information

Article

Dramatically Reducing Search for High Utility Sequential Patterns by Maintaining Candidate Lists †

Scott Buffett

Research Centre for Digital Technologies, National Research Council Canada, Fredericton, NB E3B 9W4, Canada; scott.buffett@nrc.gc.ca

† This paper is an extended version of our paper published in the 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 10–13 December 2018.

Received: 29 October 2019 ; Accepted: 9 January 2020; Published: 15 January 2020

Abstract: A ubiquitous challenge throughout all areas of data mining, particularly in the mining of frequent patterns in large databases, is centered on the necessity to reduce the time and space required to perform the search. The extent of this reduction proportionally facilitates the ability to identify patterns of interest. High utility sequential pattern mining (HUSPM) seeks to identify frequent patterns that are (1) sequential in nature and (2) hold a significant magnitude of utility in a sequence database, by considering the aspect of item value or importance. While traditional sequential pattern mining relies on the downward closure property to significantly reduce the required search space, with HUSPM, this property does not hold. To address this drawback, an approach is proposed that establishes a tight upper bound on the utility of future candidate sequential patterns by maintaining a list of items that are deemed potential candidates for concatenation. Such candidates are provably the only items that are ever needed for any extension of a given sequential pattern or its descendants in the search tree. This list is then exploited to significantly further tighten the upper bound on the utilities of descendent patterns. An extension of this work is then proposed that significantly reduces the computational cost of updating database utilities each time a candidate item is removed from the list, resulting in a massive reduction in the number of candidate sequential patterns that need to be generated in the search. Sequential pattern mining methods implementing these new techniques for bound reduction and further candidate list reduction are demonstrated via the introduction of the *CRUSP* and *CRUSPPivot* algorithms, respectively. Validation of the techniques was conducted on six public datasets. Tests show that use of the *CRUSP* algorithm results in a significant reduction in the overall number of candidate sequential patterns that need to be considered, and subsequently a significant reduction in run time, when compared to the current state of the art in bounding techniques. When employing the *CRUSPPivot* algorithm, the further reduction in the size of the search space was found to be dramatic, with the reduction in run time found to be dramatic to moderate, depending on the dataset. Demonstrating the practical significance of the work, experiments showed that time required for one particularly complex dataset was reduced from many hours to less than one minute.

Keywords: high utility sequential pattern mining; sequential pattern mining; frequent pattern mining; candidate list maintenance

1. Introduction

High utility sequential pattern mining (HUSPM) [1,2] is a subfield of frequent pattern mining [3] that assigns levels of relative magnitude or importance to objects with the goal of identifying more impactful patterns. Frequent pattern mining is a general area of data mining that focuses on the fast identification of objects or items that appear together in a similar fashion with regularity in a transactional database. A number of frequent pattern mining techniques fall under the broader term. Itemset mining techniques [4] aim to identify items that frequently co-exist in events or transactions. Association rule mining attempts to identify rules that dictate when the presence of certain items tends to imply the presence of others [5]. Sequential pattern mining (SPM) [6] introduces a temporal nature to the patterns of interest, allowing for the ability to identify patterns of items that appear across multiple transactions, in a certain order. Thus, while itemset and association rule mining can be useful to help identify, for example, which items retail consumers tend to purchase together when considering product placement strategies, SPM extends the scope of item-based pattern mining to allow for the consideration of, perhaps, the ordering of stock on shelves throughout the entire store. Athlete movement sequences, surgical technique, student learning behaviours, just to name a few potential examples, can hence be identified and examined.

The key shortcoming of the frequent pattern mining methods outlined above is that all items identified in the frequent patterns are considered to have equal significance, with the "interestingness" of a pattern based solely on the number of times it appears. In reality, it is often the case that the most highly frequent patterns are also the least interesting ones. Since it is more commonly desirable to seek out patterns that contain more significant (i.e., more important, profitable, etc.) items, even if they appear less frequently, SPM methods can fall short of the desired intention to identify patterns of high interest.

To illustrate, consider the task of discovering key purchase patterns. Here, it might be far more impactful to identify patterns involving items such as TVs or cars than those involving bread and eggs. Other examples might instead involve covert activity such as malicious behaviour on a network or fraudulent transactions in a banking system, which are rare by their very nature yet extremely valuable to discover. To facilitate the identification of these important yet less frequent patterns, the area of *high utility sequential pattern mining* (HUSPM) has been proposed. To accomplish this, items are assigned a measure of their impact via a numeric *utility*. The goal of HUSPM then draws the focus away from identifying patterns meeting a prespecified frequency, and rather seeks to identify patterns whose sum utility of all of its instances in the database exceeds a prespecified minimum utility.

In order to effectively manage the potentially unwieldy search for the full set of sequential patterns, state-of-the-art SPM algorithms employ the *downward closure property*. Simply put, the downward closure property dictates that, for any sequential pattern sp, if sp is deemed to be frequent (i.e., support exceeds the minimum), then any of its subsequences must also be frequent. The contrapositive of this property can be thus exploited by top-down SPM approaches that search for sequential patterns by extending visited patterns to supersequences of those patterns as children in the search tree, since, for any such pattern that is found to be infrequent, it must be the case that any of its supersequences are also infrequent. Thus, whenever a potential pattern is encountered with support that falls below the minimum, then according to the downward closure property, the frequency of all supersequences of that pattern can be no higher and thus the search can be pruned at that point. This has the benefit of restricting the search and facilitating fast identification of sequential patterns, by offering an upper bound at each node on the support for all supersequences appearing at descendent nodes.

With HUSPM, however, the downward closure property does not hold. In this case, the minimum utility constraint is non-monotonic, as utility might either increase or decrease as a pattern is extended to a supersequence. Thus, the exploration of alternative methods for reducing the search space in HUSPM by eliminating possible extensions is imperative, and has become a key focus of research in the area. As a result, one of the primary objectives in HUSPM has been the development of new methods for identifying upper bounds on the utility of future candidate sequential patterns in the search. As an initial step in this challenge, Ahmed et al. [1] proposed the sequence-weighted downward closure property. This property dictates that an upper bound on the utility of any supersequence of a sequential pattern can be computed using the utilities of the input sequences containing instances of that pattern. The *sequence weighted utility (SWU)* for a sequential pattern sp is thus defined as the sum of the utilities of all items in every sequence that contains sp. Yin et al. [2] and Wang et al. [7] propose additional methods that further tighten upper bounds by considering the utilities of input sequences that contain a candidate pattern, but only using the portion of the sequence that appears after the completion of the first instance of the pattern. Specifically, the *prefix extension utility (PEU)* of a sequential pattern sp in sequence s is introduced as an upper bound on all extensions of that pattern in s, and is computed as the maximum possible sum in a sequence of (1) the utility of sp in s, and (2) all of the "remaining" utilities of items that appear after sp in s, over all instances of the pattern in the sequence. The PEU for sp in a sequence database S is then the sum PEU over all $s \in S$. The *reduced sequence utility (RSU)* for sp given a item i is subsequently proposed as an upper bound of any concatenation of sp with i, and is defined as the sum PEU of sp over all sequences that contain the concatenation of sp with i.

A novel approach, initially sketched in [8] and extended in this paper, is presented that advances the state of the art as follows. Initially, existing bound-based search pruning methods, namely the PEU and RSU approaches discussed above, are extended by a search technique that maintains a list of *candidate concatenation items*. The use of this list has a significant impact on the search process since, for any sequential pattern sp under consideration in the search, only the items in the candidate list associated with sp need ever be considered for concatenation for *any* supersequence of sp with sp as prefix. Thus, even when existing approaches are unable to prune the search entirely at any particular point, and are thus unable to make any progress in reducing the size of the search, a number of potential candidates may still be removed from future consideration. Second, in addition to the reduced search space required as a result of the removal of candidate sequences from consideration, a mechanism is proposed for further tightening the upper bound on potential pattern extensions by exploiting the resulting reduced concatenation item list. This facilitates earlier pruning, which further reduces the size of the search space. The idea is that, if we know that certain items will never appear in any future concatenation of the prefix pattern in question, then we can remove the utilities of those items from the remaining utility portion of PEU computation, thus reducing the upper bound. The resulting reduced upper bound subsequently facilitates further reduction of the candidate list, and so on. Experimental validation on a number of publicly available datasets demonstrates a substantial improvement in terms of both run time as well as the size of search space when compared with existing state-of-the-art pruning techniques.

One drawback to this method is the high computational cost required to maintain the input database. Each time an item is removed from the candidate list, its utilities must subsequently be removed (or otherwise stored for future reference) to facilitate updated computation of the decreased PEU. This limits the number of iterations through the candidate list that can be performed for each sequential pattern in search of potential item removals, since the additional computational cost can eventually outweigh the benefit of reducing the number of candidates. To remedy this, a further advancement is presented in this paper that addresses this cost and facilitates a consequentially dramatic reduction in the search. Initially, a *relaxed* upper bound on the utility of all pattern extensions, referred to as the *pivot-centered prefix extension utility (PPEU)*, is proposed. While this value will always be greater than or equal to the PEU for a particular sequential pattern, seemingly rendering it less effective at pruning, it has the significant benefit that remaining utility values do not need to be maintained at all positions in the database. In fact, they do not need to be kept at all, which is not the case for any of the other bound computation methods mentioned above. For this method, remaining utility values are only required at the pivots, which are easily computed at each step. As a result, for each sequential pattern, the list of candidate items can be traversed multiple times. As candidates are removed, PPEU values for other candidates are decreased, leading to further removals, and so on. The result is an extremely significant reduction in the item candidate list and a corresponding reduction in the number of candidate sequential patterns that need to be generated and tested, often by a factor of more than 10.

Overall, a number of innovations are proposed. These include, specifically:

1. The maintenance of an item candidate list through out the search, which significantly reduces the time and space required to search. This reduction is due to the fact that items not in the list for

a given candidate pattern are provably never required to be considered as possible extensions for that pattern, or for that pattern's extensions.

2. A method for computing an upper bound on the utility of all future extensions of a sequential pattern involving a particular item, referred to as the Reduced Concatenation Utility (RCU). Given a sequential pattern sp and item i, the RCU gives an upper bound on the possible utility of any other sequential pattern that is an extension of sp (i.e., that has sp as prefix) that includes i. If the RCU is found to be below the minimum utility, then i can be removed from the search. By capitalizing on the knowledge of candidate list of eligible items, this bound is proven to be tighter than the best existing bound from the literature, the Reduced Sequence Utility (RSU). This tighter upper bound results in earlier pruning and, thus, faster search.
3. The introduction of a slightly more loose upper bound, referred to as the Pivot-Centered Prefix Extension Utility (PPEU) that is significantly less expensive computationally to compute than RCU, allowing it to more fully exploit the removed candidates and dramatically reduce the required search space. Experiments show that search time is reduced significantly for some datasets.
4. The *CRUSP* (Candidate Reduced Utility-boundedness for Sequential Patterns) and *CRUSPPivot* algorithms, designed to implement search using the RCU and PPEU bound computation methods, respectively.

The paper is organized as follows: Section 2 begins with a review of existing literature in frequent pattern mining including high utility sequential pattern mining, then gives a formal introduction to the sequential pattern mining and high utility sequential pattern mining problems. Section 3 then reviews previous bounding methods and candidate search and pruning methods, with Section 4 then motivating the work presented in this paper by arguing the necessity for candidate generate-and-test approaches for HUSPM, and consequently the requirement for candidate reduction schemes. Section 5 next details the approach proposed in this paper for maintaining a concatenation candidate item list, and demonstrates the usage of this list to derive tighter upper bounds on the utilities of candidate pattern extensions further down the search. The "*CRUSP*" algorithm for performing this search is also described in detail. The pivot-centered method for the fast elimination of candidate items is then presented in Section 6, along with details of the "*CRUSPPivot*' algorithm that implements the technique. Performance of the proposed algorithms is then validated against existing bounding techniques in Section 7 using six publicly available datasets used previously in the literature. Finally, Section 8 concludes the paper with a discussion on findings and results, as well as key research directions for future work.

