Mining Top-k Quantile-based Cohesive Sequential Patterns

Len Feremans* Boris Cule* Bart Goethals*†

Abstract
Finding patterns in long event sequences is an important data mining task. Two decades ago research focused on finding all frequent patterns, where the anti-monotonic property of support was used to design efficient algorithms. Recent research focuses on producing a smaller output containing only the most interesting patterns. To achieve this goal, we introduce a new interestingness measure by computing the proportion of the occurrences of a pattern that are cohesive. This measure is robust to outliers, and is applicable to sequential patterns. We implement an efficient algorithm based on constrained prefix-projected pattern growth and pruning based on an upper bound to uncover the set of top-k quantile-based cohesive sequential patterns. We run experiments to compare our method with existing state-of-the-art methods for sequential pattern mining and show that our algorithm is efficient and produces qualitatively interesting patterns on large event sequences.

1 Introduction

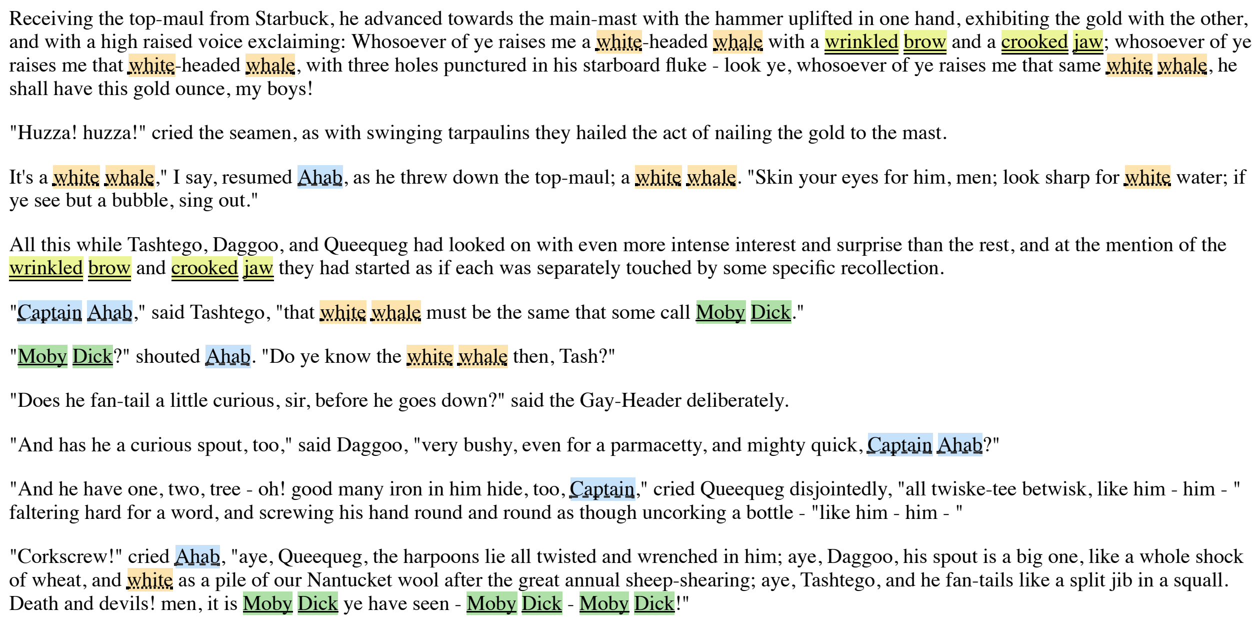
Pattern discovery in sequential data is an established field in data mining. The earliest research focused on the setting where data consisted of *many* sequences, where a pattern was defined as a sequence that re-occurred in a high enough number of such input sequences. Among the algorithms that produce a ranking of the most frequent sequential patterns, given a large database of typically short sequences, are GSP [2] and PrefixSpan [8]. For mining patterns in a *single* long sequence, the first method was proposed by Mannila et al. [11]. Their Winepi method uses a sliding window of a fixed length to traverse the sequence, and a pattern is then considered frequent if it occurs in a high enough number of these sliding windows. Laxman et al. [10] reformulate frequency as the maximal number of non-intersecting *minimal windows* of the pattern in the sequence. In this context, a minimal window of the pattern in the sequence is defined as a subsequence of the input sequence that contains the pattern, such that no smaller subsequence also contains the pattern. All of the above algorithms are able to generate all frequent patterns, by leveraging the so-called Apriori property [1]. This property implies that the frequency of a pattern is never smaller than the frequency of any of its super-patterns. In other words, frequency is an *anti-monotonic* quality measure. While this property is computationally very practical, since large candidate patterns can be generated from smaller patterns, the undesirable side-effect is that single items, and small patterns in general, will always be ranked higher than their super-patterns. Another argument against classical frequency-based techniques is that they report sets or sequences of items where items occur frequently together in a window, but do not account for all individual occurrences of these items. If two items occur frequently, and through pure randomness often occur near each other, they will together form a top-ranked pattern, even though they are not correlated.

Recent research, however, stepped away from mining *all* frequent patterns. Some authors reduce the number of patterns, for example, using information theoretic approaches such as *Minimal Description Length* [7], thereby producing a smaller set of patterns that covers the sequence best [9, 17, 6]. Other authors propose different *measures of interestingness*, that do not benefit from an anti-monotonic quality measure to prune the search space of candidate patterns, but produce a more interesting ranking of patterns [3, 13, 15].

Cule et al. [3, 4] introduced a new interestingness measure called *cohesion*, defined as a measure of how near each other the items making up an interesting itemset occur on average. However, just like frequency-based methods, cohesion has its drawbacks. For example, suppose items a and b occur very frequently next to each other in the input sequence. Now suppose that one occurrence of b is very far from the nearest a. Since cohesion is inversely proportional to the mean of *all* minimal windows, itemset $\{a, b\}$ would score low on cohesion. Furthermore, cohesion is only defined for itemsets and is not a suitable measure for sequential patterns.

In this work, we tackle this problem by measuring the proportion of a pattern's occurrences that are cohesive, where we consider an occurrence to be cohesive if the minimal window length is small relative to the size of the pattern. We call this measure the *quantile-based cohesion* of the pattern. This is a more robust measure that is not susceptible to random outliers. While we concentrate on sequential patterns, the work presented here can directly be applied to other pattern types, such as itemsets, too. In the example above, itemset $\{a, b\}$ would, evaluated by our new measure, score very highly.

*University of Antwerp, Belgium.
†Monash University, Melbourne, Australia



Receiving the top-maul from Starbuck, he advanced towards the main-mast with the hammer uplifted in one hand, exhibiting the gold with the other, and with a high raised voice exclaiming: Whosoever of ye raises me a white-headed whale with a wrinkled brow and a crooked jaw; whosoever of ye raises me that white-headed whale, with three holes punctured in his starboard fluke - look ye, whosoever of ye raises me that same white whale, he shall have this gold ounce, my boys!

"Huzza! huzza!" cried the seamen, as with swinging tarpaulins they hailed the act of nailing the gold to the mast.

It's a white whale," I say, resumed Ahab, as he threw down the top-maul; a white whale. "Skin your eyes for him, men; look sharp for white water; if ye see but a bubble, sing out."

All this while Tashtego, Daggoo, and Queequeg had looked on with even more intense interest and surprise than the rest, and at the mention of the wrinkled brow and crooked jaw they had started as if each was separately touched by some specific recollection.

"Captain Ahab," said Tashtego, "that white whale must be the same that some call Moby Dick."

"Moby Dick?" shouted Ahab. "Do ye know the white whale then, Tash?"

"Does he fan-tail a little curious, sir, before he goes down?" said the Gay-Header deliberately.

"And has he a curious spout, too," said Daggoo, "very bushy, even for a parmacetty, and mighty quick, Captain Ahab?"

"And he have one, two, tree - oh! good many iron in him hide, too, Captain," cried Queequeg disjointedly, "all twiske-tee betwisk, like him - him - " faltering hard for a word, and screwing his hand round and round as though uncorking a bottle - "like him - him - "

"Corkscrew!" cried Ahab, "aye, Queequeg, the harpoons lie all twisted and wrenched in him; aye, Daggoo, his spout is a big one, like a whole shock of wheat, and white as a pile of our Nantucket wool after the great annual sheep-shearing; aye, Tashtego, and he fan-tails like a split jib in a squall. Death and devils! men, it is Moby Dick ye have seen - Moby Dick - Moby Dick!"

Figure 1: Fragment of the novel Moby Dick written by Herman Melville.

We illustrate the various interestingness measures in Figure 1. Here, we show a fragment of the novel Moby Dick written by Herman Melville with four sequential patterns highlighted. These four patterns are all ranked in the top-10 using quantile-based cohesion. Frequency defined as the number of minimal non-overlapping windows, as proposed by Laxman et. al, would report $(white, whale)$, $(Captain, Ahab)$ and $(Moby, Dick)$ in the top-10, but not $(wrinkled, brow, crooked, jaw)$ because this pattern does not occur frequently enough. We also remark that other patterns, such as $(Ahab, Dick)$, are ranked highly using frequency alone, despite not being correlated. The cohesion-based FCI algorithm [3] does not rank $(Captain, Ahab)$ high, due to fact that the two items, though correlated, also occasionally appear far from each other, resulting in a relatively large mean of minimal window lengths.

