{j_p}) \in S_i$ with $\sigma_{j_p} \preceq \sigma_j$ and $f_{j_p} \geq F(y\sigma_j)$, and* (c) *$S_i$ is minimal w.r.t. to the previous properties.*

As we assumed δ to be a sequence of *distinct* elements of P, no pair of an extremal set can be cancelled by another pair in the set. (If duplicates are allowed, then uniqueness can be enforced by requiring e.g. that σ_{j_p} always equals the first occurrence of the element in δ for a given f_{j_p}-value.)

Lemma 5. *Every T_i ($i \geq 1$) has an extremal subset S_i, and this subset is unique.*

Proof. Consider any set T_i ($i \geq 1$). Define the binary relation $\preceq_i$ on its pairs by $(F(u\sigma_k), \sigma_k) \preceq_i (F(u\sigma_j), \sigma_j)$ if and only if $(F(u\sigma_k), \sigma_k) = (F(u\sigma_j), \sigma_j)$ or $(F(u\sigma_k), \sigma_k)$ cancels $(F(u\sigma_j), \sigma_j)$. The key fact to observe is that $\preceq_i$ is a partial order on T_i.

Let A be the set of all pairs of T_i that are *minimal* in $\preceq_i$. As T_i is finite, A is well-defined. Clearly, A is extremal in T_i. Observe that every extremal subset of T_i must include all the minimal elements of T_i, i.e. all elements of A. Thus, as an extremal subset in T_i, A is unique. □

It is easily seen then, that by repeating cancellations in T_i (in arbitrary order) until no further ones are possible, one necessarily arrives at the one subset of T_i that must be extremal.

Next, we consider the natural way in which every extremal set S_i is partitioned into w subsets by the Dilworth decomposition of P.

Definition 6. *For $1 \leq t \leq w$, let $S_{t,i} = \{(f_{j_p}, \sigma_{j_p}) \in S_i \mid \sigma_{j_p} \in C_t\}$.*

Lemma 6. *For every t ($1 \leq t \leq w$), $S_{t,i}$ is a totally ordered set (under the natural tuple ordering).*

Proof. Assume that $S_{t,i}$ is not empty. Let $(f_{j_p}, \sigma_{j_p}), (f_{j_q}, \sigma_{j_q}) \in S_{t,i}$. As σ_{j_p} and σ_{j_q} belong to the same C_t, they are ordered and we may assume w.l.o.g. that $\sigma_{j_p} \preceq \sigma_{j_q}$. Suppose $f_{j_p} \geq f_{j_q}$. Then S_i would still be extremal after deleting pair (f_{j_q}, σ_{j_q}), contradicting minimality. It follows that $(f_{j_p}, \sigma_{j_p}) \leq' (f_{j_q}, \sigma_{j_q})$, where $\leq'$ is used to denote the natural partial ordering on tuples. □

For the further analysis we need various concepts and notations for the 'range' of F, both in general and when restricted to the subsequences that have their front end in a given chain C_t.

Definition 7. (a) $s_i = ||S_i||$. (b) $s_{t,i} = ||S_{t,i}||$.

Definition 8. (a) $R_i = \{F(y) \mid y \neq \lambda, y \lhd \delta_i,$ and y is ascending$\}$. (b) *The* depth *of F at position i is equal to $D_i = ||R_i||$.*

Definition 9. *(a) $R_{t,i} = \{F(y\sigma_j) \mid y\sigma_j \lhd \delta_i$, $\sigma_j \in C_t$ and $y\sigma_j$ is ascending$\}$.* (b) *The* depth *of F in chain C_t at position i is equal to $D_{t,i} = ||R_{t,i}||$.*

Clearly $D_{t,i} \leq D_i$ but $D_i \leq \Sigma_{t=1}^{w} D_{t,i} \leq wD_i$ due to the fact that the ranges $R_{t,i}$ ($1 \leq t \leq w$) need not be disjoint. Also, using Lemma 2 one sees that $F(\delta_i)$ is the largest value in R_i. Hence, if F is integer-valued, one has $D_i \leq F(\delta_i)$.

Proposition 1. *For every t ($1 \leq t \leq w$), $s_{t,i} \leq D_{t,i}$ and $s_i = \Sigma_{t=1}^{w} s_{t,i} \leq \min(i, wD_i)$.*

Proof. The inequality $s_{t,i} \leq D_{t,i}$ follows because the pairs $(f_{j_p}, \sigma_{j_p}) \in S_{t,i}$ must all have different f_{j_p}-values, which are values in $R_{t,i}$. Subsequently $s_i = \Sigma_{t=1}^{w} s_{t,i} \leq \Sigma_{t=1}^{w} D_{t,i} \leq wD_i$. Also $s_i \leq i$, as we have processed i terms of δ. □

3.2 Online Algorithm

The preparatory analysis implies that $F(\delta_i)$ is equal to the largest F-value of any pair in S_i, for any $i \geq 1$. We show how one can efficiently compute S_{i+1} from S_i when the next element σ_{i+1} of δ is taken in.