2. Background

2.1. Literature Review

Early research in frequent pattern mining dates back to the data mining work of Hajek [9]. Agrawal and Srikant [5,6] laid the groundwork for contemporary pattern mining, in particular making seminal contributions to itemset mining and sequential pattern mining with the proposal of the *Apriori* and *AprioriAll* algorithms, respectively. As a necessary component of those search algorithms, the downward closure property (also referred to as the *Apriori* property) was first proposed as a constraint for limiting search, as described in the previous section.

While subsequent algorithms for itemset mining were later developed, such as *FP-Growth* [10] and *Eclat* [11], to name two examples, sequential pattern mining has grown into a highly-regarded field of study of its own in the ensuing years. Srikant and Agrawal [12] proposed the first key extension of *AprioriAll* with the *Generalized Sequential Patterns (GSP)* algorithm, which allowed for the consideration of time periods between transactions, facilitating the identification of patterns that frequently occur within a particular time window. Zaki [13] further improved the time taken to identify the set of sequential patterns with the *SPADE* algorithm, which employs a *vertical* sequence representation that, as a result, requires far fewer accesses to the database. Ayres et al. [14] also employed the vertical format in the *SPAM* algorithm, but advanced the state of the art by introducing the use of bitmaps for the vertical sequence representation. Perhaps the most significant advancement in SPM research was proposed by Pei et al. [15] with the introduction of the *PrefixSpan*, which utilizes a divide-and-conquer strategy, referred to as *pattern growth*, that does not require the generation and testing of a large number of unpromising candidate patterns. This concept is illustrated in depth later in the paper.

The concept of high utility itemset mining rose from work by Yao et al. [16] to address the problem that arises in situations where items should not be treated equally. For example, major purchases such as TVs or cars may happen very infrequently compared to more minor purchases like bread and eggs; however, purchase patterns involving the former might likely be deemed far more interesting than those involving the latter. Since it is possible, and likely regrettable, that frequent itemset mining will miss these highly important events, the proposed solution assigns a *utility* to each item or object, and itemset patterns are subsequently mined whose sum utility meets or exceeds some prespecified minimum. To limit the search required, the *transaction-weighted downward closure property* was proposed by Liu et al. [17]. This property loosely states that the utility of any superset of an itemset is bounded by the sum utility of the input transactions that contain the itemset as a subset, and thus this value can be used to restrict the search. Building upon this work, there have been a number of advances in the area of high utility itemset mining. Fournier-Viger et al. produced a number of advancements, including the proposed the use of co-occurrence pruning to significantly improve search time [18], as well as the concept of representing high utility itemsets using generator patterns [19]. Tseng et al. [20] proposed algorithms for addressing the problem of identifying top-*k* high utility patterns, while Lin et al. [21] explored the concept of high average utility itemsets.

Ahmed et al. [1] proposed the first extension of sequential pattern mining to include the notion of utility with the *UtilityLevel* and *UtilitySpan* algorithms, which utilized the candidate generate-and-test and pattern growth approaches, respectively. These algorithms were the first to exploit the sequence-weighted downward closure property as a means for limiting the search. Here, for each candidate pattern under consideration, the *sequence-weighted utility* is computed as the sum of the total utilities of all input sequences that contain the pattern, offering an upper bound on all supersequences. If this value falls below the minimum allowable utility, the search can be cut off at that point. Yin et al. [2] then outlined a more general framework for the concept of sequential pattern utility, and defined the *High Utility Sequential Pattern Mining* (HUSPM) problem as a special case within their framework, detailing the *uSpan* algorithm as a proposed solution technique. Wang et al. [7] proposed the *HUS-Span* algorithm, and formalized the *prefix extension utility* strategy as a tighter upper bound on the utility of any extensions to a given sequential pattern. Rather than use the total utility of each input sequence in the computation of this value, only the utility of the pattern and the utilities of items appearing *after* the pattern in the sequence are considered.

There have since been a number of advances for problems related to HUSPM. Lin et al. examine the use of multiple minimum utility thresholds [22], borrowing from a related concept in association rule mining that attempts to identify "rare" items in the search [23]. Zhang et al. [24] address the problem of uncertainty in sequence databases. Zihayat et al. [25] consider HUSPM approaches with data streams as input. Xu et al. [26] propose the possibility for negative item values. Zida et al. [27] propose solutions to the problem of identifying high utility sequential rules. Advances in data-driven monitoring and cyber-physical systems [28–30] also highlight the importance of utility-based pattern mining.

While these efforts address related areas and propose solutions to new, relevant problems, this work is believed to be the first to advance the state of the art established by Ahmed et al., Yin et al., and Wang et al. for reducing the search space for general HUSPM search.

2.2. Sequential Pattern Mining

Sequential pattern mining (SPM) [6,31] is a data mining research discipline that focuses on the identification of frequently occurring patterns of items (i.e., objects, events, etc.), where ordering of these items is preserved. The ability to mine such patterns can be advantageous in a number of applications, such as identifying frequent purchase patterns to determine which items tend to be purchased before or after others [32], analysing network activity to capture frequent patterns of both normal and attack behaviour in order to identify anomalies that might indicate malicious activity [33], or even to scrutinize technique being exhibited by a student or trainee to identify consistencies with common flaws and suggest areas for improvement [34].

The sequential pattern mining problem comprises an input set of sequences and has the objective of identifying the full set of subsequences that appear within those input sequences with frequency meeting or exceeding a prespecified minimum. More formally, let I be a set of *items*, and S be a set of *input sequences*, where each $s \in S$ consists of an ordered list of *itemsets*, or sets of items from I, also referred to as *transactions*. A sequence $\langle a_1 a_2 \ldots a_n \rangle$ is said to be *contained* in another sequence $\langle b_1 b_2 \ldots b_m \rangle$ if there exist integers $i_1, i_2, \ldots, i_n$ with $i_1 < i_2 < \ldots < i_n$ such that $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \ldots, a_n \subseteq b_{i_n}$. A sequence $s \in S$ *supports* a sequence s' if s' is contained in s. The support $sup(s')$ for a sequence s' given a set S of input sequences is the portion of sequences in S that support s', and is equal to $sup(s') = |\{s \in S | s \text{ supports } s'\}| \,/ |S|$. A sequence s' is deemed to be a *sequential pattern* if $sup(s')$ is greater than some pre-specified minimum support. Such a pattern with total cardinality of its itemsets summing to n is referred to as an *n-sequence* or *n-pattern*. A sequential pattern s' is deemed to be a *maximal sequential pattern* in a set S' of sequential patterns if $\forall s'' \in S'$, where $s'' \neq s'$, s'' does not contain s'. The objective of the sequential pattern mining problem is to identify the set entire S' of sequences that are considered to be sequential patterns according to the definitions above. In some cases, the set consisting of only maximal sequential patterns is preferred.

To illustrate, consider the example set S of sequences given in the first column of Figure 1. Using a minimum support of 0.4, the set of all sequential patterns is indicated, specified by 1-sequences, 2-sequences, and 3-sequences. Note that it is accepted conventionally to denote sequences without set braces and commas separating internal elements. Thus, $\langle \{b, c\}, \{d\} \rangle$ can be written as $\langle bc, d \rangle$. In addition, note that, while this table gives all of the frequent sequential patterns, only $\langle a, d \rangle$, $\langle c, c \rangle$, $\langle bc, d \rangle$, $\langle b, d, e \rangle$ and $\langle c, d, e \rangle$ are *maximal* sequential patterns.

Sequence Database	1-seq	2-seq	3-seq
$\langle bc, cd, e \rangle$	$\langle a \rangle$	$\langle a, d \rangle$	$\langle bc, d \rangle$
$\langle ac, bc, d \rangle$	$\langle b \rangle$	$\langle bc \rangle$	$\langle b, d, e \rangle$
$\langle c, e \rangle$	$\langle c \rangle$	$\langle b, d \rangle$	$\langle c, d, e \rangle$
$\langle c, d, ef \rangle$	$\langle d \rangle$	$\langle b, e \rangle$	
$\langle ab, d, e \rangle$	$\langle e \rangle$	$\langle c, c \rangle$	
		$\langle c, d \rangle$	
		$\langle c, e \rangle$	
		$\langle d, e \rangle$	

Figure 1. Example sequence database with mined sequential patterns using a minimum support of 0.4.

2.3. High Utility Sequential Pattern Mining

High utility sequential pattern mining (HUSPM) addresses the inherent problem of SPM that all items are treated equally, and that the criteria for identifying any such sequential pattern is strictly frequency based. HUSPM instead allows items to be considered more or less important or significant than others, and selects patterns based on the consideration of both frequency and the level of importance of its members. Put formally, the HUSPM problem comprises a set S_q of input sequences and additionally includes a utility function $u : I \Rightarrow \Re_{\geq 0}$ assigning a utility to each item (also referred to as the *external* utility). The model also optionally allows non-negative quantities to be included for each item in each input sequence transaction. Thus, each member s_q of the set S_q, referred to as a q-sequence, is a sequence $\langle QI_1, \ldots, QI_m \rangle$ over q-itemsets $\{(i_1, q_1), (i_2, q_2), \ldots, (i_n, q_n)\}$, where q_j specifies the quantity of item i_j in the q-itemset. Each element (i, q) of a q-itemset is referred to as a q-item.

Similar to the *contains* relation in the SPM problem, a q-sequence $s_q = \langle QI_1, \ldots, QI_m \rangle$ is said to be a q-supersequence of $s'_q = \langle QI'_1, \ldots, QI'_n \rangle$ (and thus s'_q is a q-subsequence of s) if there exist integers $i_1 \ldots i_n$, ordered from least to greatest, such that $QI'_1 \subseteq QI_{i_1}, \ldots, QI'_n \subseteq QI_{i_n}$. The subsequence relation with q-sequences is denoted by $s'_q \sqsubseteq_q s_q$. In addition, a q-sequence $s_q = \langle QI_1, \ldots, QI_n \rangle$ is said to *match* a sequence $s = \langle I_1, \ldots, I_n \rangle$ if, for all $QI_j = \{(a_1, q_1), (a_2, q_2), \ldots, (a_k, q_k)\}$ and $I_j = \{b_1, b_2, \ldots, b_m\}$, $k = m$ and $a_i = b_i$ for all i. The match relation is denoted by $s_q \sim s$.

The objective of high utility sequential pattern mining is to identify sequential patterns that contain total utility in a sequence database that meets or exceeds a prespecified minimum. Such utility is determined as follows:

Definition 1 (q-item utility). *The utility u_{it} of a q-item (i, q) is the product of the item utility and the quantity:*

$$u_{it}(i, q) = u(i)q. \tag{1}$$

Definition 2 (q-itemset utility). *The utility u_{is} of a q-itemset I_q is the sum of its q-item utilities:*

$$u_{is}(I_q) = \sum_{i_q \in I_q} u_{it}(i_q). \tag{2}$$

Definition 3 (q-sequence utility). *The utility u_{seq} of a q-sequence s_q is the sum of its q-itemset utilities:*

$$u_{seq}(s_q) = \sum_{I_q \in s_q} u_{is}(I_q). \tag{3}$$

Definition 4 (Sequential pattern utility set). *The utility set for a sequential pattern sp in a q-sequence s_q holds the q-sequence utility for every possible match for s contained by (i.e., is a subsequence of) s_q:*

$$us(sp, s_q) = \{u_{seq}(s'_q) | s'_q \sim sp \land s'_q \sqsubseteq_q s_q\}. \tag{4}$$

Definition 5 (Sequential pattern utility). *The utility of a sequential pattern sp in a q-sequence s_q is equal to the maximum element of its utility set:*

$$u_{sp}(sp, s_q) = \max(us(sp, s_q)). \tag{5}$$

The utility of a sequential pattern sp in a q-sequence database S_q is equal to sum of the sequential pattern utilities for all q-sequences in S_q:

$$u(sp, S_q) = \sum_{s_q \in S_q} u_{sp}(sp, s_q). \tag{6}$$

Figure 2 depicts an example input sequence database, with corresponding item utilities provided in a separate table (commonly referenced as a *profit table* [7]). For convenience, q-itemsets are denoted with shorthand notation as $q_1 i_1 q_2 i_2 \ldots$ with commas then separating the q-itemsets in a q-sequence. Thus, the q-sequence $\langle \{(a, 4), (b, 2)\}, \{(c, 3)\} \rangle$, for example, is written as $\langle 4a2b, 3c \rangle$. To illustrate the concept of sequential pattern utility computation, consider the candidate sequential pattern $\langle ab, c \rangle$:

- $\langle ab, c \rangle$ has two matches in sequence 1, $\langle 6a4b, 4c \rangle$ and $\langle 6a4b, 2c \rangle$, which have utilities $6(2) + 4(3) + 4(1) = 28$ and $6(2) + 4(3) + 2(1) = 26$, respectively. Thus, the utility set for $\langle ab, c \rangle$ in sequence 1 is $\{28, 26\}$.
- $\langle ab, c \rangle$ has no match in sequence 2.
- $\langle ab, c \rangle$ has three matches in sequence 3: $\langle 3a4b, 2c \rangle$, $\langle 3a4b, 5c \rangle$ and $\langle 2a9b, 5c \rangle$, yielding the utility set $\{20, 23, 36\}$.
- $\langle ab, c \rangle$ has two matches in sequence 4: $\langle 3a1b, 2c \rangle$ and $\langle 2a3b, 2c \rangle$, yielding the utility set $\{11, 15\}$.

