AWLCO: All-Window Length Co-Occurrence

Joshua Sobel
Department of Computer Science, University of Rochester

Noah Bertram
Department of Mathematics, University of Rochester

Chen Ding
Department of Computer Science, University of Rochester

Fatemeh Nargesian
Department of Computer Science, University of Rochester

Daniel Gildea
Department of Computer Science, University of Rochester

Abstract

Analyzing patterns in a sequence of events has applications in text analysis, computer programming, and genomics research. In this paper, we consider the *all-window-length* analysis model which analyzes a sequence of events with respect to windows of all lengths. We study the exact co-occurrence counting problem for the all-window-length analysis model. Our first algorithm is an offline algorithm that counts all-window-length co-occurrences by performing multiple passes over a sequence and computing single-window-length co-occurrences. This algorithm has the time complexity $O(n)$ for each window length and thus a total complexity of $O(n^2)$ and the space complexity $O(|I|)$ for a sequence of size n and an itemset of size $|I|$. We propose `AWLCO`, an online algorithm that computes all-window-length co-occurrences in a single pass with the time complexity of $O(n)$ and space complexity of $O(\sqrt{n|I|})$, assuming perfect hashing. Following this, we generalize our use case to patterns in which we propose an algorithm that computes all-window-length co-occurrence with time complexity $O(n|I|)$, assuming perfect hashing, with an additional pre-processing step and space complexity $O(\sqrt{n|I|} + |I|)$, plus the overhead of the Aho-Corasick algorithm [3].

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Itemsets, Data Sequences, Co-occurrence

Digital Object Identifier 10.4230/LIPIcs.CPM.2021.8

Related Version *Full Version*: `https://arxiv.org/abs/2011.14460`

1 Introduction

Analyzing regularities in streams and event sequences has applications in data analytics as well as programming languages, natural language processing, and genomics. Examples of an event sequence include a sequence of system logs, memory requests by a program, tweets by a user, a series of symptoms, a sequence of words in a document, or an RNA sequence. One metric of regularity is *co-occurrence* [13, 21] — the number of times that an entire set of items or more broadly of patterns is contained within a sliding window of an arbitrary size. For example, consider the sequence "*abccba*" and window size three. This sequence of events contains four such windows: "*abc*", "*bcc*", "*ccb*", and "*cba*". We see that both "*a*" and "*b*" appear together in two windows. Thus, itemset $\{a, b\}$ co-occurs twice for window size of three. In the sequence "*cat dog cat*" with window size seven, we see that the words, referred to as patterns, "*cat*" and "*dog*" both appear as substrings in two windows, and thus the pattern set $\{cat, dog\}$ has co-occurrence of two with window size seven.

Most applications assume that the window is given by a user or defined in an ad hoc manner. Existing counting algorithms for streams often assume the *sliding-window* model of computation, that is, answering queries or mining is done over the last w most recent data elements [6, 7]. Successful pattern-searching tools, such as *ShapeSearch*, enable the search for desired patterns within a fixed window size in trendlines [20]. However, in certain applications of co-occurrence analysis, the query is about identifying the time windows that satisfy certain conditions on the co-occurrence. For instance, in text analysis, what is the time window in which a set of events are very likely to appear? Or, at which time window does the co-occurrence of a set of words in a document become random? Or, how often do two or multiple gene expression patterns co-occur in an RNA sequence? These applications require the analysis of all possible window lengths, possibly as large as the size of the sequence.

The All-Window-Length Analysis Model In this paper, we consider a new analysis model of computation for streams and sequences, the *all-window-length analysis model*, where the analysis of a sequence of data elements is done in *one pass* for all window lengths, starting from the size of a pattern up to the size of a sequence. Unlike single-window-length analysis, in this model, window length becomes a variable. We consider the co-occurrence counting of items and patterns in this analysis model. A pattern is a string with characters drawn from alphabet $\mathbb{A}$. Given a sequence T of size n, and an itemset I consisting of patterns, find the number of windows in which every pattern in I occurs for all window lengths $x \in \{1, \ldots, n\}$ in T. This model enables us to perform analysis without apriori knowledge of window-size, i.e., a window size can be chosen and analyzed on demand at query time. For a sequence T of size n and an itemset I consisting of $|I|$ unique tokens, the co-occurrence analysis considers $\sum_{x=1}^{n}(n-x+1)$ windows. We propose efficient exact algorithms and theoretical analysis for the co-occurrence counting of sets of items and patterns under this analysis model. Note that this analysis model is different from the setting of counting frequent itemset in a stream, in which data elements arrive in baskets of arbitrary lengths, and the goal is to find the itemset that appears in s fraction of the baskets, where s is a support threshold [1, 2, 14, 17].

Applications We expect the all-window-length analysis model to open research opportunities that lead to solving problems in natural language processing, the optimization of the memory layout of programs, and accelerating the search for RNA sequences in genomes. In natural language processing, the co-occurrence of words within a sliding window is the basis for training word embeddings, which are vector representations of a word's meaning and usage [15, 16]. Different window sizes are useful for different purposes; embeddings derived from smaller windows tend to represent syntactic information while larger windows represent semantic information [18]. Identifying an effective window length for training word embeddings requires the efficient exploration of the relationship between window size and co-occurrence frequency of words [12].

The application of all-window-length co-occurrence analysis in programming languages is in the optimization of the memory layout of programs. Modern processor performance is dependent on cache performance and cache block utilization. A set of data elements belong to the same *affinity group* if they are always accessed close to each other. This closeness is defined by k-linkedness. A reference affinity forms a unique partition of data for every k, and the relation between different ks is hierarchical, meaning the affinity groups at link length k are a finer partition of the groups at $k+1$. Reference affinity has been used to optimize the memory layout in data structure splitting [24], whole-program code layout [10], and both [22]. Finding affinity groups requires the analysis of the access co-occurrence of data elements in memory access traces for all ks.

Research has shown that analyzing nucleotide co-occurrence over the entire human genome

provides a powerful insight into the evolution of viruses [8, 19]. Co-occurrence is a method for tracking cooperative genomic interactions as a major force underlying virus evolution. Existing co-occurrence network construction tools such as `cooccurNet` [25] consider pairs of nucleotides or amino acids for analysis and apply filters on the significance of the co-occurrence of genes. The distance in a co-occurrence network counts for the relatedness of genes. An all-window-length analysis of the co-occurrence gene sequences provides further insight into pattern analysis in genomics.

Results In this paper, we propose an efficient algorithm that computes all-window-length exact co-occurrence of patterns in a single pass. For co-occurrence of itemsets of size one or two, our past work proposed a linear time algorithm (in sequence length) to compute co-occurrence for all window lengths [13]. To analyze co-occurrence, first, we introduce an algorithm to calculate co-occurrence that runs in $O(n)$ time, is easily understood, and uses $O(|I|)$ space for single-window-length co-occurrence, where n is the length of the sequence, and I is the set of co-occurring items. However, to find the co-occurrence across all window lengths, the algorithm would require to compute the co-occurrence for each window length separately and use $O(n^2)$ time, which is impractical for large datasets.

We propose `AWLCO`, a time- and space-efficient algorithm that computes the exact co-occurrence of itemsets for all window lengths, in a single pass. The algorithm computes co-occurrence by finding *gaps* in the sequence, or substrings of the sequence that do not contain subsets of the queried pattern. This is a novel approach to compute co-occurrence and provides an improved algorithm, since the stored gaps are not bound to any window lengths. Thus, the collection of gaps allows the co-occurrence to be determined for all window lengths in a single pass through the gaps. Furthermore, we propose a simple approach for computing all of the gaps for an itemset in a single pass through the sequence. The relevant gaps can be found by iterating through the sequence and keeping track of the items and the orders they last appeared. We theoretically prove that gaps are only relevant and counted if the current item encountered in the sequence is the item that was seen furthest in the past, thus, drastically reducing the amount of space and updates needed. `AWLCO` enables all-window-length queries in $O(n)$ time by using $O(\sqrt{n|I|})$ additional space, assuming a perfect hashing function.

Finally, we generalize our problem to finding the co-occurrence of a set of patterns. We argue that finding an algorithm that handles multiple elements at the same index of a sequence would solve all window length pattern co-occurrence. We present an algorithm for pattern co-occurrence counting with time complexity $O(n|I|)$, assuming perfect hashing, and space complexity $O(\sqrt{n|I|} + |I|)$, with additional space overhead from the Aho-Corasick algorithm [3].