Instead of maintaining set S_i directly, we will maintain its partition into subsets $S_{t,i}$ $(1 \leq t \leq w)$. Notice that $F(\delta_i)$ can easily be determined in $O(w)$ time if the sets are structured right, by picking the largest f-value of the top pairs in these (sorted) subsets.

For efficiency we store each set $S_{t,i}$ in sorted order in a *concatenable queue*. This type of data structure allows one to perform *Min* (*Max*), *Find*, *Insert*, *Delete*, *Split*, and *Join* operations in (at most) logarithmic time per operation. Concatenable queues can be implemented by means of 2-3 trees or other balanced search trees [1, 28].

Lemma 7. *For every $i \geq 1$, S_i can be stored in $O(\min(i, wD_i))$ memory space.*

Proof. Concatenable queues can be implemented in linear space [1, 28]. Using Proposition 1, it follows that the subsets $S_{t,i}$ can be stored in $\Sigma_{t=1}^{w} s_{t,i} = O(\min(i, wD_i))$ space. □

Let the next element σ_{i+1} of δ be provided, and suppose that $\sigma_{i+1} \in C_l$. We want to update the partition of S_i, to obtain the partition of S_{i+1}. As a first step we have to determine the pair $(f_0, \sigma_{i+1}) \in T_{i+1}$ with largest f_0. Next we have to see how to insert it.

This step is not merely a matter of updating subset $S_{l,i}$. Element σ_{i+1} may be comparable to elements in *several* chains, and thus (f_0, σ_{i+1}) may cause cancellations in other subsets besides $S_{l,i}$ as well, in the process of maintaining extremality. Fortunately, one can efficiently delimit the parts of every $S_{t,i}$ that must be updated.

Definition 10. (a) *$pre_{t,i}$ = the greatest pair in $S_{t,i}$ with σ-value less than σ_{i+1} (*undefined if no such pair exists*).* (b) *$post_{t,i}$ = the smallest pair in $S_{t,i}$ with σ-value greater than σ_{i+1} (*undefined if no such pair exists*).*

***Notation* 1** *We write $pre_{t,i} = (preF_{t,i}, preP_{t,i})$ and $post_{t,i} = (postF_{t,i}, postP_{t,i})$ (with components undefined if $pre_{t,i}$ and $post_{t,i}$ are).*

As the elements of δ are assumed to be all distinct, $preP_{t,i}$ and $postP_{t,i}$ uniquely determine the pairs $pre_{t,i}$ and $post_{t,i}$ respectively. Within each subset $S_{t,i}$ alone, $preF_{t,i}$ and $postF_{t,i}$ do so as well.

Lemma 8. *For every $i \geq 1$, the pairs $pre_{t,i}$ and $post_{t,i}$ $(1 \leq t \leq w)$ can be determined in $O(w \log \min(\frac{i}{w}, D_i))$ steps.*

Proof. The concatenable queues store their elements (f, σ_j) in tuple-sorted order. Hence, using the assumptions for comparing elements of P, one can determine $pre_{t,i}$ by searching (using the second component) for the largest pair with second component less than σ_{i+1} in $S_{t,i}$ (if existent), treating elements of

P that happen to be incomparable to σ_{i+1} as being 'larger' than σ_{i+1}. In the search tree implementation of concatenable queues, this can be done in at most logarithmic time. One can determine $post_{t,i}$ in a similar way. For every t, this takes a number of operations (comparisons) in the order of

$$\log(1+s_{1,i})+\cdots+\log(1+s_{w,i}) \leq w \log \frac{w+\Sigma_{t=1}^{w} s_{t,i}}{w} \leq w \log \min(1+\frac{i}{w}, 1+D_i)$$

using *Jensen's Inequality* and Proposition 1. The lemma follows, using our convention for log-values. □

This leads to the recipe for updating S_i: compute the proper pair $(f_0, \sigma_{i+1}) \in T_{i+1}$, do the necessary cancellations in all subsets $S_{t,i}$ in view of the new information, and insert (f_0, σ_{i+1}) into whatever is remaining of $S_{l,i}$. The details are given in the following result.

Lemma 9. *For all $i \geq 1$, S_{i+1} can be computed from S_i in $O(w \log \min(\frac{i}{w}, D_i))$ time.*

Proof. Assume w.l.o.g. that $i \geq 1$ and that the subsets $S_{t,i}$ of S_i are available in memory. Let $Q_{t,i}$ denote the concatenable queue in which the elements of $S_{t,i}$ are stored ($1 \leq t \leq w$). Recall that we assumed that $\sigma_{i+1} \in C_l$. The recipe for updating S_i proceeds as follows.