Choosing the maximum for each utility set and summing over the entire database gives a total utility of $28 + 36 + 15 = 79$. If this total meets or exceeds the prespecified minimum, then we deem $\langle ab, c\rangle$ to be a high utility sequential pattern.

a	b	c	d	e	f
2	3	1	2	3	4

ID	Sequence
1	$\langle 6a4b, 2b4c, 3d, 2c1e\rangle$
2	$\langle 2a, 3b2d, 1e3f, 2b\rangle$
3	$\langle 3a4b, 2c, 2a9b, 3f, 5c\rangle$
4	$\langle 2f, 3a1b, 2a3b, 5d, 2c, e\rangle$

Figure 2. Example profit table and sequence database.

3. Existing Bounding and Search Space Pruning Strategies

3.1. Lexicographic Tree Search

In order to present an overview techniques for bounding and search space pruning, the common tree structure is first demonstrated. As with their frequency-based sequential pattern counterparts, the use of a lexicographic tree structure [14] is employed by many existing HUSPM algorithms. With this sort of tree structure, each node represents a potential sequential pattern in the search (with the root node representing the empty pattern $\{\}$), with each child node representing an *extension* of the parent node's pattern. An extension of a pattern always consists of that same pattern with a single item appended to the end. There are two ways to accomplish this sort of extension:

- *i-extensions*: an item is added to the final itemset of the pattern,
- *s-extensions*: an item is added to a newly added itemset appended to the end of the pattern.

To illustrate, for a sequential pattern $sp = \langle x, y\rangle$, $sp = \langle x, yz\rangle$ would be an example i-concatenation while $sp = \langle x, y, z\rangle$ would be an example s-concatenation. Adjacent children of a sequential pattern in the tree are ordered lexicographically from left to right, with the i-concatenation children appearing to the left of the s-concatenation children. Itemsets themselves are also lexicographically ordered, so only those items that are higher in the lexicographical order than the final item in the final itemset are explored for potential i-concatenation. Yin et al. [2] provide an example implementation of the lexicographic tree for HUSPM for the *uSpan* algorithm, while Wang et al. [7] does so similarly for the *HUS-Span* algorithm.

With the tree established, search proceeds top down, beginning at the empty pattern node and continuing to the 1-sequence nodes, 2-sequence nodes, etc. Clearly, without proper pruning techniques, the tree can become quickly unwieldy, as the number of possible i-extensions for a node representing sp is equal to the number of items in I that follow the maximum item in the final itemset of sp in the lexicographic order, while the number of possible s-extensions is equal to $|I|$. Thus, early pruning is key to efficient search.

3.2. Existing Pruning Strategies for HUSPM

As mentioned above, the downward closure property, which can be used effectively to limit the search in SPM, does not apply to HUSPM. As a first attempt to limit search in HUSPM, Ahmed et al. [1] proposed the sequence-weighted utilization factor (SWU) for a sequential pattern, formally defined as follows:

Definition 6 (sequence-weighted utilization factor (SWU))**.** *Given a sequential pattern sp and a q-sequence database S_q, with each $s_q \in S_q$ having total utility $u_{seq}(s_q)$, and given $S'_q \subseteq S_q$ as the set $\{s'_q \in S_q | \exists s_q \sqsubseteq s'_q \wedge s_q \sim sp\}$ of all q-sequences from S_q containing a match for sp, the sequence-weighted utilization factor (SWU) for sp is computed by*

$$swu(sp) = \sum_{s'_q \in S'_q} u_{seq}(s'_q). \tag{7}$$

Since it is not possible for a supersequence of sp to reside in any q-sequence in $S_q \setminus S'_q$, the sum utility of the sequences in S'_q must necessarily constitute an upper bound on the utilities of sp's supersequences (see Ahmed et al. [1] for a formal proof). Thus, if $swu(sp)$ is less than the minimum utility, it follows that the utility of any supersequence of sp must be less than the minimum. As a result, there is no need to continue search and the lexicographic tree can be pruned at that point.

HUSPM search using SWU as an upper bound was implemented in the *UtilityLevel* and *UtilitySpan* algorithms by a step in which the SWU was computed for each pattern in the tree. Such patterns with SWU exceeding the minimum were deemed *high SWU* patterns, and only these patterns were deemed to be promising, i.e., of having any chance of being considered a high utility sequential pattern. All candidates were scanned to identify those with sufficient utility at a later stage. This pruning technique was also implemented in *uSpan* in the candidate generation stage.

While use of the SWU was demonstrated as an effective search pruning mechanism, it is clear that use of the total utility over entire input q-sequences is likely to offer an unnecessarily high upper bound on utility in practice. This is due to the fact that, typically, high utility sequential patterns are significantly smaller in size (and utility) when compared with the size of the input q-sequences. To accommodate this property and consequently further tighten this upper bound, the *prefix extension utility (PEU)* [2,7] and the *reduced sequence utility (RSU)* [7] measures were proposed, and are defined as follows:

Definition 7 (prefix extension utility (PEU))**.** *Let sp be a sequential pattern and S_q be a q-sequence database. In addition, let $p(sp, s_q)$ be the set of ending positions for sp in s_q (referred to as the* extension *positions), let $u_{sp}(sp, s_q, p)$ be the maximum utility of all instances of sp appearing in s_q that end at p, let $s_q/_{(sp,p)}$ be the* remaining sequence *of s_q with respect to the instance of sp that ends in position p (i.e., the rest of the sequence that appears after sp) and let $u_{seq}(s_q/_{(sp,p)})$ be the sum utility of this remaining sequence. The prefix extension utility (PEU) for sp in s_q is computed by:*

$$peu(sp, s_q) = \max_{p \in p(sp, s_q)} \{u_{sp}(sp, s_q, p) + u_{seq}(s_q/_{(sp,p)})\}. \tag{8}$$

The PEU for sp is then

$$peu(sp) = \sum_{s_q \in S_q} peu(sp, s_q). \tag{9}$$

Definition 8 (reduced sequence utility (RSU))**.** *Given a sequential pattern sp and a candidate item i, which when concatenated with sp gives a new sequence denoted by $sp + i$, the reduced sequence utility (RSU) for $sp + i$ in a sequence s_q is computed by:*

$$rsu(sp, i, s_q) = \begin{cases} peu(sp, s_q), & \text{if } sp + i \sqsubseteq_q s_q, \\ 0, & \text{otherwise,} \end{cases} \tag{10}$$

where $\sqsubseteq_q$ denotes the sequential pattern subsequence relation with q-sequences. That is, $s \sqsubseteq_q s_q$ iff there exists a q-sequence s'_q such that $s \sim s'_q \wedge s'_q \sqsubseteq_q s_q$. The RSU for $sp + i$ is then

$$rsu(sp, i) = \sum_{s_q \in S_q} rsu(sp, i, s_q). \tag{11}$$

Since $rsu(sp, i, s_q)$ is equal to the maximum possible sum of (1) sp's utility in s_q and (2) the maximum possible increase in utility that would be the result of any concatenation (taken as the utility of all items in s_q appearing after the earliest instance of sp), $rsu(sp, i)$ gives an upper bound on the utility in s_q of any extension of sp where the extension contains i. In addition, since $rsu(sp, i)$ gives the maximum such utility in the entire database, if $rsu(sp, i)$ is below the minimum threshold, i can then be deemed an unpromising candidate (see [7] for proofs).

Once the RSU stage is completed and extension nodes are constructed for promising candidates, the PEU value for each extension can be computed in an effort to determine whether any search is further necessary. Specifically, if the PEU value for an extension is found to be below the minimum utility, then the corresponding node can be declared a leaf, and the search can thus be pruned at that point.

To further illustrate the utilization of PEU and RSU, consider the example database and profit table previously presented in Figure 2. Let $sp = \langle b \rangle$, and let c be the item under consideration for potential s-concatenation, resulting in the new pattern $sp + i = \langle b, c \rangle$. In addition, let the minimum utility for this example be 120. PEU and RSU are then computed and utilized as follows. Consider sequence s_1, which contains both $\langle b \rangle$, and c. Since $sp = \langle b \rangle$ contains only a single item, the ending positions are simply the positions of q-itemsets that contain b, and thus $p(\langle b \rangle, s_1) = \{1, 2\}$ (for demonstration purposes, position numbering here begins with 1). The PEU for s_1 is then computed as follows:

- for $p = 1$: $u_{sp}(\langle b \rangle, s_1, 1) = 12$, $s_1/_{(\langle b \rangle, 1)} = \langle 2b4c, 3d, 2c1e \rangle$, and $u_{seq}(s_1/_{(\langle b \rangle, 1)}) = 21$,
- for $p = 2$: $u_{sp}(\langle b \rangle, s_1, 2) = 6$, $s_1/_{(\langle b \rangle, 2)} = \langle 4c, 3d, 2c1e \rangle$, and $u_{seq}(s_1/_{(\langle b \rangle, 2)}) = 15$.

Thus, $peu(\langle b \rangle, s_1) = \max\{12 + 21, 6 + 15\} = 33$. The PEU values are similarly calculated for s_2, s_3, and s_4, yielding the values 34, 62, and 31, respectively. RSU for $sp + i = \langle b, c \rangle$ is then computed as the sum of the PEU values for all sequences that contain $\langle b, c \rangle$. Since all but s_2 contain $\langle b, c \rangle$, $rsu(\langle b \rangle, \langle b, c \rangle) = 33 + 0 + 62 + 31 = 126$. Since $126 \geq 120$, c is considered a promising candidate for concatenation, and $\langle b \rangle$ is extended to $\langle b, c \rangle$.

After creating a node for $\langle b, c \rangle$, pruning proceeds to the leaf evaluation phase, where the PEU value for the new pattern is computed. The PEU value will yield an upper bound on the utility of any further extensions, and thus will help dictate whether the new node is a leaf and thus search can be pruned. Again considering sequence s_1, note the two ending positions for $\langle b, c \rangle$, specifically $\{2, 4\}$. One can see that the maximum utility of $\langle b, c \rangle$ is 16 and 14 at those two ending positions, respectively. Incorporating the remaining utility residing in the sequences following those two ending positions yields a PEU value of $peu(\langle b, c \rangle, s_1) = \max\{16 + 11, 14 + 3\} = 27$. The PEU values are similarly calculated for s_2, s_3 and s_4, yielding values 0, 62 and 14, respectively, giving a total PEU value of $peu(\langle b, c \rangle) = 27 + 0 + 62 + 14 = 103$. Since 103 is the highest possible utility of any further extension of $\langle b, c \rangle$, and 103 falls below the minimum threshold of 120, clearly the search can be terminated at this point and the node corresponding to $\langle b, c \rangle$ can be declared a leaf.