2 Problem Definition

We begin by fixing a vocabulary $\mathbb{A}$ that we will be working in. Let T be a sequence with elements in $\mathbb{A}$. Sequence T can be considered as a stream. Let n be the length of the sequence T and for any natural number l, let $[l] = \{1, \ldots, l\}$. A sequence will have its indices zero indexed, i.e., $T[0]$ is the first element that appeared in the sequence and $T[i]$ is the element that appeared at position i. We use $T[i \ldots j]$ to denote a sub-string of T. For example, $T[0 \ldots j]$ indicates the first $j + 1$ elements of sequence T. An itemset I is a finite non-empty subset of $\mathbb{A}$. For a sequence T, a window is a sub-string of T, or a contiguous selection of elements of T. For sequence T we define the window at index i of length x where $x \leq i + 1$, $\omega(T, i, x)$, to be the window containing the i-th element of T and the $x - 1$ previous elements

of T. When it is clear what sequence is being referenced, we will refer simply to $\omega(i, x)$. For example, for the sequence T ="*abcdef*", $\omega(3, 3)$ is "*bcd*". We define the co-occurrence count as the number of windows of length x in sequence T that contain all elements of the itemset I.

▶ **Definition 1.** Single-window length co-occurrence problem*: Given a sequence T and an itemset I, find the co-occurrence count of itemset I in windows of length x in sequence T.*

$$\textit{co-occurrence}\,(T, I, x) = |\{\omega(i, x) \,:\, i \in \{x-1, \ldots, n-1\}, \forall e \in I, e \in \omega(i, x)\}| \quad (1)$$

▶ **Example 2.** Consider the sequence T="abcabe". The co-occurrence count of itemset $\{a, b\}$ in all windows with size four, $\textit{co-occurrence}\,(abcabe, \{a, b\}, 4)$, is three.

In this paper, we consider the new problem of finding co-occurrence counts of I in T for all window lengths.

▶ **Definition 3.** All-window length co-occurrence problem*: Given a sequence T of size n, and an itemset I, find the co-occurrence counts of itemset I in all windows of lengths $x \in \{|I|, \ldots, n\}$ in sequence T.*

In Section 3, we define a baseline algorithm for finding all window length co-occurrence counts based on finding the single window length co-occurrence count. In Section 4, we describe our algorithm for simultaneously finding co-occurrence counts of all window lengths in linear time in the length of the sequence assuming perfect hashing and with space complexity of $O(\sqrt{n|I|})$.

A pattern is a string with characters drawn from alphabet $\mathbb{A}$. A pattern e's ith component is denoted $e[j]$ and the length of the pattern is $|e|$. A pattern occurs in a sequence T if there exists $j \in \{0, ..., n\}$ such that for all $i \in \{0, \ldots, |e| - 1\}$, $T[j + i] = e[i]$.

▶ **Definition 4.** All-window length pattern co-occurrence problem*: Given a sequence T of length n, and an itemset I consisting of patterns, find the number of windows in which every pattern in I occurs for all window lengths $x \in \{1, \ldots, n\}$ in sequence T.*

3 Single-Window-Length Co-occurrence

Consider an item $e \in \mathbb{A}$ and a sequence T. The time elapsed since last access of e at index i, $\text{tesla}(T, e, i)$, is the difference between i and the greatest index where e occurs in T up to and possibly including i, and, in the case that there is no occurrence of e in the interval up to i, we define it to be ∞. When the choice of T is clear we use the shorthand $\text{tesla}(e, i)$ instead. There is a direct connection between the tesla values for items in the itemset and the number of times the items of the itemset co-occur.

▶ **Lemma 5.** *Itemset I co-occurs in a window $\omega(i, x)$ if and only if* $\max\{\text{tesla}(e, i) | e \in I\} < x$.

Proof. The statement implies that for each $e \in I$, $\text{tesla}(e, i) < x$, which implies that $e \in \omega(i, x)$. Conversely, if each $e \in \omega(i, x)$, then we have $\text{tesla}(e, i) < x$; therefore, we have $\max\{tesla(e, i) | e \in I\} < x$. ◀

▶ **Example 6.** Consider the sequence T ="*abcabe*" and itemset $\{a, b\}$. Suppose we have processed $T[0 \ldots 3]$ and we know $\text{tesla}(a, 3) = 0$ and $\text{tesla}(b, 3) = 2$. Since the max tesla value is two, the itemset does not co-occur in the size two window $\omega(3, 2)$.

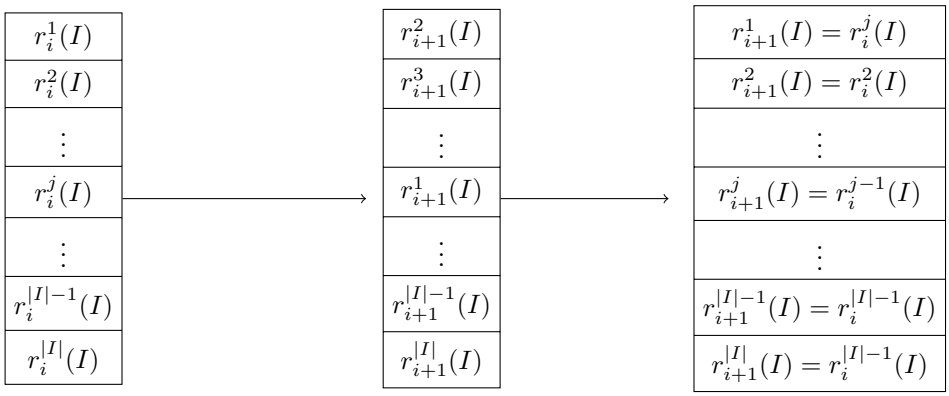

Figure 1 The book-stack, when $T[i+1] = r_i^j(I)$. This change is shown in the first two book-stacks. The third reflects the book-stack at index $i+1$ after it has been updated.

By the lemma, the co-occurrence defined in Equation 1 can be computed by iterating through each index of the sequence and counting the number of times $\max\{\text{tesla}(e,i)|e \in I\} < x$.

$$\textit{co-occurrence}\,(T,I,x) = |\{i \in \{x-1,\ldots,n-1\} : \max\{\text{tesla}(e,i)|e \in I\} < x\}| \quad (2)$$

Book Stack. We now wish to have a systematic way of ordering items according to their corresponding time elapsed since last access. Let Q denote the set of non-empty subsets of $\mathbb{A}$. Let A be some element of Q and suppose that $A = \{e^1, e^2, \ldots, e^{|A|}\}$, and they are labeled in such a way that at index i in our sequence,

$$\text{tesla}(e^1, i-1) \leq \text{tesla}(e^2, i-1) \leq \cdots \leq \text{tesla}(e^{|A|}, i-1).$$

Now let $r_i^j : Q \to \mathbb{A}$ be given by $r_i^j(A) = e^j$, for $j \in \{1, ..., |A|\}$. That is to say that, r arranges the members of A in a finite sequence according to $\text{tesla}(\cdot, i-1)$. This notation is robust as it allows for weak ordering and will be used to consider a generalized case later on. We call the realization of r_i^j a *book-stack*, i.e., $\mathcal{S}_i = [(r_i^1(I), \text{tesla}(r_i^1(I), i)), \ldots, (r_i^{|I|}(I), \text{tesla}(r_i^{|I|}(I), i))]$ based on the above ordering, given a set A. We define $\mathcal{S}_i.\texttt{retrieve}(j) = \text{tesla}(r_i^j, i)$. We define $\mathcal{S}_i.\texttt{find} : A \to \{1, \ldots, |A|\}$, such that $\texttt{find}(a) = j$, where $r_i^j(A) = a$. We define $\mathcal{S}_i.\texttt{update} : \{1, \ldots, |A|\} \to \times_{l=1}^{|A|} A$, in which $\mathcal{S}_i.\texttt{update}(j) = (r_{i+1}^1(A), \ldots, r_{i+1}^{|A|}(A))$, where we have

$$r_{i+1}^l(A) = \begin{cases} r_i^j(A), & l = 1 \\ r_i^{l+1}(A), & 1 \leq l < j \\ r_i^j(A), & j < l \leq |A|. \end{cases}$$

We therefore define $\mathcal{S}_{i+1} = \mathcal{S}_i.\texttt{update}(\mathcal{S}_i.\texttt{find}(T[i]))$. It is straightforward to see that the `update` guarantees the correct ordering for r_{i+1}^j based on $\text{tesla}(\cdot, i)$. Figure 1 illustrates `update` to a book-stack data structure step by step. By an abuse of notation, in our algorithms we refer to $\mathcal{S}_i$ with $\mathcal{S}$.

Algorithm 1, `SINGLECOUNTING` demonstrates co-occurrence count for a specific window length. The co-occurrences of an itemset can be calculated for multiple window lengths by repeating Algorithm 1 and varying the argument x.