1) *Compute the pair $(f_0, \sigma_{i+1}) \in T_{i+1}$ with largest f_0.* This first step requires us to determine a subsequence u with $u \lhd \delta_i$, such that $u\sigma_{i+1}$ is ascending and $F(u\sigma_{i+1}) = B(F(u), i+1, \sigma_{i+1})$ is maximum (cf. Lemma 3). As B is monotone in its first argument, it means we must maximize $F(u)$, subject to $u \lhd \delta_i$ and $u\sigma_{i+1}$ ascending. A first value to consider is $F(\lambda)$, taking $u = \lambda$.

To see if a larger value can be obtained, write $u = u'\sigma'$, consider all possibilities with σ' occurring in δ_i and $\sigma' \prec \sigma_{i+1}$, and look for the largest value $F(u'\sigma')$ in any pair $(F(u'\sigma'), \sigma') \in T_i$. As S_i is extremal in T_i, the largest F-value must occur in one of the pairs $pre_{t,i}$ in $S_{t,i}$ for $t = 1, \cdots, w$. Determining all pairs $pre_{t,i}$ can be done in $O(w \log \min(\frac{i}{w}, D_i))$ time, by Lemma 8.

If no σ' with the required characteristic exists (thus $pre_{t,i}$ is undefined for all t), we stay with $u = \lambda$ and set $f_0 = B(F(\lambda), i+1, \sigma_{i+1}) = F(\sigma_{i+1})$. Otherwise, let the maximum F-value occur in the pair $pre_{t,i}$ with $t = p$. (This value is surely larger than $F(\lambda)$.) Then compute $f_0 = B(preF_{p,i}, i+1, \sigma_{i+1})$.

2) *Do the necessary cancellations in all subsets $S_{t,i}$ in view of the new information implied by the pair (f_0, σ_{i+1}).* By construction, the new pair (f_0, σ_{i+1}) cannot be cancelled by any existing pair in S_i. On the other hand, $S_i \cup \{(f_0, \sigma_{i+1})\}$ may not be extremal in T_{i+1} just yet.

In order to restore the extremal property, we have to determine all pairs $(f', \sigma') \in S_i$ which are now cancelled by (f_0, σ_{i+1}). This means we have to look at every chain C_t ($1 \leq t \leq w$), and determine the pairs $(f', \sigma') \in S_{t,i}$ with $f' \leq f_0$ and $post_{t,i} \preceq \sigma'$ and delete them. By the sortedness of $S_{t,i}$ these elements all occur consecutively, from the pair with σ-value $postP_{t,i}$ (if it exists) onward, i.e. as a contiguous segment. Deleting them is easy, given the concatenable queues. For every t, we do this as follows:

- First split $Q_{t,i}$ into concatenable queues $Q^1_{t,i}$ and $Q^2_{t,i}$, such that $Q^1_{t,i}$ consists of all pairs with σ-value σ' such that $\sigma' \prec postP_{t,i}$ and $Q^2_{t,i}$ consists of all remaining (bigger) pairs. This split can be realized as we located the pair $post_{t,i}$ before.
- Next split $Q^2_{t,i}$ into concatenable queues $Q^{2a}_{t,i}$ and $Q^{2b}_{t,i}$, such that $Q^{2a}_{t,i}$ consists of all pairs (f', σ') of $Q^2_{t,i}$ with $f' \leq f_0$ and $Q^{2b}_{t,i}$ consists of all remaining (bigger) pairs. This split can be realized efficiently after locating the pair with F-value equal to f_0, using a search in $Q^2_{t,i}$ on the first component of the pairs. If no such pair exists, the search will automatically yield the pair with largest possible f' such that $f' \leq f_0$. This will be the pair to split around.
- Join $Q^1_{t,i}$ and $Q^{2b}_{t,i}$ into concatenable queue $Q^3_{t,i}$, which can be done efficiently as the pairs of $Q^1_{t,i}$ are all less than those of $Q^{2b}_{t,i}$. We 'forget' (delete) the elements in $Q^{2a}_{t,i}$ as these are all cancelled by (f_0, σ_{i+1}).

3) *Insert* (f_0, σ_{i+1}) *into whatever is remaining of* $S_{l,i}$. As all necessary cancellations are now done, (f_0, σ_{i+1}) can be inserted in the queue $Q^3_{l,i}$.

The resulting overall set of pairs in the concatenable queues $Q^3_{t,i}$ is extremal and thus equal to S_{i+1} by construction. Renaming $Q^3_{t,i}$ into $Q_{t,i+1}$ for every t $(1 \leq t \leq w)$ gives the representation for S_{i+1} as needed for processing the next element of δ.