Since quantile-based cohesion is not anti-monotonic, designing an efficient algorithm to exactly find all sequential patterns is not trivial. To facilitate our search, we define an *upper bound* that computes the maximal quantile-based cohesion for any super-pattern of the current candidate sequential pattern that could still be generated to *prune* candidate patterns. Our algorithm uses *constrained prefix-projected pattern growth* to generate all candidates, and uses this upper bound for additional pruning. Computation of all minimal windows and generation of candidates becomes more efficient as the projected input sequence becomes smaller ensuring our algorithm is also efficient on larger event sequences with many items. We perform several experiments to validate that these algorithms perform well on artificial and text datasets from a qualitative and performance perspective.

The remainder of this paper is organized as follows. In Section 2 we formally describe the problem setting and define the patterns we aim to discover. Section 3 provides a detailed description of our algorithm and upper bound. In Section 4 we present an experimental evaluation of our method, and compare with a number of related state-of-the-art methods. We present an overview of the most relevant related work in Section 5 and conclude our work in Section 6.

2 Problem Setting

The input dataset consists of a *single* sequence of *items* (or *events*), that is $\mathcal{S} = (\langle i_1, t_1\rangle, \ldots, \langle i_n, t_n\rangle)$, where $i_k \in \Omega$ is an item coming from a finite domain Ω of all possible items, and t_k is a timestamp. The sequence is ordered chronologically so for any $1 < k \leqslant n$, it holds that $t_{k-1} \leqslant t_k$. A *window* $\mathcal{S}[t_a, t_b]$ is a subsequence of $\mathcal{S}$ between timestamps t_a and t_b, that is $\mathcal{S}[t_a, t_b]$ contains all $\langle i_k, t_k\rangle \in \mathcal{S}$ for which $t_a \leqslant t_k \leqslant t_b$. We define the window *length* as $|\mathcal{S}[t_a, t_b]| = t_b - t_a$. For simplicity we omit the timestamps from our *examples* and write a sequence as $(i_1, \ldots, i_n)$ thereby assuming the timestamps are consecutive integers.

A sequential pattern is denoted as $X_s =$

$(s_1, \ldots, s_m)$, representing a pattern that consists of items s_1 until s_m in that order, where $s_k \in \Omega$. We do not require each item s_k to be unique, that is sequential patterns can contain repeating items. Unlike windows, a sequential pattern occurrence allows gaps between items. We say that a sequential pattern X_s occurs in a window $\mathcal{S}[t_a, t_b]$ if all items in X_s occur in the specified order in the window, that is,

$$\begin{aligned} X_s = (s_1, \ldots, s_m) \prec \mathcal{S}[t_a, t_b] \Leftrightarrow \\ \exists \langle s_1, t_1 \rangle, \ldots, \langle s_m, t_m \rangle \in \mathcal{S}[t_a, t_b] : \\ \forall i, j \in \{1, \ldots, m\} : i < j \Rightarrow t_i < t_j. \end{aligned}$$

To evaluate a pattern, we make use of *minimal windows.* For every *distinct* item $i \in X_s$ and every timestamp t where $\langle i, t \rangle \in \mathcal{S}$, we need to find the shortest window around t that contains an occurrence of X_s. Formally, we define the minimal window at timestamp t as

$$W_t(X_s, \mathcal{S}) = \begin{cases} \infty & \text{if } \nexists \mathcal{S}[t_a, t_b] : t_a \leqslant t \leqslant t_b \wedge X_s \prec \mathcal{S}[t_a, t_b] \\ \min\limits_{\mathcal{S}[t_a, t_b]} \{ |\mathcal{S}[t_a, t_b]| \mid t_a \leqslant t \leqslant t_b \ \wedge X_s \prec \mathcal{S}[t_a, t_b] \} & \text{otherwise.} \end{cases}$$

Note that a sequential pattern is sometimes not covered by any window at timestamp t. For example, given the sequential pattern $X_s = (a, b)$ and the sequence $\mathcal{S} = (\ldots, b, a)$, for the last a there is no window that would cover X_s. In this case we say the minimal window has a length of ∞. Since we measure the proportion of occurrences that are cohesive, this is not a problem, as large minimal windows are discarded anyway. Finally, we denote the set of occurrences of the pattern as $cover(X_s, \mathcal{S}) = \{t \mid \langle i, t \rangle \in \mathcal{S} \wedge i \in X_s\}$, and define its support as $support(X_s, \mathcal{S}) = |cover(X_s, \mathcal{S})|$.

We are now ready to define *quantile-based cohesion.* This measure tackles the problems of both frequency-based and cohesion-based methods, as illustrated in Section 1. Given a cohesion threshold α, that determines, relative to the pattern size, if a pattern occurrence is cohesive enough, we compute the proportion of the occurrences that are cohesive. Formally, given a set of minimal windows for each occurrence of a pattern, we define the *quantile-based cohesion* for a sequential pattern X_s in sequence $\mathcal{S}$ as

$$C_{quan}(X_s) = \frac{|\{t | \ t \in cover(X_s) \wedge W_t(X_s) < \alpha \cdot |X_s|\}|}{support(X_s)},$$

where we omit the $\mathcal{S}$ argument if it is clear from the context. Finally, we remark that, given a frequency threshold θ, we ignore all *infrequent* items, that is if $support(i) < \theta$, we do not use item $i \in \Omega$ in the generation of candidate patterns.

Our goal is to solve the following problem: *Given a single sequence of items $\mathcal{S}$, a cohesion threshold α, a frequency threshold θ, a size limit maxsize, and the number of desired patterns k, find each sequential pattern X_s where $|X_s| \leqslant maxsize$, for each $i \in X_s, support(i) \geqslant \theta$, and X_s is ranked in the top-k set of patterns according to $C_{quan}(X_s)$ w.r.t α.*

3 Algorithm

In this section, we present a detailed description of our algorithm for mining sequential patterns. We first show how we generate candidates using prefix-projected pattern growth. We then discuss how we can prune large numbers of potential candidates by computing an upper bound on quantile-based cohesion. Parameters for controlling our algorithm include k, α and *maxsize*.

3.1 Prefixed-projected pattern growth Our algorithm combines ideas from two different methods. At its core, our algorithm is similar to the depth-first search by Cule et al. for mining cohesive itemsets [3]. We first generate candidates in a depth-first way. For each candidate, we compute the set of minimal windows, and prune a candidate and associated super-patterns based on an upper bound of quantile-based cohesion. There are two *bottlenecks* in this baseline algorithm. First, we would have to compute the set of minimal windows for each candidate, which requires visiting all occurrences of items in the current candidate X_s. Second a naive approach would generate new candidate patterns by combining the current candidate with all items in Ω. In order to address both bottlenecks, we integrate this approach with the strategy of *recursively projecting the input sequence*, similar to prefix-projected pattern growth first used in PrefixSpan [8].

3.1.1 Definitions During the depth-first search we generate candidate *super-sequences* Z_s by adding items from the set Y at the end of the current candidate sequence X_s, that is, given $X_s = (s_1, \ldots, s_n)$ and $Y = \{y_1, \ldots, y_l\}$,

$$\begin{aligned} \mathcal{Z}(X_s, Y) = \{Z_s | Z_s = (s_1, \ldots, s_n, z_{n+1}, \ldots, z_m) \\ \wedge |Z_s| \leqslant maxsize \wedge \forall i \in [n+1, m] : z_i \in Y\}. \end{aligned}$$

We initialize Y as Ω at depth 0. Y can also be computed based on the *projection* of $\mathcal{S}$ on X_s defined as

$$\mathcal{P}_{X_s}(\mathcal{S}) = \{\mathcal{S}[t_a, t_b] | \langle s_1, t_a \rangle \in \mathcal{S} \wedge X_s \prec \mathcal{S}[t_a, t_b]\},$$

with $t_b = t_a + \alpha \cdot maxsize$. Note that we restrict the length of the projected windows to $\alpha \cdot maxsize$. We define the *suffix* of a window given a sequential pattern X_s as the subsequence after the first occurrence of X_s,

$$\begin{aligned} \mathit{suffix}(\mathcal{S}[t_a,t_b],X_s) =& \mathcal{S}[t+\epsilon,t_b] \mid t_a \leqslant t \leqslant t_b \wedge \\ & X_s \prec \mathcal{S}[t_a,t] \wedge \nexists t' : t' < t \wedge X_s \prec \mathcal{S}[t_a,t'], \end{aligned}$$

where ϵ denotes the smallest possible period of time between two non-simultaneous events. Finally, given the projection on X_s of $\mathcal{S}$ we can define the *multiset* of all possible candidate items Y^+ as

$$Y^+ = \bigcup_{\mathcal{S}[t_a,t_b]\in\mathcal{P}_{X_s}} \left(\biguplus_{s_k \in \mathit{suffix}(\mathcal{S}[t_a,t_b],X_s)} \{s_k\} \right),$$

where we use a multiset-union $\biguplus$, which allows us to bound the number of possible repetitions allowed for each item z_i in candidate super-sequences Z_s. A similar definition based on the set-union is used to compute Y.