Algorithm 1 SINGLECOUNTING

Input: Sequence T of length n, Itemset I, Window Length x
Result: *co-occurrence*(T, I, x)

```
count←0
S ←empty book-stack
for each item e ∈ I do
    S += (e, −∞)
end
for i = 0 to n − 1 do
    if T[i] ∈ I then
        j ← S.find(T[i])
        S.update(j)
    end
    if i ≥ x − 1 and i - S.retrieve(|I|)< x then
        count←count+1
    end
end
return count
```

Table 1 Book Stack changes for single-window co-occurrence counting.

initial	a (i=0)	b (i=1)	c (i=2)	a (i=3)	b (i=4)	e (i=5)
max tesla	$0-(-\infty)=\infty$	$1-0=1$	$2-0=2$	$3-1=2$	$4-3=1$	$5-3=2$
$a(-\infty)$	$a(0)$	$b(1)$	$b(1)$	$a(3)$	$b(4)$	$b(4)$
$b(-\infty)$	$b(-\infty)$	$a(0)$	$a(0)$	$b(1)$	$a(3)$	$a(3)$

▶ **Example 7.** Consider the sequence $T = abcabe$ and the itemset $I = \{a, b\}$. The algorithm initializes the $\mathcal{S}$ by adding $(e, -\infty)$ for each item e in I, representing that element e has never been seen. Table 1 shows the state of $\mathcal{S}$ and the resultant max tesla value every time an element of T is processed. At any step the max tesla value can be found by taking the current index in the sequence and subtracting the last access time of the item in the bottom of the book-stack.

3.1 Complexity Analysis

The book-stack can be implemented as a doubly linked list of items. Finding elements on the bottom of the book-stack can then be done in constant time. We can maintain a hash table from each element to the corresponding node in the book-stack. Each node can be accessed in constant time. The book-stack will only take $|I|$ space and no additional space is needed, thus the total space is $O(|I|)$. In addition, each element of the sequence is accessed once, and only constant time operations are performed, giving a time complexity of $O(n)$. For co-occurrence of a single window length, this algorithm performs optimally with respect to time complexity. This is because there is an intrinsic linear cost in computing co-occurrence, as each element in the sequence must be examined in the worst case. In the next section, we present a solution that in linear time can calculate the co-occurrence for all window lengths.

Figure 2 Gaps for certain elements in a sequence. The uppermost pattern illustrates the three gaps 'a'-gaps, the middle pattern shows the 'b'-gaps, and the bottom pattern shows the three gaps that contain neither 'a' nor 'b'.

4 All Window-Length Co-occurrence

4.1 Counting Co-occurring Windows

To find the co-occurrence of an itemset $I = \{e_1, e_2, \ldots, e_{|I|}\}$ in sequence T with window length x we must count how many x-length windows in T contain I. We will make use of the fact that counting the windows containing I is equivalent to counting the windows that do not contain I, since we know the total number of x-length windows is $n - x + 1$. For a sequence T of length n and an itemset $\{e_1\}$ we denote the x-length windows that do not contain e_1 as $\overline{\{e_1\}}_x$. For larger itemsets we extend the notation analogously where $\overline{\{e_1, e_2\}}_x$ are the x-length windows that do not contain e_1 and do not contain e_2. A window is a *non co-occurrent* window as long as there is at least one element in I that is not contained in the window. Therefore, the co-occurrence of I is the total number of x-length windows minus the number of x-length windows that do not contain at least one item of I.

$$\textit{co-occurrence}\,(T, I, x) = (n - x + 1) - |\overline{\{e_1\}}_x \cup \ldots \cup \overline{\{e_{|I|}\}}_x| \tag{3}$$

Using the inclusion-exclusion principle we can rewrite the co-occurrence as follows.

$$\textit{co-occurrence}\,(T, I, x) = (n - x + 1) - \sum_{\substack{A \subseteq I: \\ A \neq \emptyset}} (-1)^{|A|+1} |\overline{A}_x| \tag{4}$$

▶ **Example 8.** Consider again figure 2 with $I = \{a, b\}$ and $x = 2$. We have that $\overline{\{a\}}_2 \cup \overline{\{b\}}_2 = \{w(i, 2) : 1 \leq i \leq 3, i = 5, 7 \leq i \leq 10\}$, hence $\left|\overline{\{a\}}_2 \cup \overline{\{b\}}_2\right| = 8$. The nonempty subsets of I are $\{a, b\}$, $\{a\}$, and $\{b\}$. We have that $\overline{\{a, b\}}_2 = \{w(8, 2)\}$, $\overline{\{a\}}_2 = \{w(3, 2), w(7, 2), w(8, 2)\}$, and $\overline{\{b\}}_2 = \{w(i, 2) : 1 \leq i \leq 2, i = 5, 8 \leq i \leq 10\}$. This means that $\left|\overline{\{a\}}_2 + \overline{\{b\}}_2 - \overline{\{a, b\}}_2\right| = 8$, agreeing with $\left|\overline{\{a\}}_2 \cup \overline{\{b\}}_2\right|$. The idea here is that when using $\overline{\{a\}}_2$ and $\overline{\{b\}}_2$ to count $\overline{\{a\}}_2 \cup \overline{\{b\}}_2$ we double count $w(8, 2)$, so we need to subtract by 1 to get the correct result.

We know that an A-gap of size k contains $k - x + 1$ windows of length x in which none of A occurs. Thus, $|\overline{A}_x| = \sum_{k=x}^{n} (k - x + 1) N_A(k)$, where $N_A(k)$ is the number of A-gaps of length k. Working with the right term of equation (4),

$$\sum_{\substack{A \subseteq I: \\ A \neq \emptyset}} (-1)^{|A|+1} |\overline{A}_x| = \sum_{\substack{A \subseteq I: \\ A \neq \emptyset}} (-1)^{|A|+1} \sum_{k=x}^{n} (k - x + 1) N_A(k) \tag{5}$$

$$= \sum_{k=x}^{n} (k - x + 1) \sum_{\substack{A \subseteq I: \\ A \neq \emptyset}} (-1)^{|A|+1} N_A(k). \tag{6}$$

Now, let us define:

$$H[k] = \sum_{\substack{A \subseteq I: \\ A \neq \emptyset}} (-1)^{|A|+1} N_A(k) \tag{7}$$

We call the collection of $H[k]$'s for all values of k a *gap histogram*, H. The co-occurrence of I in x-length windows of sequence T is then calculated as follows.

$$\textit{co-occurrence}\,(T, I, x) = (n - x + 1) - \sum_{k=x}^{n} (k - x + 1) H[k] \tag{8}$$

An elegant property of this equation is that by storing the cumulative counts in a gap histogram we can simultaneously calculate the co-occurrence for all window lengths.

▶ **Example 9.** Return to the sequence shown in Figure 2 for $I = \{a, b\}$ and $x = 2$. Because there are two length one $\{a, b\}$-gaps and one length one $\{b\}$-gap, we obtain $H[1] = -2+1 = -1$. Additionally we have one length two $\{a, b\}$-gap, one length two $\{a\}$-gap, and one length two $\{b\}$-gap which nets $H[2] = 1 + 1 - 1 = 1$. There is one length three $\{a\}$-gap and one length three $\{b\}$-gap, so $H[3] = 1 + 1 = 2$. Finally, there is a single length four $\{b\}$-gap so $H[4] = 1$. To summarize, $H[i] = 1$ for $i = 2, 4$, $H[3] = 2$, $H[1] = -1$, and $H[i] = 0$ otherwise.

Using gap histograms to store cumulative counts has a space complexity of $\sqrt{n|I|}$. In Theorem 12, we will formally discuss the space complexity in more detail. Calculating the co-occurrence from the gap histogram instead of directly counting co-occurrent windows is beneficial since calculating gaps does not require a window length as input and yet the gap information is still sufficient to easily calculate the co-occurrence for all window lengths. Thus, all that is needed to calculate all window length co-occurrence is an algorithm to generate the gap histogram.

The simplest way to generate the gap histogram is to iterate through the sequence, keeping track of where gaps begin and end. Whenever an item in the given itemset is found at some index i it marks the end of a gap for any subset of I containing that item and also marks the beginning of a new gap spanning from $T[i + 1...k - 1]$, where k is either the index of the next occurrence of an element in the subset of I in the sequence or n if another element does not occur before the end of the sequence. Note that if an element in I occurs in two adjacent indices in the sequence (i and $i + 1$), we obtain the gap $[i + 1, i]$ which we treat as a length 0 gap and discard. The length l of each newly ended gap can be updated in the histogram by either incrementing or decrementing $H[l]$ depending on whether the subset size was odd or even respectively. This approach has run time $O(n2^{|I|})$ and performs poorly for large itemsets. It is inefficient since whenever an item from I is encountered in the sequence, we need to consider $2^{|I|-1}$ subsets of I and update the histogram (subtract or add counts) accordingly. A better approach is presented next.

4.2 Efficient Gap Counting

Since updates to the histogram have negating effects on each other (Equation 7), many of the histogram entries do not change when an item of the itemset is observed in the sequence. It turns out when an item of the itemset is observed in a sequence, we only need to update the histogram for the gaps related to the *first and second least recently seen items* of I. To keep track of the tesla's, as we iterate through the sequence we can maintain a book-stack data structure that contains each item in I along with the time that it last appeared in the sequence, so that the most recently seen item appears at the top of book-stack.