4. Candidate Generate-and-Test vs. Pattern Growth Methods

Many existing sequential pattern mining methods, including early methods such as *AprioriAll* and *GSP*, utilize a candidate generate-and-test approach where candidate sequential patterns are constructed, typically by building upon smaller frequent patterns, which are then tested for support. If the minimum support is not achieved, then no further candidates are generated based on that pattern. Otherwise, the search continues. This approach can be extremely costly in many cases due to the potentially large number of infrequent patterns that need to be generated before they can be ruled out. To mitigate this wasteful exploration of the search space, Pei et al. [15] proposed the concept of *pattern growth* approaches to sequential pattern search with the introduction of the *PrefixSpan* algorithm.

The pattern growth method keys on a divide-and-conquer approach that drastically reduces the size of the search space when attempting to construct new frequent patterns by building on smaller ones. This is achieved by considering only smaller segments of the input sequences each time a sequential pattern is considered for extension. These segments are collectively referred to as the *projected database*. Specifically, given a sequential pattern sp, referred to as a *prefix*, the projected database contains only those parts of the input sequences that can contain a possible *postfix* for sp. Thus, the search for new frequent patterns by extending sp is reduced to this new projected database. To illustrate, consider the example input database given in Figure 3, with corresponding projected database for prefix $\langle a, b, c \rangle$ given in Figure 4.

ID	Sequence
1	$\langle a, b, a, c, f, b, c, d \rangle$
2	$\langle a, d, b, bc, d, e \rangle$
3	$\langle bd, ac, c, a \rangle$
4	$\langle de, ab, bcd, c, abd, d \rangle$
5	$\langle e, a, df, be, c \rangle$

Figure 3. Example input sequences for demonstration of the pattern growth approach.

ID	Sequence
1	$\langle f, b, c, d \rangle$
2	$\langle d, e \rangle$
4	$\langle abd, d \rangle$

Figure 4. Projected database for prefix $\langle a, b, c \rangle$.

As a result, the search for sequential patterns with prefix $\langle a, b, c \rangle$, is reduced to a search for frequent items in the projected database. If the minimum support is, say, 0.4 for example, one can quickly see that b and d are frequent (appearing in at least two sequences) in the projected database, and thus can be appended to $\langle a, b, c \rangle$, producing sequential patterns $\langle a, b, c, b \rangle$ and $\langle a, b, c, d \rangle$. There is no need to consider any other extensions. Refer to [15] for a more formal treatment as well as proofs of various properties.

While pattern growth can be a very effective search method for frequent sequential patterns, unfortunately the concept does not extend very well to the high utility sequential pattern problem, particularly when utilizing prefix extension utility to control the search. This is due to the fact that multiple instances of the same pattern in a sequence cannot be treated equally, and thus one cannot build on a pattern simply by looking at the remainder of an input sequence that occurs after the first instance of a pattern. Consider the first sequence in the sample database above, this time with example utility values included:

$$\langle 2a_1, 6b_2, 16a_3, 4c_4, 1f_5, 6b_6, 5c_7, 6d_8 \rangle$$

Assume the number associated with each item to be the total utility (i.e., weight × quantity) for that item. Notice that all items are subscripted to facilitate differentiation among the multiple instances of a particular pattern. For example, there are four instances of $\langle a, b, c \rangle$: $\langle 2a_1, 6b_2, 4c_4 \rangle$, $\langle 2a_1, 6b_2, 5c_7 \rangle$, $\langle 2a_1, 6b_6, 5c_7 \rangle$ and $\langle 16a_3, 6b_6, 5c_7 \rangle$. Examining the projected sequence for $\langle a, b, c \rangle$:

First, instance of prefix: $\langle 2a_1, 6b_2, 4c_4 \rangle$,
Projected sequence: $\langle 1f_5, 6b_6, 5c_7, 6d_8 \rangle$.

One can see that, for the instance of $\langle a, b, c \rangle$ with maximum utility in the sequence, namely $\langle 16a_3, 6b_6, 5c_7 \rangle$ (utility = 27), one element ($16a_3$) appears neither in the prefix part of the sequence nor the projected part, and thus cannot be identified in this manner. In fact, there is no way to choose a single instance of $\langle a, b, c \rangle$ upon which to build a projected sequence, since different high utility sequential patterns with $\langle a, b, c \rangle$ as prefix can rely on different instances of $\langle a, b, c \rangle$ to maximize utility. Consider $\langle 16a_3, 6b_6, 5c_7, 6d_8\rangle$ and $\langle 2a_1, 6b_2, 4c_4, 1f_5\rangle$ as such examples of maximal instances for $\langle a, b, c, d\rangle$ and $\langle a, b, c, f\rangle$, respectively that utilize different instances of $\langle a, b, c\rangle$ for their prefix.

As far as computing the PEU value for $\langle a, b, c\rangle$, one can observe that the utility plus remaining values for each instance of $\langle a, b, c\rangle$ are $12 + 18 = 30$, $13 + 6 = 19$, $13 + 6 = 19$ and $27 + 6 = 33$, respectively, which should result in a PEU value of $peu(\langle a, b, c\rangle) = \max\{30, 19, 19, 33\} = 33$. However, this value cannot be ascertained from the projected database depicted above.

This demonstrates that pattern growth approaches are not directly applicable to the HUSPM search problem, particularly with PEU-based pruning, demonstrating the necessity for candidate generate-and-test schemes. One should note that pattern growth-based algorithms for HUSPM are indeed feasible, as Ahmed et al. showed with the *UtilitySpan* algorithm [1]. However, it should pointed out that this algorithm did not employ a PEU-based scheme and rather used the inferior SWU pruning method, which considers the utility of entire input sequences in its computation, regardless of the projected database. Thus, to exploit the advantages of PEU-based schemes for search space reduction, candidate generate-and-test schemes are necessary for effective HUSPM. As a result, a primary research problem in HUSPM, and key theme of the work proposed here, focuses on the reduction of the number candidates that need to be generated in order to identify the full set of sequential patterns.

5. Concatenation Candidate List Maintenance

5.1. PEU-Based Candidate Maintenance

This section introduces the concept of candidate list maintenance in HUSPM. To facilitate this, the concept of *maximum concatenation utility* (MCU) is introduced to determine whether an candidate item for concatenation can be eliminated from future consideration. By capitalizing on the knowledge that certain items can be removed from consideration for any future extensions, a reduced upper bound on the utility of future patterns can be computed. Then, by capitalizing on this reduced upper bound, further removal of additional items from consideration may then be possible later in the search, leading potentially to further reductions on the upper bound.

Note that, for convenience, the q subscript for q-sequences is dropped hereafter since there is no further need to disambiguate from traditional sequences.

Definition 9 (maximum concatenation utility (MCU))**.** *Let sp be a sequential pattern, i an item and S the set of input q-sequences. In addition, let $p(sp, s)$ be the set of ending positions for sp in s. The* maximum concatenation utility *(MCU) of i for sp in a sequence $s \in S$ is computed by*

$$mcu(sp, i, s) = \begin{cases} peu(sp, s), & \text{if } sp \sqsubseteq_q s \wedge \langle i\rangle \sqsubseteq_q s \wedge \min(p(sp, s)) < \max(p(\langle i\rangle, s)), \\ 0, & \text{otherwise}, \end{cases} \tag{12}$$

The overall MCU for sp is then computed by:

$$mcu(sp, i) = \sum_{s \in S} mcu(sp, i, s). \tag{13}$$

To validate the use of MCU as an upper bound, a formal proof is given to show that the utility of any i-concatenation or s-concatenation of an item i with a sequential pattern sp, or any descendant sequential pattern of sp in the lexicographical tree (i.e., any supersequence of sp with sp as prefix that includes i in the extension), will be less than or equal to $mcu(sp, i)$:

Theorem 1. *For any sequential pattern sp and item i, $mcu(sp, i)$ is greater than or equal to the utility $u_{seq}(sp')$ of any proper descendant sequential pattern sp' in the lexicographical tree that is the result of a concatenation with i.*

Proof. Suppose sp' is such a descendant pattern with higher utility than $mcu(sp, i)$. Then, there exists a sequence s in the database such that either (1) $u_{seq}(sp', s) > peu(sp', s)$ where the conditions in Equation (12) are true, or (2) $u_{seq}(sp', s) > 0$, where one or more conditions are false.

For case (1), since sp' is a descendant of sp, sp is a prefix for sp'. Since $peu(sp', s)$ gives the maximum possible utility of any sequential pattern with prefix sp in s, $u_{seq}(sp', s) \leq peu(sp', s)$. For case (2):

- Assume $sp \sqsubseteq_q s$ is false. Since sp' contains sp, s will not contain sp' and thus $u(sp', s) = 0$.
- Assume $\langle i \rangle \sqsubseteq_q s$ is false. Since sp' is a result of i-concatenation with i and thus contains i, s will not contain sp' and thus $u(sp', s) = 0$.
- Assume $\min(p(sp, s)) \geq \max(p(\langle i \rangle, s))$. Then, the first ending position of sp appears after the last position of i, so sp' cannot be contained in s, and thus $u(sp', s) = 0$. $\Rightarrow\Leftarrow$

□

To keep track of which items yield an MCU that meets or exceeds the minimum utility, and thus are eligible for concatenation further down the search, an *MCU-candidate list* is maintained. This list is then passed down to the child nodes of sp, informing the subsequent search down the tree that these items are unpromising and resources should not be wasted considering them.

A significant benefit of this candidate elimination is that an even tighter upper bound on the utility of future concatenations can be computed as a result. This is due to the fact that the remaining utility component of the PEU calculation (i.e., the $u_{seq}(s_q/_{(sp,p)})$ term in Equation (8)) can decrease as items are eliminated, since they no longer need to be considered as part of $s_q/_{(sp,p)}$. To illustrate, consider again the example database given in Figure 2. Let $sp = \langle a \rangle$, let a be an item under consideration for s-concatenation, and let the minimum utility be 145. The PEU value is then $peu(\langle a \rangle, a) = 0 + 0 + \max\{68, 48\} + \max\{37, 28\} = 105$. Since 105 is below the minimum allowable utility of 145, we can safely conclude that a can never be concatenated to any subsequent sequence with $\langle a \rangle$ as prefix, and thus a's utilities can be removed from subsequent upper bound computations. Consider then a subsequent extension of $\langle a \rangle$ later in the search, namely $\langle ab \rangle$, with s-concatenation candidate c. A check of the original database will reveal a PEU value of $peu(\langle ab \rangle, c) = 45 + 0 + \max\{68, 48\} + \max\{37, 28\} = 150$. However, since it is known that a subsequent a cannot possibly appear after the ab in any high utility sequential pattern with prefix $sp = \langle ab \rangle$, any utilities associated with a, specifically the utilities associated with the $2a$ appearing in the third position of both the third and fourth sequences, can be removed from the computation of the remaining sequence utility $u_{seq}(s_q/_{(sp,p)})$. Thus, we can further tighten the upper bound on the s-concatenation of c to $45 + 0 + \max\{68 - 4, 48\} + \max\{37 - 4, 28\} = 142$. Since 142 falls below the minimum of 145, c can be declared as unpromising and removed from consideration for $\langle ab \rangle$ and all of its descendants. This reduced upper bound is referred to as the *reduced concatenation utility* (RCU), and is defined below.