The number of operations (searches, splits, joins) to do the updates in each $Q_{t,i}$ is not worse than logarithmic and thus, summed over all t with $1 \leq t \leq w$, equal to $O(w \log \min(\frac{i}{w}, D_i))$ as in the proof of Lemma 8. Inserting (f_0, σ_{i+1}) in $Q^3_{l,i}$ takes another $O(\log s_{l,i})$ steps, which only adds at most a constant factor to the overall bound. □

Combining the lemmas in this section, we obtain the following solution to the *Online Maximum Ascending Subsequence Problem*.

Theorem 1. *Let F be a maximizer, and δ a sequence of distinct elements from a poset P of width w. One can compute the values $F(\delta_i)$ online, using at most $O(\min(i, wD_i))$ memory and $O(w \log \min(\frac{i}{w}, D_i))$ update time when stepping from $F(\delta_i)$ to $F(\delta_{i+1})$ $(i \geq 1)$.*

In Section 5 we will discuss how and to what extent the theorem generalizes all prior results for the Longest Increasing Subsequence problem.

4 Further Aspects of Computing Maximizers

In this Section we consider a number of further aspects of the given online algorithm for computing maximizers. This includes the (overall) complexity of evaluating a maximizer on a complete input sequence, the determination of an actual 'maximum ascending subsequence' at every stage, and a more explicit way of dealing with deletions of pairs.

4.1 Computing Maximizers for Complete Sequences

Let F be a maximizer of the general type we consider. Theorem 1 bounds the update complexity for computing the values of $F(\delta_i)$ online. Summing the update complexities, the result implies the following bound on the complexity of computing the value of $F(\delta_n)$ for any n.

Theorem 2. *Let F be a maximizer, and δ a sequence of distinct elements from a poset P of width w. For any $n \geq 1$ one can compute $F(\delta_n)$ using at most $O(\min(n, wD_n))$ memory and $O(wn \log \min(\frac{n}{w}, D_n))$ time, where D_n is the depth of F at position n.*

By the considerations in Section 1.1 it can be expected that D_n is much smaller than $\frac{n}{w}$ in practice. If memory space is bounded, then the algorithm may be run with an upperbound on the length of the subsequences that can be found.

4.2 Determining Maximizing Subsequences

With little extra effort the given online algorithm can be *extended* so a concrete ascending subsequence $u \lhd \delta_i$ such that $F(u) = F(\delta_i)$ can be deduced also, for every $i \geq 1$ for which one would want to do this. The technique for it makes use of *back chaining*, as in algorithms for the Longest Increasing Subsequence Problem.

The necessary backpointers are created in the following way. Referring to the proof of Lemma 9, whenever a next term σ_{i+1} of δ is supplied, a pair $(f_0, \sigma_{i+1}) \in T_{i+1}$ with largest f_0 is determined and added into the extremal set. If $f_0 = F(\sigma_{i+1})$ then σ_{i+1} by itself is a maximum ascending subsequence ending at σ_{i+1} and no pointer needs to be set. If $f_0 = B(preF_{p,i}, i+1, \sigma_{i+1})$ for some $1 \leq p \leq w$, one has to add a *back pointer* from (f_0, σ_{i+1}) to $pre_{p,i}$ into the record of (f_0, σ_{i+1}) in the data structure. By simply following the back pointers starting from a pair, an ascending subsequence (in reverse order) is recovered that produces the F-value in the pair. One can enumerate all maximizing ascending subsequences also, by the same technique as in [6].

To allow for back chaining at all stages, it is requires that during the algorithm, only those pairs are deleted ('cancelled') which have no incoming back pointer(s) and thus do not figure in any potential solution anymore.

4.3 Estimating Deletions

In the online algorithm in Section 3 we took an easy approach when it came to deleting cancelled pairs. In effect we assumed that the elements of every queue $Q^{2a}_{s,i}$ can be returned to 'free space' at no charge. When explicit back pointers are recorded and pairs remain 'links' in active back-chains, it may be necessary to do deletions 'manually'. This also arises when memory must be managed directly.

For this, it will be useful to have an estimate on the number of elements that is deleted in the construction of S_{i+1} from S_i, for every i.

Definition 11. *The* density *of F at position i is equal to the value of* $E_i = \max\{||(F(u), F(u\sigma_{i+1})] \cap R_i|| \mid u \lhd \delta_i \text{ and } u\sigma_{i+1} \text{ ascending}\}$.

Clearly $1 \leq E_i \leq D_i$ $(i \geq 1)$. Now consider the online algorithm in Section 3.2, in particular the proof of Lemma 9 where it is shown which pairs are cancelled when updating the online information.