We now discuss the intuition behind these definitions. We first observe that we are only interested in computing the number of minimal windows that are smaller than $\alpha \cdot |X_s|$. Therefore, it is not necessary to compute all minimal windows. A second observation is that given pattern $X_s = (s_1)$ we can guarantee that for any super-sequence, each interesting minimal window will start with an occurrence of s_1 and must be smaller than $\alpha \cdot \mathit{maxsize}$. Therefore, we can compute the set of minimal windows based on the projection of (s_1) induced on $\mathcal{S}$ for every super-sequence that starts with (s_1). On the first level, the set of projected windows might not be much smaller than $|\mathcal{S}|$, but as the pattern becomes longer, the projection will become much smaller. For instance, given $Z_s = (s_1, s_2)$ each window $\mathcal{S}[t_a,t_b] \in \mathcal{P}_{(s_1)}$ can be removed from $\mathcal{P}_{(s_1,s_2)}$ if $(s_1,s_2) \nprec \mathcal{S}[t_a,t_b]$. Furthermore, each individual window in the projection will shrink based on $\mathit{suffix}(\mathcal{S}[t_a,t_b],X_s)$. Thus by computing the projected windows, two bottlenecks of the depth-first search in this settings are resolved: first, minimal windows for a candidate $Z_s = (s_1,\ldots,s_{k+1})$ can be computed *incrementally* on the *monotonically decreasing* projection induced by $X_s = (s_1,\ldots,s_k)$, and, second, candidate items s_{k+1} for generating candidates at each next level, are not selected from the full set Ω but must occur in Y^+. Finally, an additional benefit of applying prefix-projected pattern growth is that our upper bound becomes tighter, as discussed in Section 3.2.

3.1.2 Implementation The main algorithm for mining quantile-based cohesive sequential patterns (QCSP) is shown in Algorithm 1. We maintain a stack for performing the recursive prefix-projected search. During the recursion we maintain three variables: the current candidate sequential pattern X_s, the projection on X_s of the sequence and a set of candidate items for generating super-sequences Z_s where X_s is a prefix of Z_s. We initialize this stack by setting X_s to the empty sequence, the projection is $\mathcal{S}$ itself, and the set of all candidate items is Ω (line 1). Next, we initialize an empty heap that contains at most k patterns sorted on quantile-based cohesion (line 2). This top-k of most quantile-based cohesive patterns is also returned after the main prefix-projected search loop has finished (lines 4-26). In the main loop we first pop the current node from the stack (line 5). We investigate if this is a leaf, that is, an unpruned sequential pattern, with no more super-sequences to enumerate. We add the candidate to the heap of top-k most cohesive patterns if its quantile-based cohesion is higher than the current worst candidate pattern in the heap (line 7). Note that the first k candidates will be added without any condition, but the minimal quantile-based cohesion

Algorithm 1: QCSP($\mathcal{S}, k, \alpha, \mathit{maxsize}$) finds top-$k$ quantile-cohesive sequential patterns in $\mathcal{S}$

```
1  $stack \leftarrow [\langle \varnothing, \mathcal{S}, \Omega \rangle]$;
2  $heap \leftarrow$ make_heap($k$);
3  $min\_coh \leftarrow 0.0$;
4  while $stack \neq \varnothing$ do
5      $\langle X_s, \mathcal{P}_{X_s}, Y \rangle \leftarrow stack$.pop();
6      if $Y = \varnothing$ then
7          if $|X_s| > 1 \wedge C_{quan}(X_s) > min\_coh$ then
8              $heap$.push($\langle X_s, C_{quan}(X_s) \rangle$);
9              if $heap.size() \geqslant k$ then
10                 $heap$.pop();
11                 $min\_coh \leftarrow heap$.min()
12             end
13         end
14     end
15     else
16         if $X_s \cap Y = \varnothing \wedge mingap(X_s) + |Z_{max}| > \alpha \cdot (|X_s| + |Z_{max}|)$ then continue ;
17         if $C_{maxquan}(X_s, Y) \leqslant min\_coh$ then continue ;
18         $s_{k+1} \leftarrow$ first($Y$);
19         $stack$.push($\langle X_s, \mathcal{P}_{X_s}, Y \backslash \{s_{k+1}\} \rangle$);
20         if $|X_s| = maxsize$ then continue ;
21         $Z_s \leftarrow (X_s, s_{k+1})$;
22         $\mathcal{P}_{Z_s} \leftarrow$ PROJECT($\mathcal{S}, Z_s, \mathcal{P}_{X_s}, \alpha, maxsize$);
23         $Y_{Z_s} \leftarrow$ PROJ_CANDIDATES($\mathcal{S}, Z_s, \mathcal{P}_{Z_s}$);
24         $stack$.push($\langle Z_s, \mathcal{P}_{Z_s}, Y_{Z_s} \rangle$);
25     end
26 end
27 return heap;
```

will increase as more and more candidates are uncovered, which in turn affects the pruning. If the candidate is not a leaf we evaluate the *mingap* and *upper bound* function $C_{maxquan}(X_s, Y)$ which are explained in the next subsection (line 16-17). If the current candidate, and all its super-sequences cannot be pruned, we generate a super-sequence $Z_s = (X_s, s_{k+1})$ using the first item in Y (line 21), the set of possible items to generate candidates of length $|X_s| + 1$. We compute the projection of Z_s induced on $\mathcal{S}$ by calling the function PROJECT (line 22). The details of the auxiliary functions PROJECT, which computes the projection $\mathcal{P}_{Z_s}(\mathcal{S})$ incrementally based on the projection of $\mathcal{P}_{X_s}(\mathcal{S})$ and PROJ_CANDIDATES, which computes the set of possible candidate items Y, are provided in the Appendix[1].

We use an example, shown in Figure 2, to illustrate the runtime behavior of our algorithm. We show the trace on the example sequence $(a, b, c, \_, \_, \_, \_, b, a, c)$ with parameters $\alpha = 2$, $maxsize = 3$ and $k = 2$. In theory, there are $|\Omega|^2 + |\Omega|^3$ candidates possible. The first candidate generated is $X_s = (a)$. The projected window size is at most $\alpha \cdot maxsize = 6$, and there are two windows in $\mathcal{P}_{(a)}$. After the projection, we find that only b and c are left in Y, the set of remaining items to form candidates that start with a. As c occurs twice and b only once, c is visited first, and our next candidate is $X_s = (a, c)$. Since the suffix of $\mathcal{P}_{(a,c)}$ does not contain any more items, it becomes a leaf, and we add sequential pattern (a, c) to the heap with a quantile-based cohesion of 1. Next we project on $X_s = (a, b)$ and remove one window in the projection. The only possible suffix left is c. We then generate $X_s = (a, b, c)$. This candidate pattern is a leaf node, and is added to the heap with a quantile-based cohesion of 0.5. Next, $X_s = (a, b)$ is visited again as a leaf, but not added to the heap, because its quantile-based cohesion of 0.5 is not strictly higher than the minimal score of $C_{quan}((a, b, c)) = 0.5$ of the top-2 patterns currently in the heap. Next, $X_s = (b)$ is generated, and then $X_s = (b, c)$. The quantile-based cohesion of $X_s = (b, c)$ is 1 and it is added to the heap, replacing the previously lowest-scoring element (a, b, c). Then, $X_s = (b, a)$ is generated. The maximal cohesion of (b, a), with $Y = \{c\}$ given by $C_{maxquan}((b, a), \{c\}) = \frac{4}{6}$ (see Section 3.2) is less than the the current minimal score of 1, so we can prune this branch. Finally, candidates $X_s = (c)$ and $X_s = (c, b)$ are generated, with cohesion 0.0 since no window is smaller than $\alpha \cdot 2$. The final top-k heap contains (b, c) and (a, c) both with a cohesion of 1.

[1] http://win.ua.ac.be/~adrem/bibrem/pubs/feremans18topk.pdf

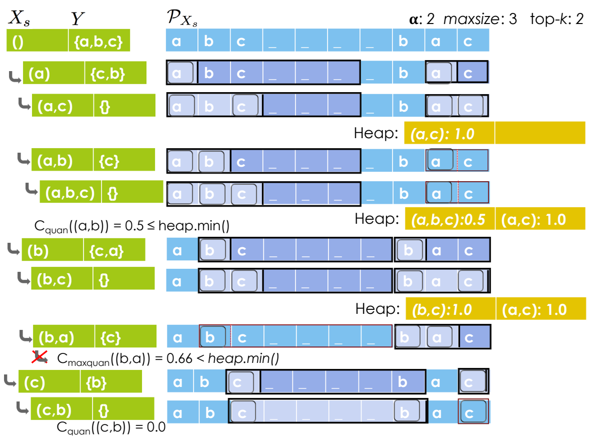


Figure 2: Example to illustrate prefix-projected pattern growth algorithm QCSP. The final top-2 quantile-based cohesive sequential patterns are (b, c) and (a, c).