Before formally defining RCU, the *removed utility* function must first be introduced. The RCU definition follows. Let the MCU-candidate list (MCL) be the list $mcl(sp)$ of items that have been deemed promising for concatenation with sp, and let the MCU-removal list (MRL) be the list $mrl(sp) = I \setminus mcl(sp)$ of those that have been removed as unpromising (i.e., those items that yielded an MCU value below the limit). Then, the removed utility $ru(sp, s, pos)$ is the total utility of the items in $mrl(sp)$ that appear in position pos in sequence s. Thus, this gives the utilities that are not relevant when computing the upper bound on the utility in s of any descendant of sp. The total removed utility *after* position pos in sequence s is then:

$$ru_{rest}(sp, s, pos) = \sum_{k=pos+1}^{|s|} ru(sp, s, k). \tag{14}$$

Definition 10 (reduced concatenation utility (RCU))**.** *Let sp be a sequential pattern, i an item and S the set of input q-sequences. In addition, let $p(sp,s)$ be the set of ending positions for sp in s. The reduced concatenation utility (RCU) of i for sp in a sequence $s \in S$ is computed by*

$$rcu(sp,i,s) = \begin{cases} \max\limits_{p \in p(sp,s)} \{u_{sp}(sp,s,p) + u_{seq}(s/_{(sp,p)}) - ru_{rest}(sp,s,p)\}, & \text{if } sp \sqsubseteq_q s \wedge \langle i \rangle \sqsubseteq_q s \wedge, \\ & \min(p(sp,s)) < \max(p(\langle i \rangle, s)), \\ 0, & \text{otherwise}. \end{cases} \tag{15}$$

The overall RCU for sp is then computed as:

$$rcu(sp,i) = \sum_{s \in S} rcu(sp,i,s). \tag{16}$$

The RCU is now shown to find unpromising candidates by providing an upper bound on the utility of any descendant.

Theorem 2. *For any sequential pattern sp with MCL-removal list $mrl(sp)$ and* candidate concatenation item *i, if $rcu(sp,i)$ is less than the minimum utility, i can be safely added to $mrl(sp')$ for any descendant sequential pattern sp' of sp.*

Proof. By induction on the size of $mrl(sp)$. *Base case*: Let $mrl(sp) = \phi$. Then, $ru_{rest}(sp,s,p) = 0$ for any s, and thus by Equations (8) and 15, $rcu(sp,i) = mcu(sp,i)$. According to Theorem 1, if $mcu(sp,i)$ is less than the minimum utility, any descendant of sp concatenated with i will have utility less than the minimum, and thus i can be eliminated and added to $mrl(sp')$. *Inductive step*: Assume all items in $mrl(sp)$ have been safely eliminated, and thus they all belong to $mrl(sp')$. Since $mcu(sp,i,s) = peu(sp,s) = \max_{p \in p(sp,s)} \{u_{sp}(sp,s,p) + u_{seq}(s/_{(sp,p)})$ is an upper bound on the utility of any future concatenation of i in sequences where the conditions in Equation (8) are met, then $\max_{p \in p(sp,s)} \{u_{sp}(sp,s,p) + u_{seq}(s/_{(sp,p)}) - ru_{rest}(sp,s,p)$ must also be an upper bound for each sequence when the conditions are met (and 0, otherwise), since the items in $mrl(sp)$ have been safely eliminated as possibilities, meaning that the utility $ru_{rest}(sp,s,p)$ of those items can be safely subtracted. Thus, if $rcu(sp,i)$ is less than the minimum utility, i can be safely eliminated and added to $mrl(sp')$. □

We show empirically in Section 7 that RCU provides a generally tighter upper bound than RSU by demonstrating that significantly fewer candidates are considered in the search.

5.2. The CRUSP Algorithm

To implement the candidate list maintenance and utility bound reduction proposed in this paper, the *CRUSP* (Candidate Reduced Utility-boundedness for Sequential Patterns) algorithm has been developed, and is outlined as follows.

In order to reduce the time taken to compute the RCU values at each step, two structures are maintained for each in put sequence: a *removed-utilities* list (RUL) and a *removed-utility-positions* list (RUPL). These lists are used to keep track of the utilities in a sequence that can be ignored as a result of the elimination of items from the MCL, and the associated positions in the sequences in which those items reside. For example, if items b and c have been eliminated, for sequence 1 in Figure 2, the *removed-utilities* list would hold $\langle 12, 10, 2 \rangle$ while the *removed-utility-positions* would hold $\langle 1, 2, 4 \rangle$, indicating that utilities 12, 10 and 2 were eliminated from positions 1, 2 and 4, respectively.

The algorithm is outlined in Figure 5. In the initial step, sp undergoes a "leaf node" test. Here, if the PEU value for sp is found to lie below the minimum utility, the node corresponding to sp is declared a leaf and the search is pruned at that point. In step 2, a special *iCandList* is maintained, which may contain items that have been eliminated as promising for itemsets after sp but could still be i-concatenated with sp itself only (i.e., added to sp's final itemset). Step 3 takes all promising candidates from *iCandList*, and stores them in *iList*. In step 4, the RCU is computed for each item in the MCU-candidate list, with all items yielding a value below the limit subsequently being removed. Since the remaining items are exactly those that are deemed promising for s-concatenation with sp, these are moved to the *sList* in step 5. The *removed-utilities* and *removed-utility-positions* lists are then updated in step 6 to reflect the utilities that can be subsequently disregarded as a result of the elimination conducted in step 4. Note that this step is quite computationally expensive, so it is only conducted if a sufficient number of items remain to make it worthwhile. In this implementation, a threshold of 70% was chosen such that RUL and RUPL are updated only if the percentage of items removed in step 4 falls below this threshold. Steps 7–14 then create the next level of candidate sequential patterns via i-concatenation and s-concatenation, determine the utilities, ending positions and remaining utility values, and recursively call the algorithm. Finally, step 15 outputs the high utility sequential patterns.

```
CRUSP(sp, sp_util, sp_pos, sp_ru, iCandList, MCL, RUL, RUPL)

 1. If this node is a leaf, then return
 2. Remove low RCU items from iCandList
 3. Put promising candidates for i-concatenation in iList
 4. Remove low RCU items from MCL
 5. sList = MCL
 6. If sufficient items remain, update RUL and RUPL
 7. For each item i in iList:
 8.     $sp'$ = i-concatenate($sp, i$)
 9.     Construct sp'_util, sp'_pos, sp'_ru
10.     CRUSP($sp'$, sp'_util, sp'_pos, sp'_ru, iCandList, MCL, RUL, RUPL)
11. For each item i in sList:
12.     $sp'$ = s-concatenate($sp, i$)
13.     Construct sp'_util, sp'_pos, sp'_ru
14.     CRUSP($sp'$, sp'_util, sp'_pos, sp'_ru, iCandList, MCL, RUL, RUPL)
15. if $u(sp) > u_{min}$, then output sp
16. return
```

Figure 5. The *CRUSP* algorithm.

6. Candidate List Maintenance with Repeated Item Traversal

6.1. Pivot-Centered PEU-Based Candidate Maintenance

As mentioned previously, there is a high cost associated with computing the reduced candidate utility (RCU) for a sequential pattern, mainly due to the repeated updating of utility values that need to be removed at various positions in the database as a result of the elimination of candidate items. This is, however, required in order to obtain an updated PEU value for each potential candidate. As demonstrated in Section 5.2, the algorithm that implements the lower bound in the search is somewhat limited. Specifically, in step 4, each candidate item is examined only once for possible removal, and, even then, utility values are only updated as a result of any removals (step 6) when conditions deem it worthwhile. Ideally, we would like to update utilities every time to achieve maximum reduction in the search space. Moreover, it would be even more beneficial to repeat step 4 in a loop each time these utilities are updated, until no more item removals are possible. To illustrate this last point, consider the example set of sequences with item utilities in Figure 6. Let $\langle a, b, c \rangle$ be the sequential pattern being considered for expansion with candidate list $\{d, e, f\}$, and let the minimum utility be 40.

ID	Sequence
1	$\langle 2a, 1b, 4e, 10a, 2c, 2b, 1c, 5d3e \rangle$
2	$\langle 5a2f, 3e, 3b, 5c, 4e, 2f \rangle$
3	$\langle 2f, 4a, 6b2d, 4c, 2d, 4f \rangle$

Figure 6. Example input sequences.

Step 4 of the *CRUSP* algorithm calls for unpromising candidate items to be culled using the RCU method described above, which yields the following RCU values for each item: $d : 41$ (i.e., $\max\{5 + 11, 13 + 8\}$ in sequence 1 and $14 + 6$ in sequence 3), $e : 40$ and $f : 39$. Since f's value is below the minimum utility, we know it can never be part of a high utility sequential pattern with $\langle a, b \rangle$ as prefix, and is removed from the list. Step 6 then updates the removed utilities and positions lists to reflect the fact that f's utilities are no longer relevant, which are then passed down to the extensions of $\langle a, b, c \rangle$ (i.e., with d and e).

Before generating and testing these extensions, however, we should note that the removal of f's utilities has the effect of decreasing the RCU value for d to 37, since a utility of 4 is removed from sequence 3, which thus falls below the utility threshold of 40, facilitating the removal of d. Further removal of d's utilities reduces the RCU for e to 33, which is also eliminated. With an empty candidate list, search can be pruned at this point. This serves as a demonstration that the search for high utility sequential patterns can indeed be minimized by looping back to update RCU values whenever candidate items have been eliminated. However, as mentioned above, updating all of an item's utilities upon removal to facilitate updated RCU computation is too costly.

As a solution to this problem, a relaxed upper bound on the value of an item extension, referred to as the *pivot-centered prefix extension utility* (PPEU), is proposed. In contrast to the PEU, where the sum of prefix utility and the remaining utility value is maximized over all instances of the prefix in a sequence, the PPEU value takes the sum of the maximum prefix utility over all instances, and the *rest value at the pivot*, i.e., the first instance in the sequence. The PPEU value is defined formally in Definition 17.

Definition 11 (pivot-centered prefix extension utility (PPEU)). *Let sp be a sequential pattern and S_q be a q-sequence database. In addition, let $p(sp, s_q)$ be the set of ending positions for sp in s_q, let $u_{sp}(sp, s_q, p)$ be the maximum utility of all instances of sp appearing in s_q that end at p, let $s_q/_{(sp,p_0)}$ be the remaining sequence of s_q with respect to the instance of sp that ends at the pivot position p_0 (i.e., the first position in $p(sp, s_q)$) and let $u_{seq}(s_q/_{(sp,p_0)})$ be the sum utility of this remaining sequence. The pivot-centered prefix extension utility (PPEU) for sp in s_q is computed by:*

$$ppeu(sp, s_q) = \max_{p \in p(sp, s_q)} \{u_{sp}(sp, s_q, p)\} + u_{seq}(s_q/_{(sp,p_0)}). \tag{17}$$

The PPEU for sp is then

$$ppeu(sp) = \sum_{s_q \in S_q} ppeu(sp, s_q). \tag{18}$$

Definition 12 (pivot-centered reduced sequence utility (PRSU)). *Given a sequential pattern sp and a candidate item i, which when concatenated with sp gives a new sequence denoted by $sp + i$, the pivot-centered reduced sequence utility (PRSU) for $sp + i$ in a sequence s_q is computed by:*

$$prsu(sp, i, s_q) = \begin{cases} ppeu(sp, s_q), & \textit{if } sp + i \sqsubseteq_q s_q, \\ 0, & \textit{otherwise.} \end{cases} \tag{19}$$

The PRSU for $sp + i$ is then

$$prsu(sp, i) = \sum_{s_q \in S_q} prsu(sp, i, s_q) \tag{20}$$

The key feature of Equation (17) to note is that the remaining utility value $u_{seq}(s_q/_{(sp,p_0)})$ is now outside of the scope of the maximization. Thus, the rest value does not need to be computed for all ending positions, but rather only for the pivot position each time. As a result, utilities do not need to be updated at all positions when an item is eliminated, but rather *only the rest value at the pivot needs to be updated*. The result is a significant speed up in the time required for updates when an item is eliminated.

With the speed up potential demonstrated, it is also necessary to prove that the new pivot-centered reduced sequence utility, computed based on the PPEU, still indeed provides an upper bound on the utility of any extension of sp with item i.