Observe that, when an item e from I is seen in the sequence at index i, a maximal gap representing each subset of I containing e is added to the histogram. Furthermore, for any one of those sets G, the length of the added gap is the minimum tesla value attained by an item in G at index $i-1$. Note that in this context we take $\text{tesla}(e,i) = i$ if the element has not yet been encountered in the sequence. These gaps account for all of the gaps in the sequence except for gaps that include the final element of the sequence, these gaps are handled specially.

For the following theorem, we first provide some notation. Let

$$H_i = (H_i[1], H_i[2], \ldots, H_i[n])$$

be the histogram up to index i in the sequence. This is only for notational convenience. It is important to note that in the actual program, there is only one histogram, rather than n.

▶ **Theorem 10.** *For any $0 < i < n$, suppose $T[i] = r_i^{|I|}(I)$ then*

$$H_i[k] = \begin{cases} H_{i-1}[k] + 1 \text{ if } k = \text{tesla}(r_i^{|I|}(I), i-1) \\ H_{i-1}[k] - 1 \text{ if } k = \text{tesla}(r_i^{|I|-1}(I), i-1) \\ H_{i-1}[k] \text{ otherwise.} \end{cases}$$

If $T[i] \neq r_i^{|I|}(I)$ then $H_i[k] = H_{i-1}[k]$ for all k. In other words, the histogram is only updated when the next element in the sequence is the item that was just at the bottom of the book-stack.

Proof Sketch. We have a maximal gap for every subset A of I containing $T[i]$. The length of this A-gap is $\min_{e\in A} \text{tesla}(e, i-1)$, hence the addition to the histogram from A is $(-1)^{|A|+1}$ to the kth spot where $k = \min_{e\in A} \text{tesla}(e, i-1)$. Suppose $T[i] \neq \arg\max_{e\in I} \text{tesla}(e, i-1)$ i.e., $T[i]$ is not the item seen furthest in the past most recently. There are the same number of even and odd subsets of I in which $T[i] = \arg\min_{e\in A} \text{tesla}(e, i-1)$ hence these subsets contribute no net updates to H. For the remaining subsets, the same argument follows, hence there are no net updates.

Now suppose that $T[i] = \arg\max_{e\in I} \text{tesla}(e, i-1)$. Similar to the above, for each item in I not equal to $r_i^{|I|-1}(I)$ and $T[i]$, there are the same number of even and odd subsets of I in which $T[i] = \arg\min_{e\in A} \text{tesla}(e, i-1)$. But for $r_i^{|I|-1}(I)$ there is but one subset in which this is satisfied, namely, $\{r_i^{|I|-1}(I), T[i]\}$, and there is also one subset in which $T[i]$ satisfies this, $\{T[i]\}$. Therefore we have

$$H_i[\text{tesla}(T[i], i-1)] = H_{i-1}[\text{tesla}(T[i], i-1)] + 1,$$
$$H_i[\text{tesla}(r_i^{|I|-1}(I), i-1)] = H_{i-1}[\text{tesla}(r_i^{|I|}(I), i-1)] - 1.$$

◀

The theorem does not handle the case for H_n, which we now address. The argument is similar to the proof above for H_i with $i < n$, except that $T[i]$ is undefined. All gaps necessarily close at the end of the sequence. This means that $|C_k| = \sum_{\ell=0}^{|I|} \binom{|I|-j}{\ell}$, for all but $j = |I|$. For $j = |I|$ there is but one set for which $r_i^{|I|}(I) = \arg\min_{e\in A} \text{tesla}(e, n)$, namely, $\{r_i^{|I|}(I)\}$. Thus $H_n[k] = H_{n-1}[k]$ for all k except when $k = \text{tesla}(n, i-1)$ in which $H_n[k] = H_{n-1}[k] - 1$.

The incremental updates that we have derived above result in algorithm `AWLCO`, shown in Algorithm 2.

4.3 Complexity Analysis

The next two theorems assume that the histogram can be implemented as a hashtable with perfect hashing. Without perfect hashing the histogram must contain space for all entries from $1 - n$ and thus will be linear in space to maintain a constant run time or constant histogram updates must be sacrificed to obtain a worst case n^2 runtime.

▶ **Theorem 11** (Time Complexity). *The time complexity of all-window length co-occurrence algorithm is linear in the length of the sequence.*

Proof. The algorithm iterates over the sequence once and possibly updates the book-stack and the histogram for each element in the sequence. Since updating the book-stack and updating the histogram are both done in constant time, the generation of the histogram is done in linear time in the length of the sequence. Once a histogram is computed, the co-occurrence for every window length is computed in a linear time by summing the histogram as shown in Equation 8. Thus, the algorithm provides an $O(n)$ method to calculate all window length co-occurrence. ◀

▶ **Theorem 12** (Space Complexity). *The space complexity of the algorithm is $O(\sqrt{n|I|})$ where n is the length of the sequence and $|I|$ is the size of the itemset.*

Proof. Space is used to maintain the book-stack and the histogram. The book-stack will use $O(|I|)$ space. Note that for any item e in the itemset the total length of gaps for $\{e\}$ is at most the length of the sequence. Thus, we have that the sum of all of the lengths of single-item gaps is bounded above by $n|I|$. Furthermore, whenever an item of the itemset is on the bottom of the book-stack a maximum of two new gaps are added to the histogram. The length of the gap associated to the bottom item in the book-stack is equal to the length of a single-item gap. The length of the other gap is bounded above by the length of the first gap. Therefore, the sum of the length of all gaps added to the histogram is bounded above by $2n|I|$. Note the size of the histogram is the number of distinct gap lengths added to it. In the worst case, gaps are greedily added to the histogram such that there is a length $1, 2, \ldots, k$ size gap added. In this case, if the total number of gaps added is k, the total length of the gaps is $\frac{k(k+1)}{2}$. We know that the sum of the gaps length in a histogram is bounded by $2n|I|$. Thus, we have that $\frac{k(k+1)}{2} \leq 2n|I|$. Solving for k, we have that $k^2 + k \leq 4n|I|$ and $k \leq 2\sqrt{n|I|}$. Thus, the total space used is bounded above by $|I| + 2\sqrt{n|I|}$ which gives a space complexity of $O(\sqrt{n|I|})$. Note that this last line is true because we make the assumption $|I| \leq n$. If not the co-occurrence is simply 0 for every window length. ◀

5 Pattern Co-occurrence

We now wish to generalize our algorithm in two ways. The first is to patterns and the second is to a stream in which multiple events can occur at the same index, for which the latter turns out to be a subproblem of the first. Pattern co-occurrence is explained first. We define a pattern as a string with characters drawn from our alphabet $\mathbb{A}$. A pattern e's ith component is denoted $e[i]$ and the length of the pattern is $|e|$. A pattern occurs in a sequence T if there exists $j \in [|T|]$ such that $T[j \ldots j + |e| - 1] = e$, also let all such j be denoted in the set $b(e)$. Thus, pattern co-occurrence for an itemset I is defined as the number of windows in which every pattern in I occurs. We wish to find an algorithm that can compute the co-occurrence for all window lengths in one pass for patterns. It is clear that tesla is no longer well-defined. Let e be a pattern. So define $\text{btesla}(e, i) = i - \max(|b(e) \cap \{0, \ldots, i\}|)$, which is the distance

Algorithm 2 `AWLCO`

Input: Sequence T, ItemSet I
Result: Co-occurrence of all window lengths

$H \leftarrow$ empty histogram
$cooc \leftarrow []$
$\mathcal{S} \leftarrow$ empty book-stack
for $e \in I$ **do**
  $\mathcal{S}$ += (e, $-\infty$)
end
// Read through entire sequence
for $i = 0$ *to* $n - 1$ **do**
  current$\leftarrow T[i]$
  // When element is seen, update bottom two gaps
  if $current \in I$ **then**
    if $\mathcal{S}$.find($current$) $= |I|$ **then**
      $f \leftarrow i - \mathcal{S}$.retrieve($|I|$)
      $s \leftarrow i - \mathcal{S}$.retrieve($|I| - 1$)
      $H[f] \leftarrow H[f] + 1$
      $H[s] \leftarrow H[s] - 1$
    end
    j $\leftarrow \mathcal{S}$.find(current)
    $\mathcal{S}$.update(j)
  end
end
// Final gap from bottom of book-stack
$f \leftarrow i - \mathcal{S}$.retrieve($|I|$)
$H[f] \leftarrow H[f] + 1$
for $x = |I|$ *to* $|T| - |I| + 1$ **do**
  $S_x \leftarrow 0$
  for $k = x$ *to* $|T|$ **do**
    $S_x \leftarrow S_x + (k - x + 1)H[k]$
  end
  $cooc[x] \leftarrow (|T| - x + 1) - S_x$
end
return $cooc$

$T[i-k_1]$		$T[i-k_2]$			$T[i]$
x_1	x_2	x_3	y_3	x_4	x_1
		y_1			x_2
		y_2			x_3
					x_4
					x_5

Figure 3 Illustration of Theorem 13. The set of patterns I consists of $x_1, x_2, \ldots$, which were seen at $T[i]$, and all other patterns $y_1, y_2, \ldots$. Here A is the set $\{x_1, x_2\}$ of patterns seen at $T[i]$ that were last seen further in the past than any of the other patterns $y_1, y_2, \ldots$. We add one to $H[k_1]$, where k_1 is the time elapsed since x_1 was last seen. We subtract one from $H[k_2]$, where k_2 is the time elapsed since y_1 was last seen.

between i and the most recent start of the pattern. If $b(e) \cap \{0, \ldots, i\}$ is empty, then let it be i.