3.2 Pruning

At any node in the search tree, let X_s denote the current candidate sequential pattern, while Y denotes all items that can still be added to X_s to form super-sequences. If we can compute an upper bound on the quantile-based cohesion for all candidates $Z_s \in \mathcal{Z}(X_s, Y)$, and this maximal score is lower than *min_coh* we can prune the branch, where *min_coh* corresponds to the current minimum score of any pattern in the heap, or, if the heap is not full, it is set to 0. In this section, we derive this upper bound.

3.2.1 Upper bound based on mingap

Our first *upper bound* comes from the observation that in some cases not a single occurrence of two items is cohesive. Intuitively, if the *minimal gap*, that is the minimal window length of any occurrence of (a, b) is already too high, the likelihood that (a, b) or any superpattern is cohesive is small. We define the minimal gap as

$$mingap(X_s) = \min_{t \,\in\, cover(X_s)} W_t(X_s).$$

THEOREM 3.1. *Let X_s be a candidate pattern and Y the set of items that can still be added to X_s. Then, for each $Z_s \in \mathcal{Z}(X_s, Y)$ generated as candidate by Algorithm 1,*

$$\begin{gathered} C_{quan}(Z_s) = C_{quan}(X_s) = 0 \\ \textit{if } mingap(X_s) + |Z_{max}| > \alpha \cdot (|X_s| + |Z_{max}|) \\ \textit{and } X_s \cap Y = \varnothing, \\ \textit{where } |Z_{max}| = \max_{\mathcal{S}[t_a,t_b] \in \mathcal{P}_{X_s}} |\textit{suffix}(\mathcal{S}[t_a, t_b], X_s)|. \end{gathered}$$

Proof. The proof is given in the Appendix.

Consider the following set of projected windows on $X_s = (a, b)$: $\mathcal{P}_{X_s} = \{(\mathbf{a}, \_, \_, \_, \mathbf{b}, d, f, \_), (\mathbf{a}, \_, \_, \_, \mathbf{b}, c, d, \_), (\mathbf{a}, \_, \_, \_, \mathbf{b}, \_, c, e)\}$. The set of remaining items is $Y = \{c, d, e, f\}$. Note that $X_s \cap Y = \varnothing$. $\mathcal{Z}(X_s, Y)$ can contain super-patterns such as (a, b, f), (a, b, d, f) or (a, b, c, c). Here, $mingap(X_s)$ is 5 and $|Z_{max}|$ is 2 (ignoring gaps). The longest possible pattern has length $|X_s| + |Z_{max}| = 2 + 2$. If we assume $\alpha = 1$ than we can prune since $5 + 2 > 1 \cdot (2 + 2)$.

3.2.2 Upper bound on quantile-based cohesion

We now present a bound on the number of remaining cohesive minimal windows of any candidate super-sequence $Z_s \in \mathcal{Z}(X_s, Y)$, even if repeating items are possible (that is $X_s \cap Y \neq \varnothing$). We can prune X_s, and all super-sequences, if the maximal value for quantile-based cohesion is lower than the current value of *min_coh*.

Theorem 3.2. *Let X_s be a candidate pattern and Y the set of items that can still be added to X_s. Then for each $Z_s \in \mathcal{Z}(X_s, Y)$ generated as candidate by Algorithm 1,*

$$C_{quan}(Z_s) \leqslant C_{maxquan}(X_s, Y), \text{ where}$$
$$C_{maxquan}(X_s, Y) = 1.0 - \frac{|\{t| \ t \in \beta \wedge W_t(X_s) \geqslant \alpha \cdot |Z'_{max}|\}|}{support(Z'_{max})},$$
$$|Z'_{max}| = min(maxsize, |X_s| + |Y^+|),$$
$$support(Z'_{max}) = \sum_{i \, \in \, X_s \cup Y} support(i),$$
$$\beta = \{t \mid \langle i, t\rangle \in \mathcal{S} \wedge \ i \in X_s \wedge \nexists \langle j, t\rangle \in \mathcal{S} : j \in Y\}.$$

Proof. The proof is given in the Appendix.

We illustrate this bound using an example. Suppose $X_s = (a, b)$ occurs 10 times and $Y = (c)$ occurs 2 times. The minimal windows of X_s are $\{2, 2, 2, 30, 30, 30, 30, 30, \infty, \infty\}$. Let us further assume that $min\_coh = 0.5$ and $alpha = 2$. There is only one non-repeating candidate item in $Y^+ = \{c\}$, thus only $Z_s = Z'_{max}(X) = (a, b, c)$ is possible. The maximal window length possible is $\alpha \cdot 3 = 6$. There are seven windows of X_s that are larger than 6. $support(Z'_{max}) = 12$, thus $C_{maxquan}(X_s, Y) = 1 - \frac{7}{12} = \frac{5}{12}$ which is lower than *min_coh* so we can prune this branch. We remark that, unlike pruning based on mingap, when X_s and Y are not disjoint, we can still count windows for non-overlapping items in X_s, for example given $X_s = (a, b, c)$ and $Y = \{c, d\}$, we can compute all windows of (a, b, c) for all occurrences of items in $X_s \backslash Y = \{a, b\}$.

4 Experiments

In our experiments, we use one synthetic dataset and three text datasets for easy interpretation of patterns. We compare QCSP with three state-of-the-art methods both in terms of performance and the quality of output. FCI [3] finds all cohesive *itemsets*, SkOPUS [13] finds the top-k sequential patterns with the highest leverage, and GoKrimp [9] finds a set of patterns that best compresses the input. For all three methods, we used publicly available implementations developed in Java[2,3,4]. Our QCSP implementation and the used datasets are publicly available, too[5]. Since we compare our method with state-of-the-art algorithms for both the single sequence and the multiple sequences setting, we use two versions of each dataset, one for each setting.

4.1 Datasets The *Synthetic* dataset is created by generating a single sequence of 2000 items randomly selected between 6 and 50. Next, we insert the sequential pattern $(1, 2, 3, 4, 5)$ with at most 5 gaps at 40 random non-overlapping locations. We transform this sequence $\mathcal{S}$ into a set of sequences $\mathcal{S}'$ by using a sliding window of size 20. The *Moby* dataset consists of all words in the novel Moby Dick written by Herman Melville[6]. We preprocessed the text using the Porter Stemmer, and removed the stopwords. We transformed the single sequence $\mathcal{S}$ into a set of sequences $\mathcal{S}'$ by creating a separate sequence for each sentence. *JMLR* consists of abstracts of papers from the Journal of Machine Learning Research[7], where each abstract is preprocessed as in *Moby*. Each abstract is considered a separate sequence. Since our method requires a single sequence, we transform this dataset by concatenating the abstracts, adding $\alpha\cdot$ *maxsize* time stamps between any two abstracts, thus avoiding generating patterns that span over two different abstracts. Finally, *Trump* consists of tweets of president Trump from 1 January 2016 until 2 October 2017[8]. We removed all re-tweets and preprocessed the tweet texts as in *Moby*, and converted into a single sequence as in *JMLR*. Table 1 shows the basic characteristics of each dataset, where $|\Omega|$ denotes the number of distinct items in the dataset and $\mu(\mathcal{S}')$ the average length of a sequence in the multiple sequences setting.

4.2 Performance comparison between methods We begin by remarking that it is not possible to compare the runtime of all algorithms directly for several reasons. First of all, the runtime depends on the chosen parameters and input representation (single

[2]https://bitbucket.org/len_feremans/sequencepatternmining_public/
[3]https://github.com/fpetitjean/Skopus
[4]http://www.philippe-fournier-viger.com/spmf/
[5]https://bitbucket.org/len_feremans/qcsp_public
[6]http://www.gutenberg.org/ebooks/2701
[7]http://www.jmlr.org/papers
[8]http://www.trumptwitterarchive.com/

Table 1: Characteristics of the used datasets.

Dataset	$\|\mathcal{S}\|$	$\|\Omega\|$	$\|\mathcal{S}'\|$	$\mu(\mathcal{S}')$
Synthetic	2 000	50	2 000	20.0
Moby	113 264	2 059	10 066	11.2
JMLR	75 515	3 846	787	96.0
Trump	57 518	1 069	5 670	17.9

Table 2: Runtimes on the different datasets.