Theorem 3. *For any sequential pattern sp and item i, $prsu(sp, i)$ is greater than or equal to the utility $u_{seq}(sp')$ of any proper descendant sequential pattern sp' with sp as prefix in the lexicographical tree that is the result of a concatenation with i.*

Proof. Since $rsu(sp, i)$ is known to provide an upper bound (proven in [7]), it is sufficient to show that $prsu(sp, i) \geq rsu(sp, i)$. This is accomplished by showing that $ppeu(sp, s_q) \geq peu(sp, s_q)$ for any sequence s_q containing sp. Since $peu(sp, s_q)$ is the maximum value of $u_{sp}(sp, s_q, p) + u_{seq}(s_q/_{(sp,p)})$ over all ending positions p of sp, then this value is less than or equal to the sum of the maximum $u_{sp}(sp, s_q, p)$ over all ending positions and the maximum $u_{seq}(s_q/_{(sp,p)})$ over all ending positions, which is precisely the value of $ppeu(sp, s_q)$, since $ppeu(sp, s_q)$ is the sum of the maximum values for $u_{sp}(sp, s_q, p)$ and $u_{seq}(s_q/_{(sp,p_0)})$, and $u_{seq}(s_q/_{(sp,p_0)})$ (i.e., the remaining utility value at the pivot) is necessarily the maximum over all ending positions. □

6.2. The CRUSPPivot Algorithm

With this new computational method, there is no need to update utility values at all positions as items are removed. Only one value per sequence, namely the remaining utility $u_{seq}(s_q/_{(sp,p_0)})$ at the pivot, needs to be changed if an item in the remaining sequence $s_q/_{(sp,p_0)}$ is removed. This is implemented as demonstrated by the *CRUSPPivot* algorithm outlined in Figure 7.

In step 1, the PPEU value for sp is computed to determine whether the node corresponding to sp is a leaf. If the PPEU lies below the minimum utility, the search is pruned at that point. In step 2, the *iCandList* is maintained which, as with the *CRUSP* algorithm, may contain items that have been eliminated as promising for itemsets after sp, but could still be i-concatenated with sp itself as part of sp's final itemset. Step 3 stores all promising candidates from *iCandList* in *iList*. In step 4, the items in the MCU-candidate list MCL are checked for PRSU values below the minimum, which are subsequently removed. Each removal (1) initiates an update of the pivot remaining utilities *sp_pivot_ru*, and (2) ensures that all items still in the MCL are assessed again. Since the remaining items are exactly those that are deemed promising for s-concatenation with sp, these are moved to the *sList* in step 5. Steps 6–13 then create the next level of candidate sequential patterns via i-concatenation and s-concatenation, determine the utilities, ending positions and pivot remaining utility values, and recursively call the algorithm. Finally, step 14 outputs the high utility sequential patterns.

```
CRUSPPivot($sp$, sp_util, sp_pos, sp_pivot_ru, iCandList, MCL)

If this node is a leaf, then return
Remove low PRSU items from iCandList
Put promising candidates for $i$-concatenation in iList
Remove low RCU items from MCL
sList = MCL
For each item $i$ in iList:
    $sp'$ = $i$-concatenate($sp, i$)
    Construct sp'_util, sp'_pos, sp'_pivot_ru
    CRUSPPivot($sp'$, sp'_util, sp'_pos, sp'_pivot_ru, iCandList, MCL)
For each item $i$ in sList:
    $sp'$ = $s$-concatenate($sp, i$)
    Construct sp'_util, sp'_pos, sp'_pivot_ru
    CRUSPPivot($sp'$, sp'_util, sp'_pos, sp'_pivot_ru, iCandList, MCL)
if $u(sp) > u_{min}$, then output $sp$
return
```

Figure 7. The *CRUSPPivot* algorithm.

7. Results

7.1. Objectives and Hypotheses

To demonstrate performance of the advances proposed in this paper, a number of different mechanisms for identifying promising concatenation candidates were tested. Two separate experiments were conducted. The first experiment tests the efficacy of maintaining a candidate list during the search. Here, two methods introduced in this paper and implemented by the *CRUSP* algorithm are tested, namely the maximum concatenation utility (MCU) method that maintains a list of items that are candidates for future concatenation, and the reduced concatenation utility (RCU) method that further reduces the upper bound on descendant pattern utilities by capitalizing on items having been removed from the candidate lists. Performance of these two approaches are compared to two state-of-the-art approaches from the literature, namely the sequence-weighted utility (SWU) method for determining upper bounds on candidate utilities as implemented by *uSpan* [2] and the reduced sequence utility (RSU) method implemented by *HUS-Span* [7]. The second experiment tests the speed up produced by the pivot-centered reduced sequence utility (PRSU) method, implemented by the *CRUSPPivot* algorithm, which facilitates fast updates of the utility values, allowing for multiple passes through the candidate item list at each node in the tree.

Tests were conducted on an HP Z820 workstation (Fredericton, NB, Canada) with 32 CPUs, running at 2.20 GHz each, with 28 GB of RAM.

Performance is measured in two respects: (1) size of search space required in terms of the number of candidate patterns generated (i.e., the number of nodes in the lexicographic tree) and (2) the runtime required. The hypotheses are that the approaches implemented by *CRUSP*, namely MCU and RCU, significantly outperform the existing SWU and RSU approaches in terms of both size and speed, with RCU improving over MCU, and that the PRSU approach implemented by *CRUSPPivot* significantly improves upon the *CRUSP* methods in both aspects.

7.2. Experiment #1: CRUSP Performance Validation

For the first experiment, the MCU and RCU methods were validated via comparison with SWU and RSU on four real-life and publicly available datasets. Details on these four datasets are given in Table 1, and are generally described as follows:

- **BMS**: e-commerce website (click-stream data),

- **ds3**: e-commerce website (data on customer transactions),
- **msnbc**: data on website page visits (msnbc.com, 28 September 1999),
- **SIGN**: sign language data.

The BMS and SIGN datasets can be attributed to the SPMF open source data mining library [35]. The ds3 dataset was used by Yin et al. [2] in the *uSpan* and was kindly made available for this paper by the authors. The msnbc dataset was constructed from the msnbc transactional dataset found at the UCI Machine Learning Repository [36], with utilities randomly added using a uniform distribution from $[1, 10]$.

Table 1. Datasets used in testing.

Dataset	Total Util	# Seq	# Items	Avg Length	Type
BMS	2×10^6	59,601	497	2.5	click-stream
ds3	9×10^7	59,476	811	5.9	transactions
msnbc	3×10^7	989,818	17	4.74	page visits
SIGN	6×10^5	730	267	52.0	sign language

Minimum utility thresholds were varied over a number of tests, with each expressed as the portion of the total utility present in the corresponding dataset. While it is desirable for the methods tested to perform well at high minimum utility thresholds, it is a far more significant result if performance is found to drastically improve at lower threshold levels, where existing methods tend to struggle to execute in a reasonable amount of space or time. Figure 8 depicts the performance of each method in terms of the number of candidate high utility sequential patterns considered in the lexicographical tree. Notice that there is only one set of plot points for the RSU and MCU methods. This is because each method actually builds the exact same tree of candidate nodes. In this case, the effect of maintaining a candidate list is expected to be realized in reduced run time, due to the fact that the RSU method is found to be continuously considering items for concatenation and then rejecting them due to their low RSU values. As a result, these candidate extensions are never actually added to the tree. However, run time is indeed spent on evaluating the RSU values for these candidates, whereas the MCU method entirely removes them from consideration for the duration of the search and devotes no further effort towards their evaluation. Experiments validate that space efficiency is dramatically improved by the MCU and RSU methods over SWU, particularly at lower threshold levels. Of greater importance, however, is the significant improvement realized by the RCU method over all other methods for each problem tested, boasting reductions in the number of candidates considered of $99.9\%, 33\%, 42\%$, and 49%, confirming the hypothesis that the RCU offers a substantially tighter upper bound on future extension utilities, and thus offering the opportunity for significantly earlier pruning, resulting in reduced search.

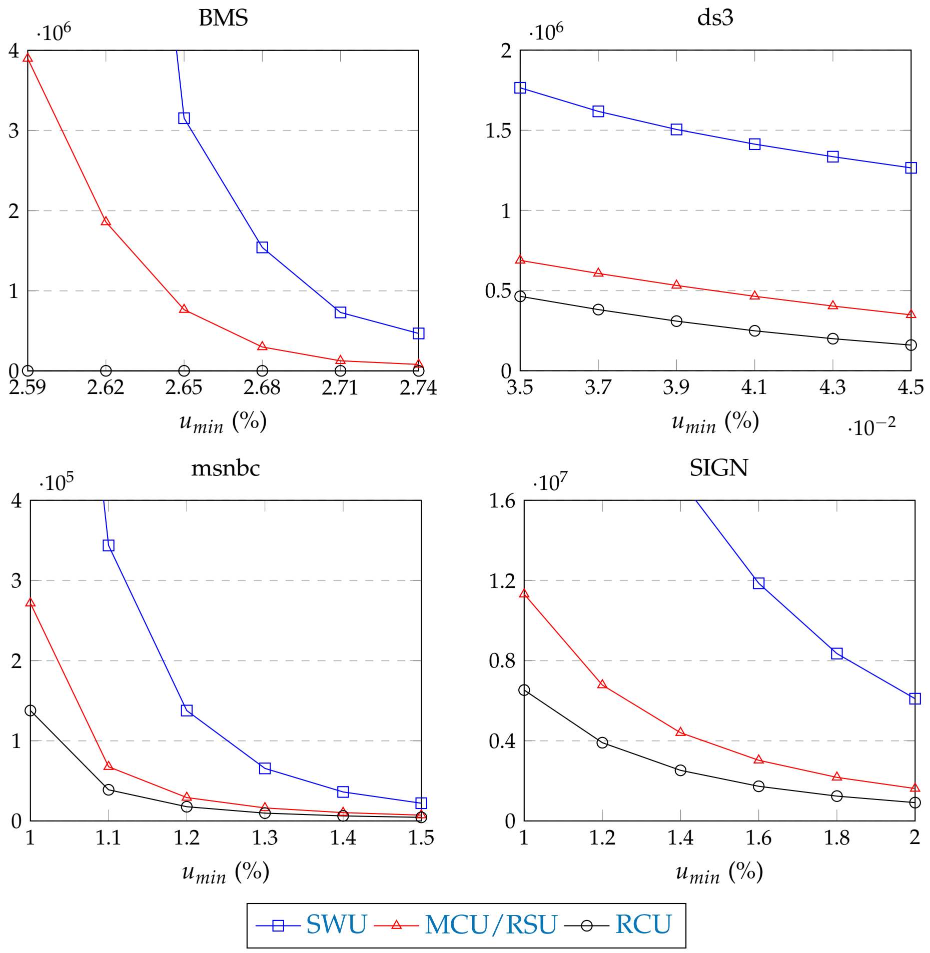

Figure 8. Results of experiment 1: Number of candidate patterns generated by each method to identify the full set of high utility sequential patterns.

Figure 9 validates the performance of the MCU and RCU methods over SWU and RSU in terms of run time (in seconds) required to identify the full set of high utility sequential patterns. Here, the expected improvement in run time performance of the MCU method over the RSU method, as hypothesized above, is confirmed, with run time reductions of 23%, 17%, 22%, and 63% over the four datasets, respectively, at the lowest level of minimum utility tested. The RCU method was found to further reduce run time, with reduction percentages of 94%, 25%, 32%, and 72% over the RSU method. The fact that the RCU method improves upon MCU in all cases (marginally in three cases, and significantly for the BMS dataset) is a very positive result. While it was clear that the RCU should reduce the size of the search space significantly due to the potentially dramatic reduction in the upper bounds computed, it was not clear at the outset that these gains would outweigh the additional computational effort.