We can use our previous definition of an A-gap for $A \subseteq I$, but the size of an A gap is now found differently. Previously, the size of an A-gap closed at time i would be $\min_{e \in A} \text{tesla}(e, i-1)$, but now it is $\min_{e \in A} \text{btesla}(e, i-1)$, since an A-gap still occurs if all but the tail ends of members of A are within said gap. Supposing that no two patterns in consideration end at the same time, it is easy to see that Theorem 10 still holds in this case, using btesla in place of tesla. Thus finding an algorithm that handles multiple events at the same index would solve all window length pattern co-occurrence as well. We now proceed to solve the problem in the case that multiple events can occur at the same index.

5.1 Multiple Item Co-occurrence

It is now natural to define co-occurrence for sets of items. We let $T[i] \subseteq \mathbb{A}$, rather than just one element of $\mathbb{A}$, for all i. A co-occurring window for some itemset $I \subseteq \mathbb{A}$ is a window in which for all $e \in I$, there exists a set $A \in w(i, x)$ such that $e \in A$. Thus the co-occurrence is the sum of these co-occurring windows. This is the natural extension. We will now present the following theorem relating to the updates of H. Let X_i denote the set of items that occur at $T[i]$.

▶ **Theorem 13.** *Suppose that for all* $a \in I \cap X_i$, $\text{tesla}(a, i-1) < \text{tesla}(e, i-1)$ *for any* $e \in I \setminus X_i$. *Then for any* $1 < i < n$,

$$H_i[k] = \begin{cases} H_{i-1}[k] + 1 \text{ for } k = \text{tesla}(r_i^{|I \setminus X_i|}(I \setminus X_i), i-1) \\ H_{i-1}[k] - 1 \text{ for } k = \text{tesla}(r_i^{|I|}(I), i-1) \\ H_{i-1}[k] \text{ otherwise.} \end{cases}$$

Otherwise, $H_i[k] = H_{i-1}[k]$ *for all* k.

▶ Remark 14. This is a generalization of Theorem 10. Observe that in the case when $|A| = 1$, this reduces to that result. Moreover, when $i = n$ the same update given in Theorem 10 follows.

Proof Sketch. The incremental updates to H correspond to all the subsets of I that contain at least one member of X_i. Weakly order X_i according to tesla. Now let A_j where $j \in [X_i]$, be the set of subsets of I that contains x_j. Let $\langle A \rangle_i$ be the updates corresponding to some

set A. Therefore

$$H_i - H_{i-1} = \sum_{J \subseteq [|K|]} (-1)^{|J|+1} \left\langle \bigcap_{j \in J} A_j \right\rangle.$$

Consider each $\mathcal{K}_J = \bigcap_{i \in J} A_i$. For each one, the update is the same if one removes all members of X_i besides the one corresponding to the smallest number in J, call this set $\mathcal{K}'_J$. Using Theorem 10, the update is $+1$ for $k = \text{tesla}(a, i-1)$, a being the item described before, and also is -1 for $k = \text{tesla}(a, i-1)$ for a being the furthest item seen in the past not in $\mathcal{K}_J \cap I$. But this implies that J that are not of the form $J_m = \{|X_i| - m, \ldots, |X_i|\}$ do not contribute to the update. For any $0 \leq m < |X_i|$, the positive update corresponding to $\mathcal{K}_{J_m}$ cancels with the negative update corresponding to $\mathcal{K}_{J_{m+1}}$. This process telescopes leaving only the positive update corresponding to $\mathcal{K}_{J_0}$ and the negative update corresponding to $\mathcal{K}_{J_{|X_i|}}$. This gives the desired result. ◀

A full proof is given in the appendix. Figure 3 provides an illustration of Theorem 13. With this result we can now construct a similar algorithm to those before, with a few modifications. Maintain a book-stack as before, but notice that it is no longer a strict ordering. For example, if $X_i = \{e_1, e_2\} \subseteq I$, then one of e_1 and e_2 will occupy the top of the book-stack and the other will occupy the second to top spot. To check whether $\max_{e \in X_i \cap I} \text{tesla}(e, i) < \min_{e \in I \setminus X_i} \text{tesla}(e, i)$, we partition the book-stack using $p \in \{0, ..., |I|\}$, where p is defined as follows: for all $j \leq p$, $\text{tesla}(r_i^j(I), i-1) = \text{tesla}(r_i^{|I|}(I), i-1)$, and for all $j > p$, $\text{tesla}(r_i^j(I), i-1) > \text{tesla}(r_i^{|I|}(I), i-1)$. Thus checking if the non-trivial conditions given in Theorem 13 hold is easy as we just check that $r_i^j(I) \leq p$ for every j corresponding to a member in X_i. It is also easy to update the histogram if these conditions hold, as we just update according to $r_j^p(I)$ and $r_j^{p+1}(I)$. The pseudocode is given in Algorithm 3 in the appendix.

5.2 Complexity Analysis

We again consider pattern co-occurrence rather than the simplified multiple item co-occurrence, to find the complexity of the algorithm in general. We complete a pass over the stream as we do in algorithm 2, but with a few additions. At each index we must maintain the partition which in the worst case takes linear scan of I at each index. Maintaining the book-stack then also requires $O(|I|)$ operations. We can use the state machine in the Aho-Corasick algorithm [3] to recognize when a pattern is completed in this same pass. For this addition, the algorithm requires an initial time linear to the sum of the lengths of all of the patterns to construct the necessary finite state machine for Aho-Corasick. Thus, the time complexity is $O(n|I|)$ with an additional pre-processing complexity of $O(\sum_{e \in I} |e|)$ due to Aho-Corasick. Space complexity for the histogram and book-stack remains $O(\sqrt{n|I|} + |I|)$[1], however additional space is now needed for user's desired implementation of the Aho-Corasick algorithm.

6 Related Work

Counting in Streams - In the count-distinct problem, the goal is to know the number of unique elements in a stream [9, 14]. In the bit-counting problem, the goal is to maintain the

[1] When considering patterns rather than single elements, $|I| \leq n$ is not necessarily true, so we include an additional factor of $|I|$ for completeness.

frequency count of 1's in the last k bits of a bit stream of size N. Datar et al. propose an approximate algorithm with for the bit-counting problem with $O(\log^2 k)$ space complexity [6]. Existing counting algorithms for streams assume the *sliding-window* model of computation, that is answering queries or mining is done over the last w elements seen so far [7]. However, `AWLCO` introduces a new analysis model – all-window-length analysis model – which is compelled to analyze and query all windows of all lengths starting from the beginning of a stream or anytime in the past. To that end, `AWLCO` presents an efficient and exact itemset counting algorithm for the all-window-length analysis model.

The frequent itemset mining in stream is a well-studied problem that adheres to the counting problem [5]. The seminal work by Manku and Motwani presents an algorithm for estimating the frequency count of itemsets in a stream and identifying those itemsets that occur in at least a fraction θ of the stream seen so far with some error parameter ϵ [14]. For example, when the input is a stream of transactions where each transaction is a set of items, the goal is to find the most frequent itemsets within transactions. The challenge is to consider variable-length itemsets and avoid the combinatorial enumeration of all possible itemsets. Many existing frequent itemset mining algorithms (with exception of [4, 11]) obtain approximate results with error bounds. A variation of frequent itemset mining is the problem of mining frequent co-occurrence patterns across multiple data streams [21]. The definition of co-occurrence patterns is slightly different than co-occurrence itemsets considered by `AWLCO`. A co-occurrence pattern is a group of items that appear consecutively showing tight correlations between these items. A frequent co-occurrence pattern is the pattern that appears in at least θ streams within a time period of length τ and the appearance of the pattern in each stream happens within a time window of δ or smaller. In this paper, `AWLCO` presents an all-window length frequency counting for a query itemset. A natural extension of the itemset frequency counting of presented by `AWLCO` is mining frequent itemsets in all window-lengths.