Dataset	FCI	SkOPUS	GoKrimp	QCSP
Synthetic	44.2s	19.8s	1.0s	1.7s
Moby	126.0s	47.8s	2.0s	18.4s
JMLR	255.4s	40.6s	2.0s	25.9s
Trump	196.9s	2.5s	5.0s	8.1s

sequence or many sequences), which are different for each method. Second, FCI solves a different problem since it mines itemsets, and QCSP allows sequential patterns to contain repeating items, which is not the case for SkOPUS or GoKrimp. However, despite these restrictions, we include this experiment to get an idea of the overall runtime required for each method. For QCSP and SkOPUS we set *maxsize* $= 5$ and $k = 50$. For FCI we set *maxsize* $= 5$, and for QCSP we set α to 2. For all experiments, we used $\theta = 10$, and we filtered out the infrequent items in a pre-processing step. Since we cannot directly control the number of patterns in FCI, we report the runtime for the run with the highest cohesion threshold that leads to discovering at least k patterns. Finally, GoKrimp has no parameters to tune. Table 2 shows the runtimes on all datasets. We note that FCI is notably slower than other methods, while GoKrimp is notably faster. However, GoKrimp is different in nature, since it does not attempt to generate all candidates and employs *heuristics* to perform greedy search. Furthermore, GoKrimp always produced fewer than 50 patterns. Overall, we can conclude that the runtime of QCSP is both acceptable and competitive.

We also analyzed the impact on performance of pruning based on mingap and based on an upper bound on quantile-based cohesion. We provide a detailed report of these experiments in the Appendix. In short, we can conclude that pruning reduces the number of candidates by an *order of magnitude*, and also improves runtime performance for higher values of maxsize. We also conclude that the effect of a higher k on runtime behavior is not very large. The effect of varying *maxsize* is clear — if *maxsize* increases, the total runtime grows rapidly, since the number of possible candidate patterns increases *exponentially*. Even so, on *Moby*, our largest dataset, with a length of over 100 000 items (see Table 1), finding the top-20 quantile-based cohesive sequential patterns up to a maximal size of 10 took only 39 minutes.

4.3 Quality comparison between methods In our final set of experiments we compare the quality of the patterns found by the different methods. We use the same parameters as in the previous section and increase k to 250. Table 3 shows the top-5 sequential patterns discovered by the various methods on all four datasets, with patterns found only by a single method shown in bold (we provide the top-20 for all methods in the Appendix). On the *Synthetic* dataset, we see that both GoKrimp and QCSP rank the desired pattern first. FCI ranks the pattern in the $9th$ position because, due to the randomness of gaps in our generator, for some subsequences (but not all) the ratio between pattern length and average minimal window length is larger. SkOPUS surprisingly does not report the sequence in the top-500. It seems that the definition of expected support seems to be biased towards shorter sequential patterns. For the text datasets, we have omitted the results for FCI for the sake of brevity. SkOPUS, GoKrimp and QCSP all seem to report interesting patterns. The main difference between the patterns produced by QCSP and GoKrimp is that the former ranks on C_{quan} and only considers the *proportion* of cohesive occurrences. As such, a sequential pattern that occurs cohesively in 2 out of a total of 2 occurrences, ranks as highly as a sequence that occurs cohesively in 100 out of a total of 100 instances. By sorting the top-250 quantile-based cohesive patterns on support rather than C_{quan}, we get a ranking very close to GoKrimp. In other words, most of the patterns found by GoKrimp are ranked relatively highly by QCSP, but, crucially, not vice versa — many interesting patterns discovered by QCSP are not found at all by GoKrimp.

SkOPUS mostly reports short patterns, and its top-250 for *Trump* consists only of patterns of length 2. There is little overlap between the output of SkOPUS and QCSP. For instance, in *JMLR*, 59 patterns found in the top-250 by SkOPUS start with *paper*, and 44 end with *result*, while the top-250 produced by QCSP contains no patterns starting with *paper*, and only one pattern ends with *result*, namely (*experi*,*result*).

Unlike other methods, QCSP ranks many *long* patterns in the top-250. Patterns such as (*crooked*, *hillary*, *clinton*) and (*repeal*, *replace*, *obamacare*) in the *Trump* dataset, or (*reproduce*, *kernel*, *hilbert*, *space*) and (*markov*, *chain*, *monte*, *carlo*) in the *JMLR* dataset are not found in the top-250 of the other methods at all. Less frequent patterns with high quantile-based cohesion, such as (*las*,*vegas*), (*mrs*,*hussey*), (*nearest*,*neighbor*), (*cross*,*validation*) or (*bayesian*, *network*)

Table 3: Top-5 patterns for each method.

FCI	SkOPUS	GoKrimp	QCSP
$\{1,2\}$	$3,4,5$	$1,2,3,4,5$	$1,2,3,4,5$
$\{3,4\}$	$2,3,4$	$3,4,5$	$2,3,4,5$
$\{4,5\}$	$1,2,3$	$1,2,3$	$3,4,5$
$\{3,4,5\}$	$2,3,5$	$4,5$	$4,5$
$\{1,2,3\}$	$2,4,5$	$1,2$	$3,4$

(a) *Synthetic* dataset

SkOPUS	GoKrimp	QCSP
sperm, whale	sperm, whale	moby, dick
white, whale	moby, dick	**mrs, hussey**
though, yet	white, whale	**ii, octavo**
old, man	mast, head	**crow, nest**
moby, dick	old, man	**iii, duodecimo**

(b) *Moby* dataset

SkOPUS	GoKrimp	QCSP
paper, show	support, vector, machin	**mont, carlo**
paper, result	real, world	**nearest, neighbor**
paper, experi	machin, learn	support, vector
paper, algorithm	state, art	**http, www**
support, vector	reproduc, hilbert, space	**cross, valid**

(c) *JMLR* dataset

SkOPUS	GoKrimp	QCSP
make, america	make, america, great, again	puerto, rico
make, great	U,S	witch, hunt
crooked, hillary	crooked, hillary	elizabeth, warren
hillary, clinton	fake, news	**las, vega**
america, great	ted, cruz	goofy, elizabeth

(d) *Trump* dataset

are not reported at all in the top-250 of SkOPUS or the limited pattern set of GoKrimp, despite the fact that, for example, 46 out of 47 occurrences of (*puerto*,*rico*) are cohesive in *Trump* tweets, which clearly makes it an important pattern. We conclude that QCSP is capable of finding interesting patterns that other methods fail to discover, while not missing out on interesting patterns that other methods do find.

5 Related Work

We have examined the most important related work in Section 1, and experimentally compared our work with the existing state-of-the-art methods in Section 4. Here, we place our work into the wider context of sequential pattern mining. In a single long input sequence, Winepi [11], Laxman [10] and Marbles [5] mine all frequent episodes, where frequency is defined using sliding windows, minimal windows and weighted minimal windows, respectively. It has been experimentally validated in recent research that the top-k most frequent patterns are often not very interesting [3, 13, 6].

SkOPUS [13] and EpiRank [15] are two approaches that rank sequential patterns and general episodes, respectively, based on an elaborate definition of *leverage*. We have compared our method with SkOPUS and found that we typically rank longer patterns higher. EpiRank ranks episodes based on the output of an existing frequent sequential pattern miner [16]. As such, it is unlikely that less frequent and/or longer, but strongly quantile-based cohesive, sequential patterns reported by QCSP would be uncovered by this method. Theoretically, we could mine a very large set of candidate sequential patterns by setting the support threshold very low, but this would result in pattern explosion, especially for enumerating longer patterns.

Related to EpiRank, Tatti also proposed a way to measure the mean and variance of minimal windows of episode occurrences, and rank episodes by comparing these values with the expected length according to the independence model [14]. Like EpiRank, this method also requires as input the output of an existing frequency based episode miner, making it, too, either unlikely or inefficient to produce less frequent or longer cohesive candidates than our direct approach.

Pattern reduction based on *Minimal Description Length* produces a smaller set of patterns that covers the sequence best. We have compared with GoKrimp [9], which is related to other MDL-based algorithms such as SQS[17] and ISM[6]. All these methods take multiple sequences as input, which is different from our approach. Another difference is that rather than enumerating as few candidates as possible and then selecting the best candidates according to an interestingness measure, they employ *heuristic search* to incrementally build a set of non-redundant patterns. Applying heuristic search to finding top quantile-based cohesive patterns is also feasible, but we would lose the guarantee that our output contains the *exact* set of top-k most quantile-based cohesive patterns.

QCSP is also related to research in *constraint pattern mining*. PG by Pei et al. [12] defines a generic sequential pattern algorithm, based on pattern-growth, that can handle a variety of constraints. We use a *length constraint* induced by *maxsize* and a *gap constraint* induced by $\alpha \cdot maxsize$.

As mentioned in Section 1, our definition of quantile-based cohesion for sequential patterns is based on the existing definition of cohesion for itemsets [3]. We note that our work (both the definition and the algorithm) could easily be adapted for mining quantile-based cohesive itemsets or episodes efficiently.

6 Conclusions and future work

In this paper, we present a novel method for finding interesting sequential patterns in event sequences. Compared to other interestingness measures, our quantile-based cohesion is not biased towards shorter patterns or patterns consisting of very frequent items. Furthermore, our measure is robust in the presence of outliers, and flexible, since we do not use a sliding window of fixed length, as is common in existing methods. We define quantile-based cohesion as the proportion of occurrences of the pattern that are cohesive, i.e., where the minimal window is small. This measure is easy to interpret, and reports both frequent and less frequent, but always strongly correlated, sequential patterns, that other methods often fail to find. Since quantile-based cohesion is not an anti-monotonic measure, we rely on an upper bound to prune candidate patterns and their supersequences. We include this pruning function in a variant of constraint-based sequential pattern mining based on pattern growth, and show both theoretically and empirically that our algorithm works efficiently.