Lemma 10. *For all $i \geq 1$, in constructing S_{i+1} from S_i, at most* $\min(i, E_i + (w-1)D_i)$ *pairs are deleted ('cancelled') and returned to free space 'en groupe'.*

Proof. Surely no more than i pairs can be cancelled, as it is the total number of pairs we have seen up until this point in the algorithm. For the second part of the estimate, we consider how the pair (f_0, σ_{i+1}) is determined in the proof of Lemma 9.

The first possibility is that $f_0 = F(\sigma_{i+1})$. In this case, no elements from C_l smaller than σ_{i+1} can occur in δ_i ('before σ_{i+1}'). It means that in subset $S_{l,i}$, at best the bottom element of the current queue can get cancelled. It would be exactly the pair $(postF_{t,i}, postP_{t,i})$ with $(F(\lambda) <)\, postF_{t,i} \leq F(\sigma_{i+1})$. If this arises, one clearly has $E_i \geq 1$. Estimating the number of cancellations in all $S_{t,i}$ with $t \neq l$ by $s_{t,i} \leq D_i$, the total number of cancellations in this case is bounded by $E_i + (w-1)D_i$.

The second possibility is that (f_0, σ_{i+1}) arises from a defined pair $pre_{p,i}$ with largest F-value. Letting $preF_{p,i} = F(u)$ for some ascending u with $u \lhd \delta_i$, f_0 was set to $f_0 = B(preF_{p,i}, i+1, \sigma_{i+1}) = F(u\sigma_{i+1})$. This means that in subset $S_{p,i}$, the new pair cancels all pairs (f, σ') with $f \leq F(u\sigma_{i+1})$ and $(\sigma_{p,i} \prec) post_{t,i} \preceq \sigma'$ (if existent). Now note that $f_0 > preF_{p,i} = F(u)$ (by the assumptions on B) and that pairs in $S_{p,i}$ all have different F-values (by extremality). It follows that at best all pairs (f, σ') with $F(u) < f \leq F(\sigma_{i+1})$ get cancelled in $S_{p,i}$, and there are at most E_i of these pairs. Estimating the number of cancellations in all $S_{t,i}$ with $t \neq p$ by $s_{t,i} \leq D_i$, the total number of cancellations in this case is once again bounded by $E_i + (w-1)D_i$. □

Corollary 1. *Let F be a maximizer, and δ a sequence of distinct elements from a totally ordered set. One can compute the values $F(\delta_i)$ online, using at most $O(\min(i, D_i))$ memory and $O(\log \min(i, D_i))$ time for updating when stepping from $F(\delta_i)$ to $F(\delta_{i+1})$. The algorithm deletes at most E_i data items 'en groupe' in doing so $(i \geq 1)$.*

Proof. Substitute $w = 1$ in Theorem 1 and Lemma 10. □

5 Applications

We conclude with some examples of the algorithmic results, which illustrate the generality of the complexity bounds that we obtained.

5.1 Longest Ascending Subsequences

Continuing the general approach, consider sequences δ of distinct elements over some poset P of finite width w. The most common example of a maximizer for

these sequences is $F(\delta_i)$ = *the length of a longest ascending subsequence of* δ_i. F is obtained by taking the measure implied by the base $B : \mathbb{R}_+ \times \mathbb{N} \times P \to \mathbb{R}_+$ with $B(s, i, \sigma) = s + 1$.

***Notation* 2** *Let* L_i *be the* length *of the longest ascending subsequence of* δ_i *according to the partial order* ($i \geq 1$).

The *depth* of F at position i can be seen to be $D_i = ||R_i|| = ||\{F(y) \mid y \neq \lambda, y \lhd \delta_i \text{ and } y \text{ is ascending}\}|| = L_i$. The *density* of F at position i is $E_i = \max\{||(F(u), F(u\sigma_{i+1})] \cap R_i|| \mid u \lhd \delta_i \text{ and } u\sigma_{i+1} \text{ ascending}\} = 1$.

Theorem 1 and Lemma 10 imply the following bounds for the Online Maximum Ascending Subsequence Problem.

Theorem 3. *Let* δ *be a sequence of distinct elements from poset* P *of width* w. *One can compute* L_i *online, using at most* $O(\min(i, wL_i))$ *memory and* $O(w \log \min(\frac{i}{w}, L_i))$ *update time when stepping from* L_i *to* L_{i+1}. *The algorithm deletes at most* $\min(i, 1 + (w-1)L_i)$ *data items 'en bloc' in doing so* ($i \geq 1$).

If one is only interested in accumulating L_n for a sequence of n distinct elements, it is clear that the total number of deletions can never be more than n. In this case, the result simplifies to the following.

Theorem 4. *Let* δ_n *be a sequence of* n *distinct elements from poset* P *of width* w. *For any* $n \geq 1$ *one can compute* L_n *in* $O(wn \log \min(\frac{n}{w}, L_n))$ *time and using at most* $O(\min(n, wL_n))$ *memory* ($n \geq 1$).