Affinity Analysis - Zhong et al. defined *reference affinity* for data elements on an access trace. A set of data elements belong to the same affinity group if they are always accessed close to each other [24]. The closeness is defined by $k-$linked-ness. They proved that reference affinity forms a unique partition of data for every k, and the relation between different ks is hierarchical, i.e. the affinity groups at link length k are a finer partition of the groups at $k+1$. This definition requires *strict* co-occurrence in that every occurrence of a group element must be accompanied by all other elements of the group. *Weak reference affinity* [23] introduces a second parameter, affinity threshold. It adheres to the unique and hierarchical partition properties with respect to both parameters. Zhang et al. showed that neither strict reference affinity nor weak reference affinity can efficiently be computed [22]. Thus they gave a heuristic solution and adapted it to use sampling. The average time complexity of their algorithm is $O(N\delta\omega^2 + N\delta\pi)$, where N is the length of the trace, δ is the sampling rate, ω is the size of the affinity group, and π is the average time length of windows containing accesses to all members of the group ω. Lavaee et al. gave an $O(L\delta\omega^2)$ algorithm to compute the affinity for all sub-groups of sizes up to ω [10]. Reference affinity has been used to optimize the memory layout in data structure splitting [24], whole-program code layout [10], and both [22].

7 Discussion and Future Work

Applications The all-window-length co-occurrence has applications in text analysis, the optimization of the memory layout of programs, and accelerating the search for RNA

sequences in genomes. In terms of practical applications, our plan is to develop interactive tools that enable the exploration of sequences of events and genomics data. Projects such as `cooccurNet` [25] provide a basis that can be extended with all-window-length co-occurrence analysis functionalities.

Mining Problems In this paper, we expounded co-occurrence counting of itemsets and patterns in the all-window-length analysis model. Going forward, we study mining algorithms in this analysis model, including mining frequent closed itemsets, i.e., given a sequence T find the top-k itemsets that have highest co-occurrences in an arbitrary window size and for a frequent itemset X, there exists no super-pattern $X \subset Y$, with the same co-occurrence as X. The algorithm requires to mine frequent itemsets for all window lengths in one pass.

Extending to Timestamped Sequences The proposed algorithms operate on a sequence of data points taken at equally spaced points in time. Thus, our sequences are discrete-time data. We plan to study co-occurrence counting and frequent itemset mining in a series of data points indexed in continuous time order. In the continuous setting, we define $T = \{(e, \omega) : e \in I, \omega \in [0, \tau]\}$, where $\tau \geq 0$, to be a set of timestamps in consideration. We can now consider the co-occurrence of items and patterns over an interval of time, which can now be considered as events not items. We wish to compute the probability of a groups of events happening in time scale r:

$$\Pr(I \in [a, b], T, r) = \Pr_{y \sim U([a,b])}(\forall e \in I, \exists (e, \omega) \in T : |y - \omega| < r).$$

This naturally leads to an analytic definition and suggests a continuous analog of co-occurrence.

Acknowledgements

We would like to give special thanks to Lu Zhang and Katherine Seeman for their efforts for the implementation and experimental evaluation of our algorithms. Partially funded by NSF awards CCF-1717877, IIS-1813823, CCF-1934986 and CNS-1909099.

References

1 Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 207–216, 1993. `doi:10.1145/170036.170072`.

2 Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 487–499, 1994. `doi:10.5555/645920.672836`.

3 Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, June 1975. `doi:10.1145/360825.360855`.

4 Joong Hyuk Chang and Won Suk Lee. Finding recent frequent itemsets adaptively over online data streams. In *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 487–492, 2003. `doi:10.1145/956750.956807`.

5 Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment (PVLDB)*, 1(2):1530–1541, 2008. `doi:10.14778/1454159.1454225`.

6 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002. `doi:10.1137/S0097539701398363`.

7 Mayur Datar and Rajeev Motwani. The sliding-window computation model and results. In *Data Stream Management - Processing High-Speed Data Streams*, pages 149–165. Springer, 2016. doi:10.1007/978-0-387-47534-9_8.

8 Xiangjun Du, Zhuo Wang, Aiping Wu, Lin Song, Yang Cao, Haiying Hang, and Taijiao Jiang. Networks of genomic co-occurrence capture characteristics of human influenza A (H3N2) evolution. *Genome Research*, 18(1):178–187, January 2008. doi:10.1101/gr.6969007.

9 Philippe Flajolet and G. Nigel Martin. Probabilistic counting. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 76–82, 1983. doi:10.1109/SFCS.1983.46.

10 Rahman Lavaee, John Criswell, and Chen Ding. Codestitcher: inter-procedural basic block layout optimization. In *Proceedings of the International Conference on Compiler Construction*, pages 65–75, 2019. doi:10.1145/3302516.3307358.

11 Carson Kai-Sang Leung and Quamrul I. Khan. DSTree: A tree structure for the mining of frequent sets from data streams. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 928–932, 2006. doi:10.1109/ICDM.2006.62.

12 Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *Proceedings of the 52nd Anuual Meeting of the Association for Computational Linguistics (ACL)*, pages 302–308, 2014. doi:10.3115/v1/P14-2050.

13 Yumeng (Lucinda) Liu, Daniel Busaba, Chen Ding, and Daniel Gildea. All timescale window co-occurrence: Efficient analysis and a possible use. In *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*, CASCON '18, pages 289–292, Riverton, NJ, USA, 2018. IBM Corp. doi:10.5555/3291291.3291322.

14 Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 346–357, 2002. doi:10.1016/B978-155860869-6/50038-X.

15 Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013. doi:10.5555/2999792.2999959.

16 Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, 2014.

17 Anand Rajaraman, Jure Leskovec, and Jeffrey D. Ullman. *Mining Massive Datasets*. Cambridge University Press, 2014. doi:10.5555/2124405.

18 Hinrich Schütze. *Ambiguity Resolution in Language Learning – Computational and Cognitive Models*. Number 10 in CSLI Lecture Notes Series. Center for the Study of Language and Information, Stanford, California, 1997.

19 Jason W. Shapiro and Catherine Putonti. Gene co-occurrence networks reflect bacteriophage ecology and evolution. *mBio*, 9(2), 2018. doi:10.1128/mBio.01870-17.

20 Tarique Siddiqui, Paul Luh, Zesheng Wang, Karrie Karahalios, and Aditya G. Parameswaran. Shapesearch: A flexible and efficient system for shape-based exploration of trendlines. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 51–65, 2020. doi:10.1145/3318464.3389722.

21 Ziqiang Yu, Xiaohui Yu, Yang Liu, Wenzhu Li, and Jian Pei. Mining frequent co-occurrence patterns across multiple data streams. In *International Conference on Extending Database Technology (EDBT)*, pages 73–84, 2015. doi:10.5441/002/edbt.2015.08.

22 Chengliang Zhang, Chen Ding, Mitsunori Ogihara, Yutao Zhong, and Youfeng Wu. A hierarchical model of data locality. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 16–29, 2006. doi:10.1145/1111037.1111040.

23 Chengliang Zhang, Yutao Zhong, Chen Ding, and Mitsunori Ogihara. Finding reference affinity groups in trace using sampling method. Technical report, Department of Computer Science, University of Rochester, 2004.

24 Yutao Zhong, Maksim Orlovich, Xipeng Shen, and Chen Ding. Array regrouping and structure splitting using whole-program reference affinity. In *Proceedings of the ACM SIGPLAN*

Conference on Programming Language Design and Implementation, pages 255–266, 2004. doi:10.1145/996841.996872.

25 Yuanqiang Zou, Zhiqiang Wu, Lizong Deng, Aiping Wu, Fan Wu, Kenli Li, Taijiao Jiang, and Yousong Peng. cooccurNet: an R package for co-occurrence network construction and analysis. *Bioinformatics*, 33(12):1881–1882, 2017. doi:10.1093/bioinformatics/btx062.