In future work, we intend to investigate adapting our algorithm for the *anytime* setting, by prioritizing more likely candidates in order to find the most interesting patterns quickly, rather than waiting for the entire output. Additionally, we are interested in applications where pattern mining is not the end goal in itself. For example, quantile-based cohesive sequential patterns could be used in tasks such as prediction, classification or anomaly detection within temporal data.

7 Acknowledgements

The authors would like to thank the VLAIO SBO HYMOP project for funding this research.

References

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *International Conference on Very Large Data Bases*, pages 487–499, 1994.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. *International Conference on Data Engineering*, 0:3–14, 1995.

[3] Boris Cule, Len Feremans, and Bart Goethals. Efficient discovery of sets of co-occurring items in event sequences. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 361–377. Springer, 2016.

[4] Boris Cule, Bart Goethals, and Céline Robardet. A new constraint for mining sets in sequences. In *SIAM International Conference on Data Mining*, pages 317–328, 2009.

[5] Boris Cule, Nikolaj Tatti, and Bart Goethals. Marbles: Mining association rules buried in long event sequences. *Statistical Analysis and Data Mining*, 7(2):93–110, 2014.

[6] Jaroslav Fowkes and Charles Sutton. A subsequence interleaving model for sequential pattern mining. *arXiv preprint arXiv:1602.05012*, 2016.

[7] Peter D Grünwald. *The minimum description length principle*. MIT press, 2007.

[8] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and MC Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *proceedings of the 17th international conference on data engineering*, pages 215–224, 2001.

[9] Hoang Thanh Lam, Fabian Mörchen, Dmitriy Fradkin, and Toon Calders. Mining compressing sequential patterns. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 7(1):34–52, 2014.

[10] Srivatsan Laxman, P. S. Sastry, and K.P. Unnikrishnan. A fast algorithm for finding frequent episodes in event streams. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 410–419, 2007.

[11] Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.

[12] Jian Pei, Jiawei Han, and Wei Wang. Constraint-based sequential pattern mining: the pattern-growth methods. *Journal of Intelligent Information Systems*, 28(2):133–160, 2007.

[13] François Petitjean, Tao Li, Nikolaj Tatti, and Geoffrey I Webb. Skopus: Mining top-k sequential patterns under leverage. *Data Mining and Knowledge Discovery*, 30(5):1086–1111, 2016.

[14] Nikolaj Tatti. Discovering episodes with compact minimal windows. *Data Mining and Knowledge Discovery*, 28(4):1046–1077, 2014.

[15] Nikolaj Tatti. Ranking episodes using a partition model. *Data Mining and Knowledge Discovery*, 29(5):1312–1342, 2015.

[16] Nikolaj Tatti and Boris Cule. Mining closed strict episodes. *Data Mining and Knowledge Discovery*, 25(1):34–66, 2012.

[17] Nikolaj Tatti and Jilles Vreeken. The long and the short of it: summarising event sequences with serial episodes. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 462–470. ACM, 2012.

Appendix

A Auxiliary Functions used by QSCP

In this section we discuss in detail the implementation of two auxiliary functions used by QCSP.

Algorithm 2 shows the pseudo code for computing the projection of $Z_s = (s_1, \ldots, s_k, s_{k+1})$ incrementally. If $k = 0$, i.e., if $X_s = \varnothing$, we create a window of size $\alpha \cdot maxsize$ at every occurrence of $\langle s_{k+1}, t\rangle \in \mathcal{S}$. If $k > 0$, we check for each window in the previous projection if Z_s occurs, and, if it occurs, we take the suffix. We also maintain the original timestamp t at which each window starts. Note that the suffix starts with the first event after time stamp t, whereby we use ϵ to denote the smallest possible period of time between two non-simultaneous events.

Algorithm 2: PROJECT($\mathcal{S}, Z_s, \mathcal{P}_{X_s}, \alpha, maxsize$) computes projection of $Z_s = (s_1, \ldots, s_k, s_{k+1})$ on $\mathcal{S}$ based on projection $\mathcal{P}_{X_s}$ on $X_s = (s_1, \ldots, s_k)$

```
1  $\mathcal{P}' \leftarrow \varnothing$;
2  if $|X_s| = 0$ then
3      $maxwin \leftarrow \lfloor \alpha \cdot maxsize \rfloor$;
4      for $t \leftarrow 1$ to $|\mathcal{S}|$ do
5          if $\mathcal{S}[t] = s_{k+1}$ then
6              $t_a \leftarrow t + \epsilon$;
7              $t_b \leftarrow t + maxwin$;
8              $\mathcal{P}' \leftarrow \mathcal{P}' \cup \langle t, \mathcal{S}[t_a, t_b]\rangle$;
9          end
10     end
11 end
12 else
13     for $\langle t, \mathcal{S}[t_a, t_b]\rangle$ in $\mathcal{P}_{X_s}$ do
14         if $s_{k+1} \in \mathcal{S}[t_a, t_b]$ then
15             $t_a' \leftarrow$ first occurrence of $s_{k+1}$ in
               $\mathcal{S}[t_a, t_b]$;
16             $\mathcal{P}' \leftarrow \mathcal{P}' \cup \langle t, \mathcal{S}[t_a' + \epsilon, t_b]\rangle$;
17         end
18     end
19 end
20 return $\mathcal{P}'$;
```

After the projection is computed we can traverse it to enumerate all possible items to form super-sequences, which is the main goal of the function PROJ_CANDIDATES. Algorithm 3 shows the pseudo code for computing candidate items for forming a super-sequence of length $k+1$ based on the projection of its prefix X_s. The function consists of taking the union of all items found in the suffix of each projected subsequence. In order to compute Y^+ for $C_{maxquan}$ a variant of this function is defined. This is omitted from the pseudo code but is trivial to compute.

Algorithm 3: PROJ_CANDIDATES($\mathcal{S}, X_s, \mathcal{P}_{X_s}$) computes set of possible items $s_{k+1} \in Y$ to generate $Z_s = (s_1, \ldots, s_k, s_{k+1})$ based on projection of $X_s = (s_1, \ldots, s_k)$.

```
1  $Y \leftarrow \varnothing$;
2  for $\langle t, \mathcal{S}[t_a, t_b]\rangle$ in $\mathcal{P}_{X_s}$ do
3      $Y \leftarrow Y \cup \mathcal{S}[t_a, t_b]$;
4  end
5  sort $Y$ on descending support in $\mathcal{P}_{X_s}$;
6  return $Y$;
```

B Proof pruning based on mingap

THEOREM 7.1. *Let X_s be a candidate pattern and Y the set of items that can still be added to X_s. Then, for each $Z_s \in \mathcal{Z}(X_s, Y)$ generated as candidate by Algorithm 1,*

$$C_{quan}(Z_s) = C_{quan}(X_s) = 0$$
$$\textit{if } mingap(X_s) + |Z_{max}| > \alpha \cdot (|X_s| + |Z_{max}|)$$
$$\textit{and } X_s \cap Y = \varnothing,$$
$$\textit{where } |Z_{max}| = \max_{\mathcal{S}[t_a, t_b] \in \mathcal{P}_{X_s}} |suffix(\mathcal{S}[t_a, t_b], X_s)|.$$

Proof. We know that for any window of any super-sequence $Z_s \in \mathcal{Z}(X_s, Y)$, where $X_s \cap Y = \varnothing$ it holds that

$$\forall\ t \in cover(Z_s) : W_t(Z_s) + (|Z_s| - |X_s|) \geqslant W_t(X_s).$$

Furthermore we know, by definition, that

$$\forall\ t \in cover(X_s) : W_t(X_s) \geqslant mingap(X_s).$$

Formally we want to prove that, $\forall Z_s \in \mathcal{Z}(X_s, Y)$,

$$C_{quan}(Z_s) = 0$$
$$\iff \frac{|\{t|\ t \in cover(Z_s) \wedge W_t(Z_s) < \alpha \cdot |Z_s|\}|}{support(Z_s)} = 0$$
$$\iff \nexists t \in cover(Z_s) : W_t(Z_s) < \alpha \cdot |Z_s|.$$

From previous equations, if follows that the smallest minimal window of Z_s is bounded by

$$\min_{t \in cover(Z_s)} W_t(Z_s) \geqslant mingap(X_s) + (|Z_s| - |X_s|),$$

and we can deduce that

$$C_{quan}(Z_s) = 0$$
$$\iff mingap(X_s) + (|Z_s| - |X_s|) \geqslant \alpha \cdot |Z_s|.$$

What remains to be proven is that no pattern $Z_s \in \mathcal{Z}(X_s, Y)$ generated as candidate by Algorithm 1 can be longer than $|X_s| + |Z_{max}|$. In each recursive call, the algorithm adds *exactly* one item to the candidate sequence, and removes *at least* one item from each window in the projection. Therefore, X_s can *at most* grow by the number of items in the largest suffix, and it directly follows that, for each generated candidate Z_s,

$$|Z_s| \leqslant |X_s| + |Z_{max}|.$$

Finally, we can derive that we can prune a candidate pattern X_s and any super-sequence Z_s if

$$mingap(X_s) + |Z_{max}| \geqslant \alpha \cdot (|X_s| + |Z_{max}|),$$

because the inequality becomes tighter proportionally to $(1 - \alpha) \cdot |Z_s|$, which is maximized by $|Z_{max}|$. This concludes the proof.