Theorem 4 generalizes the known results for $w = 1$, in particular it generalizes the $O(n \log n)$ bounds arising from the classical *Robinson-Schensted algorithm* for computing longest increasing subsequences of permutations [35] and Fredman's $O(n \log L)$ result [16].

In Section 1.1 it was argued that 'in practice' L_n can be expected to be 'small' with respect to n. For random permutations of numbers one has $L_n \approx 2\sqrt{n}$ [18, 33], whereas for finite sequences over arbitrary posets P of width w one may expect L_n to be $O(\sqrt{\frac{n}{w}})$. This means that the generalized result has a similar range of practicality as the original Longest Increasing Subsequence Problem.

5.2 Heaviest Ascending Subsequences

We now consider sequences δ of distinct, *weighted* elements over a poset P of finite width w. For a general example, we assume that the weights are chosen from a fixed set of positive real weights $\{b_1, \cdots, b_k\}$ with $b_1 < \cdots < b_k$ ($k > 1$).

We consider the sequence measure ('weight') implied by the base function $B : \mathbb{R}_+ \times \mathbb{N} \times P \to \mathbb{R}_+$ with $B(s, i, \sigma) = s + b_{s,i,\sigma}$, where $b_{s,i,\sigma} \in \{s_1, \cdots, s_k\}$ is a parameterized weight assignment such that $b_{s,i,\sigma} \leq b_{s',i,\sigma}$ when $s \leq s'$. Note that we allow $b_{s,i,\sigma}$ to depend not only on position and symbol but also on the initial sum s. This gives a valid measure base. The corresponding maximizer is $F(\delta_i)$ = *the largest total weight of any ascending subsequence of* δ_i. We use the term 'heaviest' for any subsequence of the required characteristic.

***Notation* 3** *Let W_i be the* weight *of any heaviest ascending subsequence of δ_i according to the partial order ($i \geq 1$).*

Define $\beta = (b_k - b_1)/\min_j(b_{j+1} - b_j)$ as the *amplitude* of the set of weights, and $\gamma = b_k/\min_j(b_{j+1} - b_j)$ as the *stretch*. Note that always $\beta \geq k - 1$, but that e.g. $\beta = k - 1$ for k equally spaced weights. To estimate the characteristics of F, we observe the following. Recall that L_i denotes the length of a longest ascending subsequence of δ_i, independent of the weights.

- The *depth* of F at position i is bounded by $D_i = ||R_i|| = ||\{F(y) \mid y \neq \lambda, y \lhd \delta_i \text{ and } y \text{ is ascending}\}|| \leq \beta L_i$. This follows because the possible F-values must all lie between b_1 (smallest) and $b_k L_i$ (largest) and are separated by (at least) $\min_j(b_{j+1} - b_j)$.
- The *density* of F at position i is bounded by $E_i = \max\{||(F(u), F(u\sigma_{i+1})] \cap R_i|| \mid u \lhd \delta_i \text{ and } u\sigma_{i+1} \text{ ascending}\} \leq \gamma$. This follows because the difference between any two values $F(u)$ and $F(u\sigma_{i+1})$ is at most b_k and intermediate values are (at least) $\min_j(b_{j+1} - b_j)$ apart.

Theorem 1 and Lemma 10 imply the following bounds for the Online Maximum Ascending Subsequence Problem in this weighted case.

Theorem 5. *Let δ be a weighted sequence of distinct elements from poset P of width w, in the sense as defined. One can compute W_i online, using at most $O(\min(i, w\beta L_i))$ memory and $O(w \log \min(\frac{i}{w}, \beta L_i))$ update time when stepping from W_i to W_{i+1}. The algorithm deletes at most $\min(i, \gamma + (w-1)\beta L_i)$ data items 'en bloc' in doing so ($i > 1$).*

Finally, if one is interested in merely accumulating W_n for a weighted sequence of n distinct elements, it is clear that the total number of deletions is again never be more than n. In this case, the result simplifies to the following.

Theorem 6. *Let δ_n be a weighted sequence of n distinct elements from poset P of width w, in the sense as defined above. For any $n \geq 1$ one can compute W_n in $O(wn \log \min(\frac{n}{w}, \beta L_n))$ time and using at most $O(\min(n, w\beta L_n))$ memory ($n \geq 1$).*

This improves and generalizes the $O(n \log n)$ bound for $w = 1$ arising from the algorithm of Jacobson and Vo [21]. Moreover, the algorithm proceeds fully online. Interestingly, the bounds are almost equal to those for the Longest Ascending Subsequence Problem, except for a factor that is determined by the 'amplitude' of the set of weights.