Appendix

Proof of Theorem 10

Proof. We have a maximal gap for every subset of I containing $T[i]$. This collection of subsets can be written as

$$C = \{A \subseteq I | A = \{T[i]\} \cup B, B \subseteq I \setminus \{T[i]\}\}.$$

For each $A \in C$, the update to H is $(-1)^{|A|+1}$ to $H\left[\text{tesla}(r_i^1(A), i-1)\right]$ as we have found an A-gap of size $\text{tesla}(r_i^1(A), i-1)$ at index i. Let $C_k = \{A \in C | \text{tesla}(r_i^1(A), i-1) = k\}$. The incremental updates can be expressed by

$$H_i[k] = H_{i-1}[k] + \sum_{A \in C_k} (-1)^{|A|+1}, \tag{9}$$

for each k. Suppose $T[i] = r_i^{j_0}(I)$ where $j_0 < |I|$, i.e., $T[i]$ is not the item seen furthest in the past most recently. Then there are $\binom{|I|-j}{\ell}$ sets $A \in C$ of length $\ell + 1$ in which $T[i] = r_i^1(A)$. Thus for $k = \text{tesla}(T[i], i-1)$, we have

$$H_i[k] - H_{i-1}[k] = \sum_{A \in C_k} (-1)^{|A|+1} = \sum_{\ell=0}^{|I|-j} \binom{|I|-j}{\ell} (-1)^\ell = (1-1)^{|I|-j} = 0. \tag{10}$$

Now for every $j < j_0$ (which means that j never equals $|I|-1$ in this case), we have that there are $\binom{|I|-j-1}{\ell}$ members $A \in C$ of length $\ell + 2$ in which, $r_i^j(I) = r_i^1(A)$. Therefore for $k = \text{tesla}(r_i^j(I), i-1)$,

$$H_i[k] - H_{i-1}[k] = \sum_{A \in C_k} (-1)^{|A|+1} = \sum_{\ell=0}^{|I|-j-1} \binom{|I|-j-1}{\ell} (-1)^{(\ell+1)} = -(1-1)^{|I|-j-1} = 0.$$

But for $|I| \geq j > j_0$, there are no such sets $A \in C$ in which $r_i^j(I) = r_i^1(A)$, as $T[i] = r_i^{j_0}(I)$ is contained in all $A \in C$.

But if $j_0 = |I|$, i.e., $T[i] = r_i^{|I|}(I)$, then for each $j < |I|-1$, there are again $\binom{|I|-j-1}{\ell}$ members $A \in C$ of length $\ell + 2$ in which, $r_i^j(I) = r_i^1(A)$, so again equation (10) holds for $k = \text{tesla}(r_i^j(I), i-1)$, giving no net updates for such k. But there is exactly one $A \in C$ in which $r_i^{|I|-1}(I) = r_i^1(A)$, namely, $\{r_i^{|I|-1}(I), T[i]\}$, and there is also one $A \in C$ in which $T[i] = r_i^{|I|}(I) = r_i^1(A)$, which is $\{T[i]\}$. Therefore we have

$$\begin{aligned} H_i[\text{tesla}(r_i^{|I|}(I), i-1)] &= H_{i-1}[\text{tesla}(r_i^{|I|}(I), i-1)] + 1, \\ H_i[\text{tesla}(r_i^{|I|-1}(I), i-1)] &= H_{i-1}[\text{tesla}(r_i^{|I|}(I), i-1)] - 1. \end{aligned}$$

◀

Proof of Theorem 13

Proof. Suppose without loss of generality that $X_i \subseteq I$. We wish to find $H_i - H_{i-1}$. Denote

$$X_i = \{r_i^1(X_i), r_i^2(X_i), \ldots, r_i^{|X_i|}(X_i)\} = \{x_1, x_2, \ldots, x_{|X_i|}\},$$

as r defined before. Now let

$$U_i = \{A \subseteq I : A = B \cup \{x_j\}, B \subseteq I \setminus \{x_j\}, j \in [|X_i|]\},$$

which in words, is all subsets of I that contain at least one member of X_i. Observe that

$$U_i = \bigcup_{j=1}^{|X_i|} \{A \subseteq I : A = B \cup \{x_j\}, B \subseteq I \setminus \{x_j\}\}.$$

Now let $K_j = \{A \subseteq I : A = B \cup \{x_j\}, B \subseteq I \setminus \{x_j\}\}$ for all j. Therefore $U_i = \bigcup_{j=1}^{|X_i|} K_j$.

The update rule is known for each K_j based on our previous result. The remains the of the proof is as follows. We can leverage the update rule currently known to compute the total update. But the intersection of K_j's is non-empty, meaning if we update according to each K_j, we would be overcounting some members of U. Once this is determined, we will find the update rule according for each arbitrary intersection of these K_j's, which completes the proof.

Define $\langle \cdot \rangle_i$ to be a mapping from subsets of I to an integer valued n dimensional vector. $\langle A \rangle_i^k$ is the sum of the number of maximal gaps of length k ending at index i given by even subsets of A, minus the sum of the number of maximal gaps of length k ending at index i given by the odd subsets of A. Using this new definition, $\langle U_i \rangle_i^k = H_i[k] - H_{i-1}[k]$. We can now appeal to the inclusion exclusion principle to write that

$$H_i - H_{i-1} = \langle U_i \rangle = \left\langle \bigcup_{j=1}^{|X_i|} K_j \right\rangle = \sum_{J \subseteq [|X_i|]} (-1)^{|J|+1} \left\langle \bigcap_{j \in J} K_j \right\rangle. \tag{11}$$

The right hand side of the above equality will now be used.

Denote for any $J \subseteq [|X_i|]$,

$$\mathcal{K}_J = \bigcap_{j \in J} K_j.$$

Let $\mathcal{X}_J$ be the set of members of X_i that lie in every member of $\mathcal{K}_J$. Observe that

$$\mathcal{X}_J = \bigcap_{G \in \mathcal{K}_J} G.$$

It also follows that $\mathcal{X}_J = \bigcup_{j \in J} \{x_j\}$. Moreover, we can write

$$\mathcal{K}_J = \{A \subseteq I : A = \mathcal{X}_J \cup B, B \subseteq I \setminus \mathcal{X}_J\}.$$

For each set $A \in \mathcal{K}_J$, there is a corresponding set A' in $\mathcal{K}'_J = \{A \subseteq I : A = \{r_i^1(\mathcal{X}_J)\} \cup B, B \subseteq I \setminus \mathcal{X}_J\}$, in which $\langle A \rangle = (-1)^{|J|+1} \langle A' \rangle$. This correspondence is easy to find. Let $A \in \mathcal{K}_J$. Thus $A = \mathcal{X}_J \cup B$, for some $B \in I \setminus \mathcal{X}_J$. Then the corresponding set $A' \in \mathcal{K}'_J$ is $\{r_i^1(\mathcal{X}_J)\} \cup B$. This is clear, because items that lie in every $A \in \mathcal{K}_J$ that never satisfy $\arg\min_{e \in A} \text{tesla}(e, i-1)$ for all A never contribute towards any updates and hence can be ignored, except they may change the parity of the set and hence change the sign of the update. From here, we can apply the first theorem taking I in that theorem to be $I \setminus \mathcal{X}_J$, which gives

$$\langle \mathcal{K}_J \rangle_i^k = (-1)^{|J|+1} \langle \mathcal{K}'_J \rangle_i^k = (-1)^{|J|+1} \begin{cases} 1 \text{ for } k = \text{tesla}(r_i^{|I \setminus \mathcal{X}_J|}((I \setminus \mathcal{X}_J) \cup \{x^*\}), i-1) \\ -1 \text{ for } k = \text{tesla}(r_i^1(\mathcal{X}_J), i-1) \\ 0 \text{ otherwise,} \end{cases} \tag{12}$$

when $r_i^1(\mathcal{X}_J) = r_i^{|I \setminus \mathcal{X}_J|+1}((I \setminus \mathcal{X}_J) \cup \{r_i^1(\mathcal{X}_J)\})$. Every update is 0 otherwise.

Now assume for all $x \in X_i$ and $e \in I \setminus X_i$, $\text{tesla}(x, i-1) \geq \text{tesla}(e, i-1)$. For if this does not hold for some $x' \in X_i$, then by the above, no updates occur due to x', so analysis is the same.

We now wish to compute the right hand side of equation (11). We can employ equation (12) for each $\mathcal{K}_J$. If $r_i^1(\mathcal{X}_J) \neq r_i^{|I\setminus\mathcal{X}_J|+1}((I \setminus \mathcal{X}_J) \cup \{r_i^1(\mathcal{X}_J)\})$, that is, the first ranked item of $\mathcal{X}_J$ is not ranked below all of $I \setminus \mathcal{X}_J$, then $\langle \mathcal{K}_J \rangle = \mathbf{0}$. We claim that the J in which $\langle \mathcal{K}_J \rangle \neq \mathbf{0}$ are of the following form:

$$J_m = \{|X_i| - b : b \in [m]\}, \tag{13}$$

for $0 \leq m < |X_i|$. We first show that if $J \neq J_m$ for some m, then $\langle \mathcal{K}_J \rangle = \mathbf{0}$. If $J \neq J_m$ for some m, then there exists b_0 such that $r_i^{|X_i|-b_0}(X_i) \notin \mathcal{X}_J$, and there is some b_1 such that $b_1 > b_0$ and $r_i^{|X_i|-b_1}(X_i) \in \mathcal{X}_J$. Since $b_1 \leq |X_i| - 1$, $b_0 < |X_i| - 1$ which gives that $|X_i| - b_0 > 1$. Let c_0 and c_1 be such that $r_i^{c_0}((I \setminus \mathcal{X}_J) \cup \{r_i^1(\mathcal{X}_J)\}) = r_i^{|X_i|-b_0}(X_i)$ and $r_i^{c_1}((I \setminus \mathcal{X}_J) \cup \{r_i^1(\mathcal{X}_J)\}) = r_i^1(\mathcal{X}_J)$. We have that $c_0 > c_1$. Now since $c_0 \leq |I \setminus \mathcal{X}_J| + 1$, $c_1 \neq |I \setminus X_J| + 1$. Therefore $r_i^1(\mathcal{X}_J) \neq r_i^{|I\setminus X_J|+1}((I \setminus \mathcal{X}_J) \cup \{r_i^1(\mathcal{X}_J)\})$, hence $\langle \mathcal{K}_J \rangle = \mathbf{0}$.