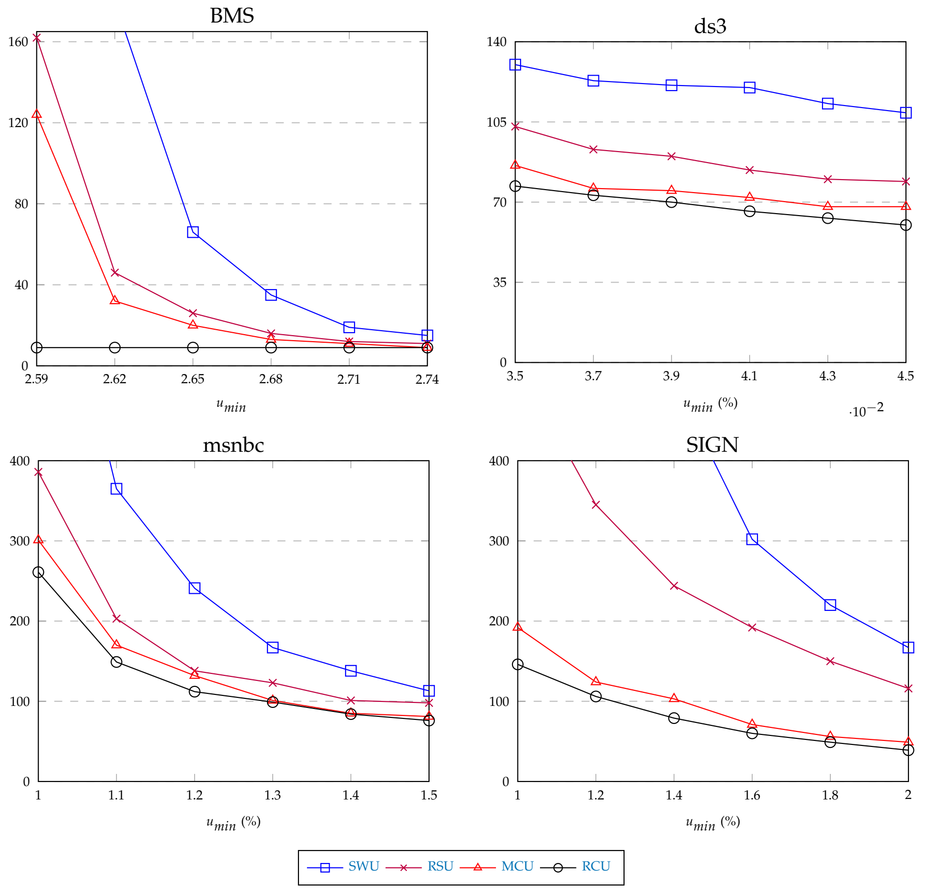

Figure 9. Results of experiment 1: Time (s) taken by each method to identify the full set of high utility sequential patterns.

7.3. Experiment #2: CRUSPPivot Performance Validation

In the second experiment, the PRSU method implemented by *CRUSPPivot* is tested against RCU to assess the improvement in performance in terms of size of search space and run time. Each of the four datasets used in the first experiment were utilized, including two additional datasets described in Table 2. Each of these two new datasets are based on click-stream activity, and were acquired from the SPMF open source data mining library [35]. These two datasets were not included in the first experiment as they were unsolvable by the non-candidate-list-based methods in a reasonable amount of time at reasonable minimum utility levels.

Table 2. Additional datasets used in testing for experiment 2.

Dataset	Total Util	# Seq	# Items	Avg Length	Type
FIFA	7×10^5	20,450	2990	36.24	click-stream
Kosarak10K	1×10^6	10,000	10,094	8.14	click-stream

Figure 10 depicts the performance of each method in terms of the number of candidate high utility sequential patterns considered in the lexicographical tree. The reduction in the number of candidate patterns generated by the PRSU method is highly significant in all cases, and particularly for datasets ds3, SIGN, FIFA, and Kosarak, where reduction by more than a factor of 10 occurs at lower minimum utility levels. Comparing PRSU with the results of existing methods SWU and RSU depicted in Figure 8 shows a massive reduction in the number of candidates generated, for all datasets at all levels. To illustrate just one example, for the BMS dataset at 2.65% minimum utility, the SWU approach required the generation of 3,153,620 candidates while the PPEU approach required just 498, resulting in a reduction of 99.98%.

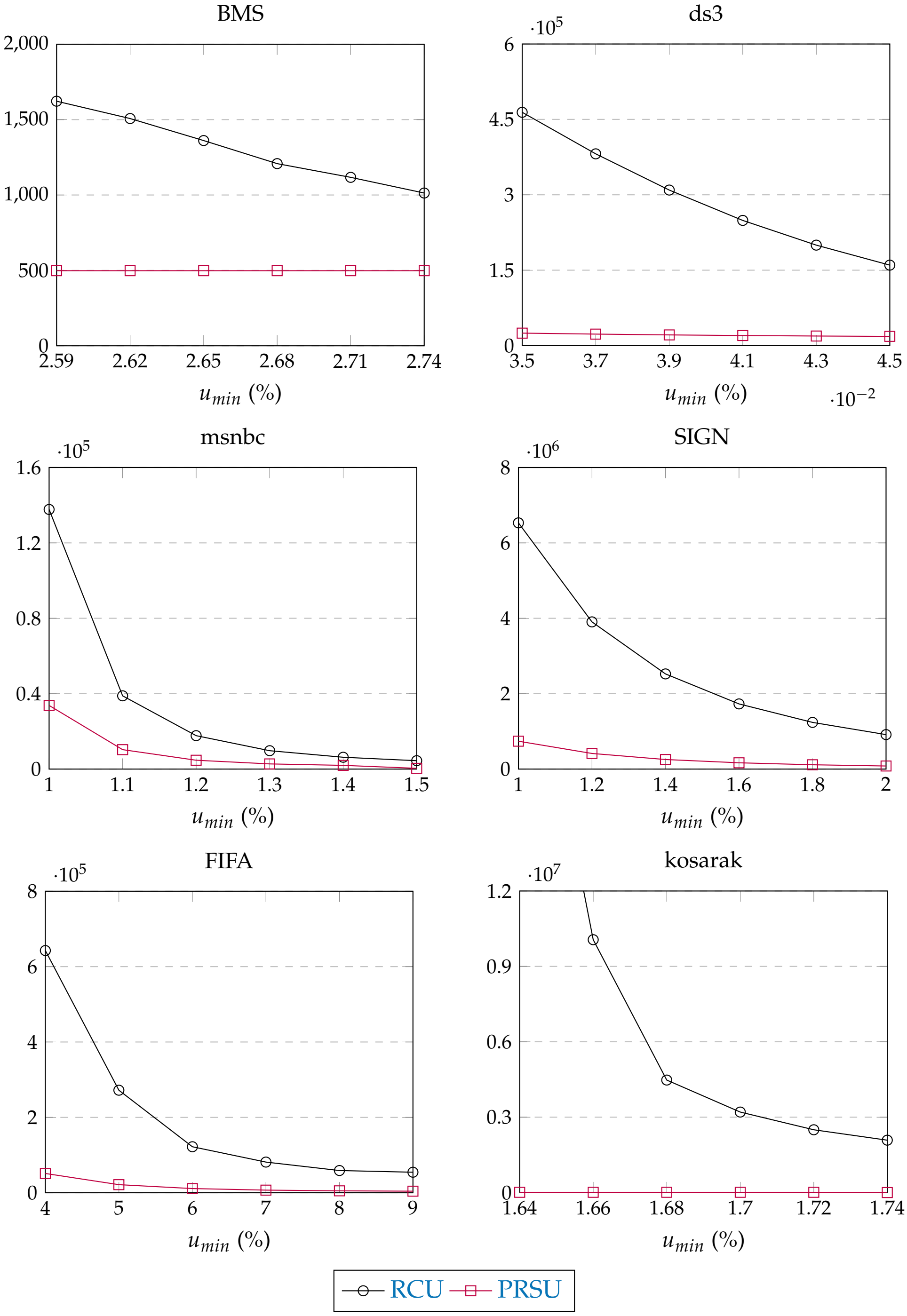

Figure 10. Results of experiment 2: Number of candidate patterns generated by each of the RCU and PRSU methods to identify the full set of high utility sequential patterns.

Figure 11 depicts the performance of each method in terms of the total time (in seconds) taken to identify the full set of high utility sequential patterns. For three of the datasets, BMS, ds3, and msnbc, performance is essentially unchanged, indicating that the decrease in candidate patterns generated and the time taken to process the item candidate lists are essentially an even trade-off in these cases. Tests on the SIGN database show significant improvement for PRSU, with an approximate 13% reduction in run time across the board. FIFA and Kosarak, on the other hand, experience a dramatic reduction in runtime. For FIFA, nearly a 50% reduction in run time is realized at low minimum utility values. For Kosarak, the problem quickly becomes unmanageable for RCU once the minimum falls below 1.66%, where the run time seemingly remains constant for PRSU. It is thus apparent that, depending on the problem, the benefits of PRSU are sometimes neutralized by the additional computational cost; however, at other times, the PRSU method has the potential to yield dramatically improved results.

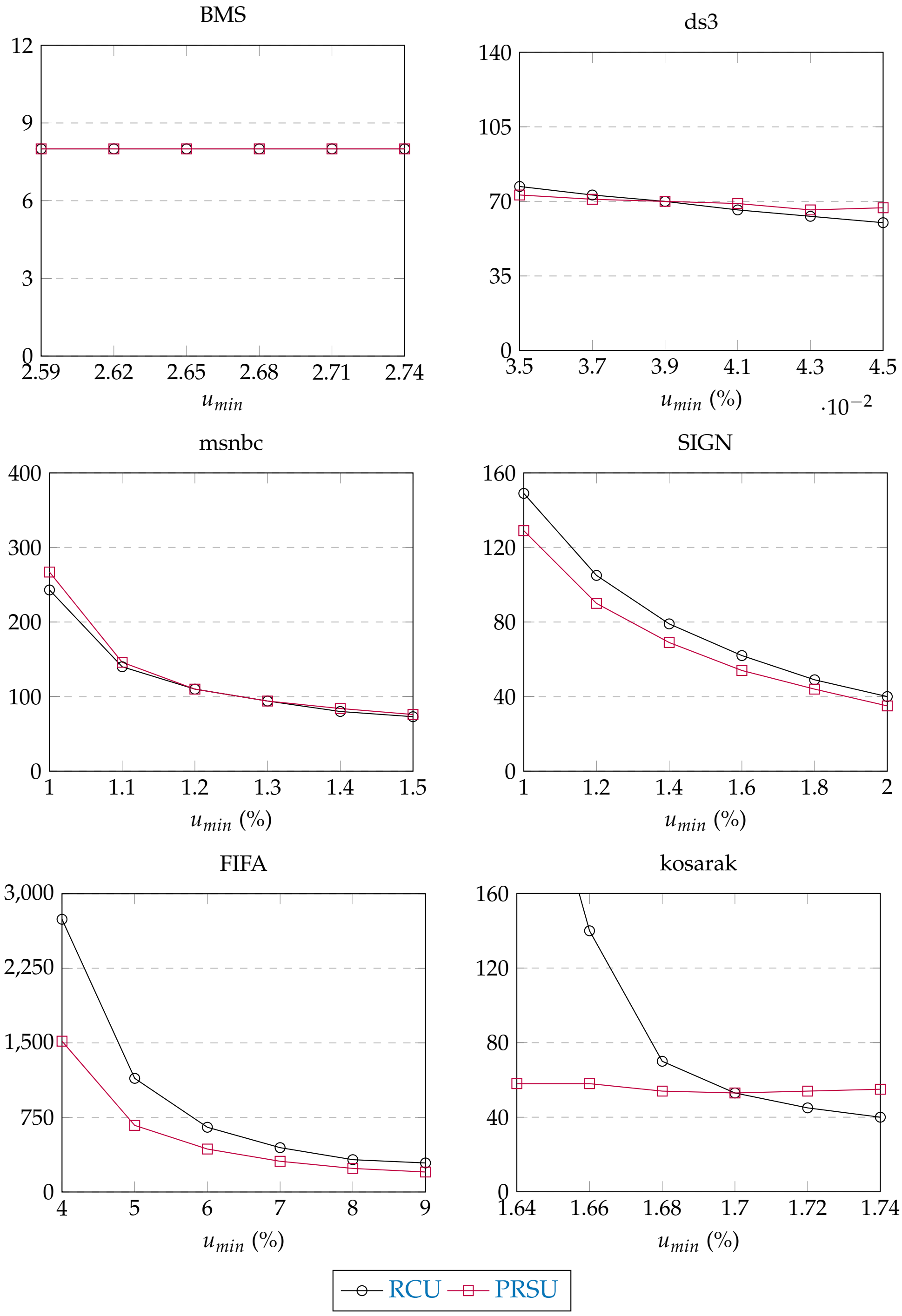

Figure 11. Results of experiment 2: Time (s) taken by each of the RCU and PRSU methods to identify the full set of high utility sequential patterns.