6 Discussion

The algebraic framework of sequence measures and maximizers gives us a general way to deal with problems like computing longest or heaviest increasing subsequences, with a unified methodology. The framework appears to capture all underlying properties and capitalizes on them in the design of a general

online algorithm for computing the values of any maximizer as a sequence is input.

The framework enabled us to solve the Online Maximum Ascending Subsequence Problem in a very general way, for the general class of admissible measures and sequences of inputs from any poset of finitely bounded width w that can be effectively represented. It not only generalizes the results for the familiar Longest and Heaviest Increasing Subsequence Problems to a broader setting, but also elucidates the principles on which their solution is based.

It is likely that further scrutiny could lead to further improvements. For example, we have only derived worst case bounds. From the analysis of concatenable queues [38] we know that amortized bounds can be much more favorable. Also, we have desisted from any further assumptions on the range of the values that are stored. If a fixed universe could be assumed, then we could use *van Emde Boas trees* and get $\log \log n$ bounds where we now have single-logarithmic bounds.

Acknowledgement We thank Erik Jan van Leeuwen (Utrecht) for several useful comments on an early version of this paper.

References

1. A.V. Aho, J.E. Hopcroft, J.D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, Reading, MA, 1974.
2. M.R. Alam, M.S. Rahman, A divide and conquer approach and a work-optimal parallel algorithm for the LIS problem, *Inf. Proc. Lett.* 113:13 (2013) 470-476.
3. M.H. Albert, A. Golynskib, A.M. Hamel, A. López-Ortiz, S. Srinivasa Raob, M.A. Safari, Longest increasing subsequences in sliding windows, *Theoretical Computer Science* 321 (2004) 405-414.
4. D. Aldous, P. Diaconis, Longest increasing subsequences: from patience sorting to the Baik-Deift-Johansson theorem, *Bull. AMS* 36 (1999) 413-432.
5. J. Baik, P. Deift, K. Johansson, On the distribution of the length of the longest increasing subsequence of random permutations, *J. AMS* 12:4 (1999) 1119-1178.
6. S. Bespamyatnikh, M. Segal, Enumerating longest increasing subsequences and patience sorting, *Inf. Proc. Lett.* 76 (2000) 711.
7. G.S. Brodal, K. Kaligosi, I. Katriel, M. Kutz, Faster algorithms for computing longest common increasing subsequences, *Technical Report* BRICS-RS-05-37, BRICS, Aarhus University, Aarhus, 2005.
8. C. Cérin, C. Dufourd, J.F. Myoupo, An efficient parallel solution for the longest increasing subsequence problem, *5th International Conference on Computing and Information* (ICCI'93), Proceedings, IEEE Press, pp. 220-224.
9. E. Chen, L. Yang, H. Yuan, Longest increasing subsequences in windows based on canonical antichain partition, *Theoretical Computer Science* 378 (2007) 223-236.
10. M. Crochemore, E. Porat, Fast computation of a longest increasing subsequence and application, *Information & Computation* 208 (2010) 10541059.
11. S. Deorowicz, On some variants of the longest increasing subsequence problem, *Theoretical and Applied Informatics* 21:3-4 (2009) 135-148.
12. S. Deorowicz, A cover-merging-based algorithm for the longest increasing subsequence in a sliding window problem, *Computing and Informatics* 31:6 (2013) 1217-1233.
13. R.P. Dilworth, A decomposition theorem for partially ordered sets, *Annals of Mathematics* 51:1 (1950) 161-165.
14. P. Erdös, G. Szekeres, A combinatorial problem in geometry, *Compositio Mathematica* 2 (1935) 463470.