Note that the restriction $X_s \cap Y = \varnothing$ remains necessary, because otherwise the previous inequality between minimal window length $W_t(X_s)$ and its *extension* $W_t(Z_s)$ does not hold. This *mingap* bound is thus only applicable to prune candidates when no repeating elements are in any suffix of the current projection, that is $X_s \cap Y = \varnothing$.

We illustrate this upper bound with an example. Assume $X_s = (a, b)$ and we have the following set of projected windows:

$$\mathcal{P}_{(a,b)} = \{(\mathbf{a}, \mathbf{b}, d, d, c), (\mathbf{a}, \_, \mathbf{b}, d, c, c), (\mathbf{a}, \_, \_, \mathbf{b}, c, e)\}.$$

The union of the remaining items in the suffixes is $Y = \{c, d\} \cup \{c, d\} \cup \{c, e\} = \{c, d, e\}$. However, we want to count all occurrences of the repeating cs and ds, because $\mathcal{Z}(X_s, Y)$ can contain patterns such as (a, b, c, c) or (a, b, d, d). We therefore take the multiset-union and compute $Y^+ = \{c, c, d, d, e\}$. Using this definition the longest possible pattern would be bound by $|Y^+| = 5$. However, we can further lower this upper bound, by considering each window separately. Since the longest suffix in our example has size 3, we will never be able to add more than 3 items to X_s, which is why we define $|Z_{max}|$ as

$$|Z_{max}| = \max_{\mathcal{S}[t_a,t_b] \in \mathcal{P}_{X_s}} |suffix(\mathcal{S}[t_a, t_b], X_s)|.$$

C Proof pruning based on upper bound quantile-based cohesion

THEOREM 7.2. *Let X_s be a candidate pattern and Y the set of items that can still be added to X_s. Then for each $Z_s \in \mathcal{Z}(X_s, Y)$ generated as candidate by Algorithm 1,*

$$\begin{aligned}
&C_{quan}(Z_s) \leqslant C_{maxquan}(X_s, Y), \text{ where}\\
&C_{maxquan}(X_s, Y) = 1.0 - \\
&\quad \frac{|\{t |\ t \in \beta \wedge W_t(X_s) \geqslant \alpha \cdot |Z'_{max}|\}|}{support(Z'_{max})},\\
&|Z'_{max}| = min(maxsize, |X_s| + |Y^+|),\\
&support(Z'_{max}) = \sum_{i\ \in\ X_s \cup Y} support(i),\\
&\beta = \{t \mid \langle i, t\rangle \in \mathcal{S} \wedge\ i \in X_s \wedge \nexists \langle j, t\rangle \in \mathcal{S} : j \in Y\}.
\end{aligned}$$

Proof. For any pattern Z_s we define support using

$$support(Z_s, \mathcal{S}) = |\{t \mid \langle i, t\rangle \in \mathcal{S} \wedge i \in Z_s\}|.$$

We can partition all items in Z_s in two disjoint sets. Let $Z_s = (X_s, Y_s)$, then the items in Z_s are either in $X_s \backslash Y_s$, or in $Y_s = (Y_s \backslash X_s) \cup (X_s \cap Y_s)$. We can rewrite the previous equation as:

$$\begin{aligned}
support(Z_s, \mathcal{S}) = |\{t \mid \langle i, t\rangle \in \mathcal{S} \wedge i \in X_s \backslash Y_s\} \cup \\
\{t \mid \langle i, t\rangle \in \mathcal{S} \wedge i \in Y_s\}|.
\end{aligned}$$

This is important: as we allow for repetitions, the set $X_s \cap Y_s$ might not be empty. For example, given $X_s = (a, b)$ and $Z_s = (a, b, a)$ an occurrence of item $\langle a, t\rangle$ might have a minimal window as first item in Z_s or as third item in Z_s. This complicates matters since we cannot state that the minimal window at t of X_s is smaller or equal than that of Z_s.

Another issue is caused since we allow multiple items to occur at the same timestamp t. We want to exclude timestamps of items in $X_s \backslash Y_s$ also in Y_s and we refine the previous equation to define the following *disjoint* partition of timepoints:

$$\begin{aligned}
support(Z_s, \mathcal{S}) &= |\{t \mid t \in \beta\} \cup \{t \mid t \in \gamma\}|\\
&= |\{t \mid t \in \beta\}| + |\{t \mid t \in \gamma\}|, \text{ where}\\
\beta_{Z_s} &= \{t \mid \langle i, t\rangle \in \mathcal{S} \wedge i \in X_s \wedge \nexists \langle j, t\rangle \in \mathcal{S} : j \in Y_s\},\\
\gamma_{Z_s} &= \{t \mid \langle i, t\rangle \in \mathcal{S} \wedge i \in Y_s\}.
\end{aligned}$$

We now use this partition to bound the maximal number of minimal windows for any super-sequence. For any $Z_s \in \mathcal{Z}(X_s, Y)$ it holds that

$$\forall\ t \in \beta_{Z_s} : W_t(Z_s) \geqslant W_t(X_s). \qquad (1)$$

In other words, the minimal window length of any super-pattern Z_s is at least as high as the minimal window length of X_s at a timestamp t where an item $i \in X_s$ occurs, but no items from Y_s occur.

Given the current value of *min_coh* (the quantile-based cohesion of the kth pattern in the current top-k, or 0 if we have found fewer than k patterns), we only want to enumerate candidates Z_s where $C_{quan}(Z_s)$ could turn out to be higher than or equal to *min_coh*. We now derive:

$$
\begin{aligned}
C_{quan}(Z_s) &= \frac{|\{t | t \in cover(Z_s) \wedge W_t(Z_s) < \alpha \cdot |Z_s|\}|}{support(Z_s)} \\
&= support(Z_s)^{-1} \cdot \\
&\quad (\, |\{t | \ t \in \beta_{Z_s} \wedge W_t(Z_s) < \alpha \cdot |Z_s|\}| + \\
&\quad |\{t | \ t \in \gamma_{Z_s} \wedge W_t(Z_s) < \alpha \cdot |Z_s|\}| \,) \\
&= 1.0 - support(Z_s)^{-1} \cdot \\
&\quad (\, |\{t | \ t \in \beta_{Z_s} \wedge W_t(Z_s) \geqslant \alpha \cdot |Z_s|\}| + \\
&\quad |\{t | \ t \in \gamma_{Z_s} \wedge W_t(Z_s) \geqslant \alpha \cdot |Z_s|\}| \,). \\
&\leqslant 1.0 - support(Z_s)^{-1} \cdot \\
&\quad (\, |\{t | \ t \in \beta_{Z_s} \wedge W_t(Z_s) \geqslant \alpha \cdot |Z_s|\}|). \qquad (2)
\end{aligned}
$$

We finalize the proof by producing bounds for the main three elements of the above equation. For every candidate Z_s generated by Algorithm 1, we can bound the size of Z_s, the size of β_{Z_s}, and the support of Z_s. First of all, given that Algorithm 1 produces candidates by adding items in Y^+ to X_s, until there are either no more items left to add or we have reached the size threshold *maxsize*, it directly follows that, for every candidate Z_s generated by Algorithm 1,

$$|Z_s| \leqslant min(maxsize, |X_s| + |Y^+|) = |Z'_{max}|. \qquad (3)$$

Second, note that $\beta \subseteq \beta_{Z_s}$, and, therefore

$$
\begin{aligned}
&|\{t | \ t \in \beta_{Z_s} \wedge W_t(Z_s) \geqslant \alpha \cdot |Z_s|\}| \\
&\geqslant |\{t | \ t \in \beta \wedge W_t(Z_s) \geqslant \alpha \cdot |Z_s|\}|. \qquad (4)
\end{aligned}
$$

Since $\beta \subseteq \beta_{Z_s}$, from Equation 1 it follows that

$$\forall \ t \in \beta : W_t(Z_s) \geqslant W_t(X_s),$$

and, therefore,

$$
\begin{aligned}
&|\{t | \ t \in \beta \wedge W_t(Z_s) \geqslant \alpha \cdot |Z_s|\}| \\
&\geqslant |\{t | \ t \in \beta \wedge W_t(X_s) \geqslant \alpha \cdot |Z_s|\}|. \qquad (5)
\end{aligned}
$$