Now suppose that $J = J_m$ for some m. Let c_1 be such that $r_i^{c_1}(I) = r_i^1(\mathcal{X}_J)$. We then have that for any $c < c_1$, $c \in J$, moreover, $r_i^c(I) \notin (I \setminus \mathcal{X}_J) \cup \{r_i^1(\mathcal{X}_J)\}$. Thus $r_i^1(\mathcal{X}_J) = r_i^{|I\setminus\mathcal{X}_J|+1}((I \setminus \mathcal{X}_J) \cup \{r_i^1(\mathcal{X}_J)\})$, for if not, then there would be $c_0 > c_1$ in which $r_i^{c_0}(I) \in (I \setminus \mathcal{X}_J) \cup \{r_i^1(\mathcal{X}_J)\}$, a contradiction.

From this, the right hand side of equation (11) becomes

$$\sum_{J \subseteq [|X_i|]} (-1)^{|J|+1} \langle \mathcal{K}_J \rangle = \sum_{m=0}^{|X_i|-1} (-1)^m \langle \mathcal{K}_{J_m} \rangle . \tag{14}$$

We now have for $J = J_m$, $r_i^1(\mathcal{X}_J) = r_i^{|X_i|-m}(X_i)$. Also when $m < |X_i| - 1$, we have that

$$r_i^{|I\setminus\mathcal{X}_J|}(I \setminus \mathcal{X}_J \cup \{r_i^1(\mathcal{X}_J)\}) = r_i^{|I\setminus\mathcal{X}_J|}(I \setminus \mathcal{X}_J) = r_i^{|X_i|-(m+1)}(X_i). \tag{15}$$

But when $m = |X_i| - 1$, $J = [|X_i|]$, therefore

$$r_i^{|I\setminus\mathcal{X}_J|}(I \setminus \mathcal{X}_J \cup \{r_i^1(\mathcal{X}_J)\}) = r_i^{|I\setminus\mathcal{X}_J|}(I \setminus \mathcal{X}_J) = r_i^{|I\setminus X_i|}(I \setminus X_i). \tag{16}$$

Now for $m < |X_i| - 1$, we can rewrite equation (11) to get

$$\langle \mathcal{K}_{J_m} \rangle_i^k = (-1)^m \begin{cases} 1 \text{ for } k = \text{tesla}(r_i^{|X_i|-(m+1)}(X_i), i-1) \\ -1 \text{ for } k = \text{tesla}(r_i^{|X_i|-m}(X_i), i-1) \\ 0 \text{ otherwise.} \end{cases} \tag{17}$$

Now define

$$u(m)_i^k = \begin{cases} 1 \text{ for } k = \text{tesla}(r_i^{|X_i|-(m+1)}(X_i), i-1) \\ -1 \text{ for } k = \text{tesla}(r_i^{|X_i|-m}(X_i), i-1) \\ 0 \text{ otherwise,} \end{cases} \tag{18}$$

for $m < |X_i| - 1$ and

$$u(|X_i| - 1)_i^k = \begin{cases} 1 \text{ for } k = \text{tesla}(r_i^{|I\setminus X_i|}(I \setminus X_i), i-1) \\ -1 \text{ for } k = \text{tesla}(r_i^1(X_i), i-1) \\ 0 \text{ otherwise,} \end{cases} \tag{19}$$

Taking $u(m)_i = (u_i^1(m), u_i^2(m), \ldots, u_i^n(m))$, we can write

$$\sum_{m=0}^{|X_i|-1} (-1)^m \langle \mathcal{K}_{J_m} \rangle = \sum_{m=0}^{|X_i|-1} (-1)^m (-1)^m u(m)_i^k = \sum_{m=0}^{|X_i|-1} u(m)_i^k. \tag{20}$$

Observe that

$$u(0)_i^k + u(1)_i^k = \begin{cases} 1 \text{ for } k = \text{tesla}(r_i^{|X_i|-2}(X_i), i-1) \\ -1 \text{ for } k = \text{tesla}(r_i^{|X_i|}(X_i), i-1) \\ 0 \text{ otherwise.} \end{cases} \tag{21}$$

Applying this for all $m < |X_i| - 1$ gives

$$\sum_{m=0}^{|X_i|-2} u(m)_i^k = \begin{cases} 1 \text{ for } k = \text{tesla}(r_i^1(X_i), i-1) \\ -1 \text{ for } k = \text{tesla}(r_i^{|X_i|}(X_i), i-1) \\ 0 \text{ otherwise.} \end{cases} \tag{22}$$

So combining this with $u(|X_i| - 1)_i^k$, we get

$$\sum_{m=0}^{|X_i|-1} u(m)_i^k = \begin{cases} 1 \text{ for } k = \text{tesla}(r_i^{|I \setminus X_i|}(I \setminus X_i), i-1) \\ -1 \text{ for } k = \text{tesla}(r_i^{|I|}(I), i-1) \\ 0 \text{ otherwise,} \end{cases} \tag{23}$$

since $r_i^{|X_i|}(X_i) = r_i^{|I|}(I)$. Combining equation (23) with equations (20), (14), and (11) (and considering the components of each of those equations), we finally get,

$$H_i[k] - H_{i-1}[k] = \begin{cases} 1 \text{ for } k = \text{tesla}(r_i^{|I \setminus X_i|}(I \setminus X_i), i-1) \\ -1 \text{ for } k = \text{tesla}(r_i^{|I|}(I), i-1) \\ 0 \text{ otherwise,} \end{cases} \tag{24}$$

proving the result ($X_i = A$ in the statement of the theorem). ◀

Algorithms

Here we place pseudocode for PAWLCO.

Algorithm 3 `PAWLCO`

Input: Trace T, ItemSet I
Result: Co-occurrence of all window lengths

$H \leftarrow$ empty histogram, $cooc \leftarrow []$, $\mathcal{S} \leftarrow$ empty book-stack
; **for** $e \in I$ **do**
  $\mathcal{S}$ += (e, $-\infty$)
end
$p \leftarrow |I|$, $m \leftarrow 1$
while $\mathcal{S}$.retrieve$(|I|) = \mathcal{S}$.retrieve$(|I| - m)$ **do**
  $m \leftarrow m + 1$
end
$p \leftarrow |I| - m$
for $i = 0$ *to* $n - 1$ **do**
  $C \leftarrow \{e \in I \mid e[0] = T[i - |e| + 1] ... e[|e| - 1] = T[i]\}$
  min $\leftarrow i$ **for** $c \in C$ **do**
    if $\mathcal{S}$.find$(c) < min$ **then**
      min $\leftarrow \mathcal{S}$.find(c)
    end
  end
  if $min > \mathcal{S}$.retrieve$(p + 1)$ **then**
    $f \leftarrow i - \mathcal{S}$.retrieve$(|I|)$, $s \leftarrow i - \mathcal{S}$.retrieve$(p + 1)$
    $H[f] \leftarrow H[f] + 1$, $H[s] \leftarrow H[s] - 1$
  end
  for *current* $\in C$ **do**
    j $\leftarrow \mathcal{S}$.find(current)
    $\mathcal{S}$.update(j)
    // Maintain partition
    if $j = p = |I|$ **then**
      $m \leftarrow 2$
      while $\mathcal{S}$.retrieve*(/I/-1)* $= \mathcal{S}$.retrieve*(/I/-m)* **do**
        $m \leftarrow m + 1$
      end
      $p \leftarrow |I| - m$
    end
    if $p \leq j < |I|$ **then**
      $p \leftarrow p + 1$
    end
  end
end
// Final gap from bottom of book-stack
$f \leftarrow i - \mathcal{S}$.retrieve(|I|)
$H[f] \leftarrow H[f] + 1$
for $x = 0$ *to* $|T| - |I| + 1$ **do**
  $S_x \leftarrow 0$
  for $k = x$ *to* $|T|$ **do**
    $S_x \leftarrow S_x + (k - x + 1)H[k]$
  end
  $cooc[x] \leftarrow (|T| - x + 1) - S_x$
end
return *cooc*