8. Conclusions

The high utility sequential pattern mining (HUSPM) problem seeks to identify frequent patterns that are (1) sequential in nature and (2) hold a significant magnitude of utility in a sequence database, by considering the aspect of item value or importance. The objective is thus to offer a solution to the shortcoming associated with traditional frequency-based sequential pattern mining that all items in an input database are treated with equal importance. As such, the HUSPM problem offers a host of interesting challenges. Chief among these is the non-applicability of search reduction techniques that have been previously developed based on the downward closure property, and which have been relied upon so critically by existing frequency-based pattern mining approaches. This paper offers a deep examination of the challenges of overcoming this obstacle, and details a number of methods from the literature that attempt to address these challenges. Central among these methods is a focus on establishing upper bounds on the utilities of pattern extensions that might be encountered later in the search. The idea here is that, if future extensions from some point are found to have upper bounds on utility that fall short of a prespecified minimum, then search can be cut off at that point.

In this paper, a number of new approaches are proposed that seek to further reduce such upper bounds and produce further dramatic reductions in the number of candidate patterns that need to be considered. The first such approach, referred to as the *maximum concatenation utility* (MCU) method, is proposed, which maintains a list of *candidate concatenation items*. This list specifies the only items that should ever be considered as possible candidates for extension for a particular sequential pattern, or for any future sequential pattern extension that may appear as a descendant in the search tree. The run time savings that result from the ability to disregard items that do not appear in the list is shown empirically to be substantial, exhibiting a reduction in the range of 17% to 63% for four publicly available datasets.

The second approach proposed in the paper, referred to as the *reduced concatenation utility* (RCU) method, capitalizes on the elimination of items from consideration, as deemed by the candidate list, to correspondingly compute an even tighter upper bound, resulting in earlier pruning during the search. Further experiments showed that the RCU approach provides significant improvements over the MCU approach in terms of both size of search space and run time. These new methods were implemented by the *CRUSP* algorithm, which is also outlined in the paper.

Even though the utilization of the RCU method resulted in a significant reduction in the size of the search space, the reduction in run time, while still significant, was found to be relatively limited due to the expensive computational cost required by the consideration of removed utilities in the computation of upper bounds. To alleviate this computational burden, a relaxed upper bound referred to as the pivot-centered prefix extension utility (PPEU) was proposed. This upper bound, while not as tight initially as the previously proposed prefix extension utility from the literature, was instead significantly faster to update. This speed up facilitated the ability to process the candidate list multiple times, consequently leading to the removal of more candidate items and ultimately yielding a dramatically tighter upper bound. The approach was subsequently implemented as part of a new algorithm named *CRUSPPivot*. Tests on a number of datasets showed that this new approach performed no worse than the RCU method on some datasets, while on others the improvement was immeasurable, even solving one problem easily where the RCU method could not complete the search in a reasonable amount of time. When compared against the results of existing methods SWU and RSU depicted in Figure 8, a massive reduction in the number of candidates generated was realized for all datasets at all levels. One particular example was cited that showed that, for the BMS dataset at 2.65% minimum utility, the SWU approach required the generation of 3,153,620 candidates, while the PPEU approach required just 498, resulting in a reduction of 99.98%. This demonstrates that there is significant potential in further pursuing this sort of approach to HUSPM search.

There are a number of outstanding related research questions that may be considered for future work, including (1) whether an algorithm more efficient than *CRUSP* exists that would facilitate the timely usage of the tighter upper bound (RCU) proposed in the paper, as opposed to the slightly more loose PPEU bound, (2) whether bounds tighter than RCU exist, (3) how to improve the efficiency of *CRUSPPivot* to possibly achieve reduction in run time proportionate to the reduction in search space shown in experiments, and (4) how to modify *CRUSP* and *CRUSPPivot* to mine the top-*k* patterns. This list is far from exhaustive, demonstrating the great potential for further research in this area.

Funding: This research received no external funding.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Ahmed, C.F.; Tanbeer, S.K.; Jeong, B.S. A novel approach for mining high-utility sequential patterns in sequence databases. *ETRI J.* **2010**, *32*, 676–686. [CrossRef]
2. Yin, J.; Zheng, Z.; Cao, L. USpan: An efficient algorithm for mining high utility sequential patterns. In Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Beijing, China, 12–16 August 2012; pp. 660–668.
3. Aggarwal, C.C.; Han, J. *Frequent Pattern Mining*; Springer: Berlin/Heidelberg, Germany, 2014.
4. Fournier-Viger, P.; Lin, J.C.W.; Vo, B.; Chi, T.T.; Zhang, J.; Le, H.B. A survey of itemset mining. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2017**, *7*, e1207. [CrossRef]
5. Agrawal, R.; Srikant, R. Fast algorithms for mining association rules. In Proceedings of the 20th Int. Conf. Very Large Data Bases, VLDB, Santiago de Chile, Chile, 12–15 September 1994; Volume 1215, pp. 487–499.
6. Agrawal, R.; Srikant, R. Mining sequential patterns. In Proceedings of the Eleventh International Conference on Data Engineering, Taipei, Taiwan, 6–10 March 1995; pp. 3–14.
7. Wang, J.Z.; Huang, J.L.; Chen, Y.C. On efficiently mining high utility sequential patterns. *Knowl. Inf. Syst.* **2016**, *49*, 597–627. [CrossRef]
8. BUFFETT, S. Candidate List Maintenance in High Utility Sequential Pattern Mining. In Proceedings of the 2018 IEEE International Conference on Big Data (Big Data), IEEE, Seattle, WA, USA, USA, 10–13 December 2018; pp. 644-652.
9. Hájek, P.; Havel, I.; Chytil, M. The GUHA method of automatic hypotheses determination. *Computing* **1966**, *1*, 293–308. [CrossRef]
10. Han, J.; Pei, J.; Yin, Y.; Mao, R. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.* **2004**, *8*, 53–87. [CrossRef]
11. Zaki, M.J. Scalable algorithms for association mining. *IEEE Trans. Knowl. Data Eng.* **2000**, *12*, 372–390. [CrossRef]
12. Srikant, R.; Agrawal, R. *Mining Sequential Patterns: Generalizations and Performance Improvements*; Springer: Berlin/Heidelberg, Germany, 1996.
13. Zaki, M.J. SPADE: An efficient algorithm for mining frequent sequences. *Mach. Learn.* **2001**, *42*, 31–60. [CrossRef]
14. Ayres, J.; Flannick, J.; Gehrke, J.; Yiu, T. Sequential pattern mining using a bitmap representation. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, AB, Canada, 23–26 July 2002; pp. 429–435.
15. Pei, J.; Han, J.; Mortazavi-Asl, B.; Pinto, H.; Chen, Q.; Dayal, U.; Hsu, M.C. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In Proceedings of the 2013 IEEE 29th International Conference on Data Engineering (ICDE), Heidelberg, Germany, 2–6 April 2001; p. 0215.
16. Yao, H.; Hamilton, H.J.; Butz, C.J. A foundational approach to mining itemset utilities from databases. In Proceedings of the 2004 SIAM International Conference on Data Mining, Lake Buena Vista, FL, USA, 22–24 April 2004; pp. 482–486.
17. Liu, Y.; Liao, W.K.; Choudhary, A.N. A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets. In *PAKDD*; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3518, pp. 689–695.
18. Fournier-Viger, P.; Wu, C.W.; Zida, S.; Tseng, V.S. FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In *International Symposium on Methodologies for Intelligent Systems*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 83–92.

19. Fournier-Viger, P.; Wu, C.W.; Tseng, V.S. Novel concise representations of high utility itemsets using generator patterns. In *International Conference on Advanced Data Mining and Applications*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 30–43.
20. Tseng, V.S.; Wu, C.W.; Fournier-Viger, P.; Philip, S.Y. Efficient algorithms for mining top-k high utility itemsets. *IEEE Trans. Knowl. Data Eng.* **2015**, *28*, 54–67. [CrossRef]
21. Lin, J.C.W.; Li, T.; Fournier-Viger, P.; Hong, T.P.; Zhan, J.; Voznak, M. An efficient algorithm to mine high average-utility itemsets. *Adv. Eng. Inform.* **2016**, *30*, 233–243. [CrossRef]
22. Lin, J.C.W.; Zhang, J.; Fournier-Viger, P. High-Utility Sequential Pattern Mining with Multiple Minimum Utility Thresholds. In *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 215–229.
23. Liu, B.; Hsu, W.; Ma, Y. Mining association rules with multiple minimum supports. In Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, 15–18 August 1999; Volume 99, pp. 337–341.
24. Zhang, B.; Lin, J.C.W.; Fournier-Viger, P.; Li, T. Mining of high utility-probability sequential patterns from uncertain databases. *PLoS ONE* **2017**, *12*, e0180931. [CrossRef] [PubMed]
25. Zihayat, M.; Wu, C.W.; An, A.; Tseng, V.S. Mining high utility sequential patterns from evolving data streams. In Proceedings of the ASE BigData & SocialInformatics 2015, Kaohsiung, Taiwan, 7–9 October 2015; p. 52.
26. Xu, T.; Dong, X.; Xu, J.; Dong, X. Mining High Utility Sequential Patterns with Negative Item Values. *Int. J. Pattern Recognit. Artif. Intell.* **2017**, *31*, 1750035. [CrossRef]
27. Zida, S.; Fournier-Viger, P.; Wu, C.W.; Lin, J.C.W.; Tseng, V.S. Efficient mining of high-utility sequential rules. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 157–171.
28. Jiang, Y.; Yin, S.; Kaynak, O. Data-driven monitoring and safety control of industrial cyber-physical systems: Basics and beyond. *IEEE Access* **2018**, *6*, 47374–47384. [CrossRef]
29. Jiang, Y.; Li, K.; Yin, S. Cyber-physical system based factory monitoring and fault diagnosis framework with plant-wide performance optimization. In Proceedings of the 2018 IEEE Industrial Cyber-Physical Systems (ICPS), St. Petersburg, Russia, 15–18 May 2018; pp. 240–245.
30. Jiang, Y.; Yin, S. Recent advances in key-performance-indicator oriented prognosis and diagnosis with a matlab toolbox: Db-kit. *IEEE Trans. Ind. Inform.* **2018**, *15*, 2849–2858. [CrossRef]
31. Mooney, C.H.; Roddick, J.F. Sequential pattern mining–approaches and algorithms. *ACM Comput. Surv. (CSUR)* **2013**, *45*, 19. [CrossRef]
32. Chena, Y.L.; Kuo, M.H.; Wub, S.Y.; Tang, K. Discovering recency, frequency, and monetary (RFM) sequential patterns from customers' purchasing data. *Electron. Commer. Res. Appl.* **2009**, *8*, 241–251. [CrossRef]
33. Li, Z.; Zhang, A.; Li, D.; Wang, L. Discovering novel multistage attack strategies. In *International Conference on Advanced Data Mining and Applications*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 45–56.
34. Buffett, S.; Pagiatakis, C.; Jiang, D. Pattern-Based Behavioural Analysis on Neurosurgical Simulation Data. *Proc. Mach. Learn. Res.* **2018**, *85*, 514–533.
35. Fournier-Viger, P.; Gomariz, A.; Gueniche, T.; Soltani, A.; Wu., C.; Tseng, V.S. SPMF: A Java Open-Source Pattern Mining Library. *J. Mach. Learn. Res. (JMLR)* **2014**, *15*, 3389–3393.
36. Lichman, M. UCI Machine Learning Repository. 2013. Available online: https://archive.ics.uci.edu/ml/index.php (accessed on 11 January 2020).