From Equation 3, it follows that

$$
\begin{aligned}
&|\{t | \ t \in \beta \wedge W_t(X_s) \geqslant \alpha \cdot |Z_s|\}| \\
&\geqslant |\{t | \ t \in \beta \wedge W_t(X_s) \geqslant \alpha \cdot |Z'_{max}|\}|, \qquad (6)
\end{aligned}
$$

and, by combining Equations 4, 5 and 6, we obtain

$$
\begin{aligned}
&|\{t | \ t \in \beta_{Z_s} \wedge W_t(Z_s) \geqslant \alpha \cdot |Z_s|\}| \\
&\geqslant |\{t | \ t \in \beta \wedge W_t(X_s) \geqslant \alpha \cdot |Z'_{max}|\}|. \qquad (7)
\end{aligned}
$$

Finally, since any candidate pattern Z_s can only contain items from X_s and Y, it follows that

$$support(Z_s) \leqslant \sum_{i \in X_s \cup Y} support(i) = support(Z'_{max}). \quad (8)$$

From Equations 2, 7 and 8, it now directly follows that

$$
\begin{aligned}
&C_{quan}(Z_s) \\
&\leqslant 1.0 - support(Z_s)^{-1} \cdot \\
&\quad (\, |\{t | \ t \in \beta \wedge W_t(X_s) \geqslant \alpha \cdot |Z'_{max}|\}|) \\
&\leqslant 1.0 - support(Z'_{max})^{-1} \cdot \\
&\quad (\, |\{t | \ t \in \beta \wedge W_t(X_s) \geqslant \alpha \cdot |Z'_{max}|\}|) \\
&= C_{maxquan}(X_s, Y).
\end{aligned}
$$

This concludes the proof.

D Additional Performance Experiments

To evaluate the performance of our pruning technique, we run our algorithm on the *Synthetic* and *Moby* datasets, while varying *maxsize*, and mine the top-k sequential patterns using prefix-projected pattern growth, with and without pruning. In this experiment, α is set to 2.0 and k is set to 20.

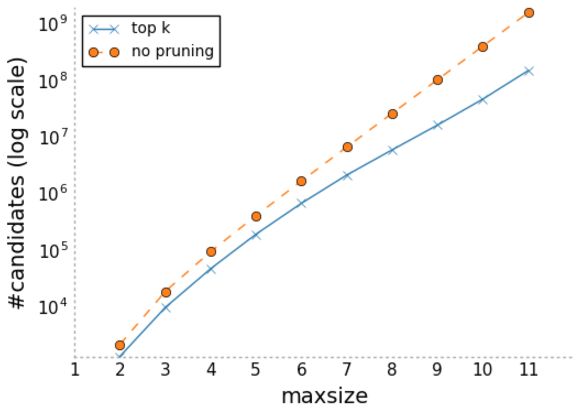


Figure 3: Number of candidates with and without pruning on *Synthetic*.

In Figures 3 and 4 we show the the number of candidates (in log scale) on the *Synthetic* and *Moby* datasets, with and without pruning. We vary *maxsize* between 2 and 11. From these results we conclude that pruning has a significant impact on the number of candidates, which are reduced by an order of magnitude for patterns of larger sizes, thereby also reducing memory consumption. Concerning performance, the number of candidates (and runtime) grows almost *exponentially* with the maximal pattern length, witch follows from the fact that the number of possible candidate patterns also increases exponentially with $\mathcal{O}(\Omega^{maxize})$.

We do not include runtime plots, but for the *Synthetic* dataset the gain in runtime is marginal, taking 2 026 seconds without pruning and 1 880 seconds with pruning for *maxsize* = 11. For the *Moby* dataset the

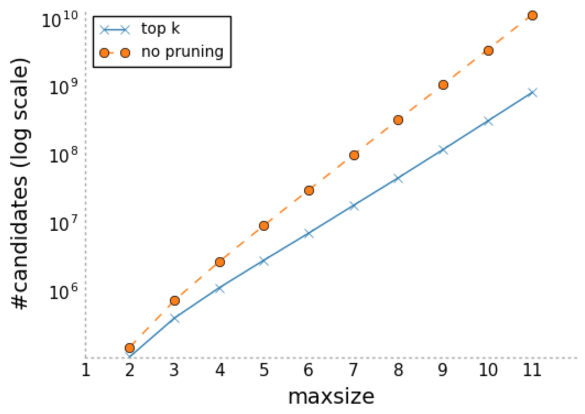


Figure 4: Number of candidates with and without pruning on *Moby*.

gain in runtime is high when *maxsize* is high, taking 196 minutes without pruning and 113 minutes with pruning for *maxsize* = 11. The moderate gain in runtime on the *Synthetic* dataset, and lower values of *maxsize* on the *Moby* dataset, is due to the fact that during each iteration we must compute the pruning functions, thereby computing the number of minimal windows of X_s based on P_{X_s}, which generates overhead. Overall, we conclude that pruning on a larger search space seems to have a large effect on runtime, while for smaller search space, the effect is marginal. We remark that in future work it would be interesting to investigate approximations based on the current bounds, that potentially prune slightly less candidates, but are faster to compute in any setting.

We also ran QCSP for varying the value of k between 10 and 10 000 on *Moby* using a fixed *maxsize* of 5. The effect of a higher k on runtime behavior is not very large, resulting in a runtime of $5s$ for top-10 and slightly increasing to $25s$ for top-10 000.

Finally, we conclude that on the real-world *Moby* dataset where the number of items is $\Omega = 2\,059$ and the total sequence length exceeds 100 000, finding the top-k quantile-based cohesive sequential patterns up to a maximal size of 10 under one hour is acceptable. Our pruning functions reduce the number of candidates with an order of magnitude, and there is a clear effect on runtimes under the assumption that search space of possible candidates is relatively large.

E Top-20 patterns

Tables 4 and 5 show the top-20 patterns for the text datasets. Patterns not found either exactly or as sub-patterns in the top-250 of other methods are shown in bold. We see that QCSP finds many interesting patterns that other methods fail to rank highly. On the other hand, the patterns SkOPUS ranks highly and other methods do not are typically combinations of very frequent items. GoKrimp, in general, produces few patterns, and misses out on many interesting patterns altogether.

Table 4: Top-20 patterns *Moby*.

QCSP	mobi, dick	**um, um**	sperm, whale	**ginger, ginger**
	mrs, hussey	**seven, seventi**	town, ho	**ha, ha**
	ii, octavo	cape, horn	**ii, iii**	**jack, knife**
	crow, nest	o, clock	dough, boy	mast, head
	iii, duodecimo	**hither, thither**	**beef, bread**	**inclin, plane**
SkOPUS	sperm, whale	mobi, dick	said, i	d, ye
	white, whale	captain, ahab	**whale, him**	**ye, see**
	though, yet	right, whale	**look, like**	quarter, deck
	old, man	**whale, head**	cri, ahab	**ahab, him**
	mast, head	**whale, ship**	**one, side**	**now, whale**
GoKrimp	mobi, dick	captain, ahab	cri, ahab	
	sperm, whale	said, i		
	mast, head	right, whale		
	white, whale	quarter, deck		
	old, man	d, ye		

Table 5: Top-20 patterns *JMLR*.

QCSP	**mont, carlo**	reproduc, hilbert	support, machin	**bayesian, network**
	nearest, neighbor	real, world	state, art	high, dimens
	support, vector	**belief, propag**	vector, machin	**collabor, filter**
	http, www	support, vector, machin	**messag, pass**	**naiv, bay**
	cross, valid	**plai, role**	data, set	learn, algorithm
SkOPUS	**paper, show**	**base, result**	**paper, propo**	**paper, set**
	paper, result	**paper, method**	vector, machin	support, machin
	paper, experi	**learn, result**	**paper, new**	**learn, data**
	paper, algorithm	**problem, experi**	**algorithm, result**	**problem, result**
	support, vector	**learn, experi**	**paper, base**	**algorithm, experi**
GoKrimp	support, vector, machin	high, dimens	well, known	
	real, world	neural, network	hilbert, space	
	machin, learn	compon, analysi	experi, result	
	state, art	**supervi, learn**		
	reproduc, hilbert, space	support, vector		

Table 6: Top-20 patterns *Trump*.

QCSP	puerto, rico	**goofi, elizabeth, warren**	**luther, strang**	https, co
	witch, hunt	prime, minist	**stock, market**	fake, news
	elizabeth, warren	goofi, warren	self, fund	**novemb, 8th**
	las, vega	radic, islam	suprem, court	white, hous
	goofi, elizabeth	e, mail	mitt, romney	**common, core**
SkOPUS	make, america	america, again	thank, trump2016	crook, clinton
	make, great	great, again	**thank, makeamer-icagreatagain**	donald, trump
	crook, hillari	make, again	fake, news	last, night
	hillari, clinton	thank, you	2016, fals	**thank, co**
	america, great	https, co	ted, cruz	interview, enjoy
GoKrimp	make, america, great, again	thank, you	trump2016http, co	north, carolina
	https, co	last, night	00, m	north, korea
	crook, hillari	hillari, clinton	new, york	make, america, great
	fake, news	2016, fals	donald, trump	white, hous
	ted, cruz	look, forward	south, carolina	work, hard