15. A. Farzan, J. Fischer, Compact representation of posets, in: T. Asano *et al* (Eds.), *Algorithms and Computation - 22nd International Symposium* (ISAAC 2011), Proceedings, Lecture Notes in Computer Science Vol. 7074, Springer-Verlag, 2011, pp. 302-311.
16. M.L. Fredman, On computing the length of longest increasing subsequences, *Discrete Mathematics* 11 (1975) 29-35.
17. M.C. Golumbic, *Algorithmic graph theory and perfect graphs*, Academic Press, New York, 1980.
18. P. Groeneboom, Ulam's problem and Hammersley's process, *The Annals of Probability* 29:2 (2001) 683-690
19. D. Gusfield, *Algorithms on strings, trees, and sequences*, Cambridge University press, Cambridge, 1997.
20. J.W. Hunt, T.G. Szymanski, A fast algorithm for computing longest common subsequences, *C.ACM* 20:5 (1977) 350-353.
21. G. Jacobson, K-P. Vo, Heaviest increasing/common subsequence problems, in: A. Apostolico *et al* (Eds.), *Combinatorial Pattern Matching - Third International Symposium* (CPM'92), Proceedings, Lecture Notes in Computer Science Vol. 644, Springer-Verlag, 1992, pp. 52-66.
22. P. Krusche, A. Tiskin, Parallel longest increasing subsequences in scalable time and memory, in: R. Wyrzykowski *et al* (Eds.), *Parallel Processing and Applied Mathematics* (PPAM 2009), Proceedings, Lecture Notes in Computer Science Vol. 6067, Springer-Verlag, 2008, pp. 176-185.
23. Y. Li, L. Zou, H. Zhang, D. Zhao, Computing longest increasing subsequences over sequential data streams, in: *Proceedings of the VLDB Endowment* 10:3 (2016) 181-192.
24. D. Liben-Nowell, E. Vee, A. Zhu, Finding longest increasing and common subsequences in streaming data, *J. Comb. Optim.* 11:2 (2006) 155-175.
25. E. Mäkinen, On the longest upsequence problem for permutations, *Int. Journal of Computer Mathematics* 77 (2001) 45-53.
26. C.L. Mallows, Problem 62-2, patience sorting, *SIAM Review* 4:2 (1962) 148-149, see also: Problem 62-2, *SIAM Review* 5:2 (1963) 375-376.
27. C.L. Mallows, Patience sorting. *Bull. IMA* 9 (1973) 216-224.
28. K. Mehlhorn, Arbitrary weight changes in dynamic trees, *R.A.I.R.O. Informatique théorique/Theoretical Informatics* 15:3 (1981) 183-211.
29. J.F. Myoupo, D. Semé, Efficient parallel algorithms for the LIS and LCS problems on BSR model using constant number of selections, *Parallel Algorithms and Applications* 14:5 (2000) 187-202.
30. T. Nakashima, A. Fujiwara, Parallel algorithms for patience sorting and longest increasing subsequence, in: J. Li, K. Kato, H. Kameda (Eds.), *Networks, Parallel and Distributed Processing, and Applications* (NPDPA 2002), Proceedings, 2002, pp. 368-374.
31. M. Orlowski, M. Pachter, An algorithm for the determination of a longest increasing subsequence in a sequence, *Computers & Mathematics with Applications* 12:7 (1989) 1073-1075.
32. P. Ramanan, Tight $\Omega(nlgn)$ lower bound for finding a longest increasing subsequence, *International Journal of Computer Mathematics* 65:3-4 (1997) 161-164.
33. D. Romik, *The surprising mathematics of longest increasing subsequences*, IMS textbooks, Cambridge University Press, Cambridge, 2014.
34. T. Sakai, J. Urrutia, On the heaviest increasing or decreasing subsequence of a permutation, and paths and matchings on weighted point sets, in: A. Márquez *et al* (Eds.), *Computational Geometry*, XIV Spanish Meeting (EGC 2011), Lecture Notes in Computer Science Vol. 7579, Springer-Verlag, 2012, pp. 175-184.
35. C. Schensted, Longest increasing and decreasing subsequences, *Canad. J. Math.* 13 (1961) 179-191.
36. D. Semé, S. Youlou, Parallel solutions of the longest increasing subsequence problem using pipelined optical bus systems, in: H.R. Arabnia, J. Ni (Eds.), *International Conference on Parallel and Distributed Processing Techniques and Applications* (PDPTA'04), Proceedings, Vol. 2, CSREA Press, 2004, pp. 617-622.
37. D. Semé, A CGM algorithm solving the longest increasing subsequence problem, in: M.L. Gavrilova *et al* (Eds.), *Computational Science and Its Applications* (ICCSA 2006), Proceedings, Lecture Notes in Computer Science Vol. 3984, Springer-Verlag, 2006, pp. 10-21.

38. D.D. Sleator, R.E. Tarjan, Self-adjusting binary search trees, *J.ACM* 32 (1985) 652-686.
39. M.J. Steele, Variations on the monotone subsequence theme of Erdös and Szekeres, in: D. Aldous *et al*, *Discrete Probability and Algorithms*, Springer-Verlag, New York, 1995, pp. 111-131.
40. X. Sun, D.P. Woodruff, The communication and streaming complexity of computing the longest common and increasing subsequences, in: *18th Annual ACM-SIAM Symposium on Discrete algorithms* (SODA'07), Proceedings, SIAM, 2007, pp. 336-345.
41. C-T. Tseng, S-H. Shiau, C-B. Yan, An optimal algorithm for finding the longest increasing subsequence of every substring, *5th Conference on Information Technology and Applications in Outlying Islands*. Proceedings, Taiwan, 2006. p. 14.
42. C-T. Tseng, C-B. Yang, H-Y. Ann, Minimal height and sequence constrained longest increasing subsequence, *Journal of Internet Technology* 10:2 (2009) 173-178.
43. P. van Emde Boas, Preserving order in a forest in less than logarithmic time and linear space, *Inf. Proc. Lett.* 6 (1977) 8082.