Sequence Pattern Mining with Variables

James S. Okolica, Gilbert L. Peterson, Robert F. Mills, and Michael R. Grimaila

Abstract—Sequence pattern mining (SPM) seeks to find multiple items that commonly occur together in a specific order. One common assumption is that all of the relevant differences between items are captured through creating distinct items, e.g., if color matters then the same item in two different colors would have two items created, one for each color. In some domains, that is unrealistic. This paper makes two contributions. The first extends SPM algorithms to allow item differentiation through attribute variables for domains with large numbers of items, e.g, by having one item with a variable with a color attribute rather than distinct items for each color. It demonstrates this by incorporating variables into Discontinuous Varied Order Sequence Mining (DVSM) [1]. The second contribution is the creation of Sequence Mining of Temporal Clusters (SMTC), a new SPM that addresses the interleaving issue common to SPM algorithms. Most SPM algorithms address interleaving by using a distance measure to separate co-occurring sequences. SMTC addresses interleaving by clustering all subsets of temporally close items and deferring the sequencing of mined patterns until the entire dataset if examined. Evaluation of the SPM algorithms on a digital forensics media analysis task results in a 96% reduction in terms to review, 100% detection of true positives and no false positives.

Index Terms—Sequence Pattern Mining, Digital Forensics, Temporal Event Abstraction

1 Introduction

The goal of sequence pattern mining (SPM) is to find multiple items that commonly occur together in a specific order. For example, if someone purchases a computer and printer, then they are likely to later purchase replacement ink cartridges and paper. It has been applied in domains as varied as market basket analysis [3], natural languages [4], biology [5], elder care [6], and digital forensics [7].

In general, SPM algorithms either ignore item attributes or use them to define constraints a prior. For instance, there are algorithms [11], [12] that might search for computer and printer purchases where the price of the computer is more than $1,000 and the price of the printer is less than $200. However, the authors have failed to find any SPM algorithms that automatically discovers what the range of constraints should be. When the attribute does matter, either a separate item is created for each potential attribute value or only sequences whose items match the predefined constraint are found.

At times, constraints are not known a priori. In those cases, it may be desirable to first define a range of constraints and then mine frequent sequences that contain all of those constraints. For instance, in the above example, it might be desirable to mine the set of purchases to find that computer prices fall into categories of under $200, between $200 and $600, $600 to $1500 and over $1500. Then, when mining for patterns, instances of all of these constraints can be captured. One issue that emerges is that if there are several attributes and each attribute has several constraints, creating items for each combination of attributes may become prohibitive.

An alternative to creating items for each combination of attributes is keeping a single item and embedding the attribute value within the item. For instance, in the digital forensics domain, it is desirable to have one item for copying files and another for printing files. A person might log into his computer, copy file `123.txt` and then print it. The items are `copy` and `print`, but it's important to associate the file `123.txt` with those items. Then, it becomes possible to create a new item, `CopyPrint`, with file `123.txt` as a value associated with attribute `filename` (whereas copying file `123.txt` and printing file `456.txt` wouldn't fit). For this paper, items that contain variable attribute values are called *terms*.

Sequence mining in the predicate logic domain [13], [14], where there are a small number of predefined items and interleaving is not an issue, has already addressed incorporating variables. This paper presents an SPM for domains with large numbers of items that are discovered during mining and where interleaving exists. This paper makes two contributions. First, it discusses adding variables to SPM algorithms and demonstrates this by extending an existing SPM, *Discontinuous Varied Order Sequence Mining (DVSM)* [1] to DVSM with Variables (DVSM-V). Second, this paper creates a new SPM, Sequence Mining of Temporal Clusters (SMTC) specifically designed to minimize the impact of interleaving.

Application of the SPM to a digital forensics media dataset produces sequences for several user activities (e.g., starting Microsoft Word). When the sequences are checked against the dataset, they are found in all locations where the user activities occurred and no locations where they didn't occur.

2 Related Work

Aggrawal and Srikant [2] describe SPM as " *Given a database of sequences, where each sequence consists of a list of transactions ordered by transaction time and each transaction is a set of items, sequential pattern mining is to discover all sequential patterns*

• *All the authors are with the Department of Electrical and Computer Engineering, Air Force Institute of Technology, WPAFB, OH, 45433.*

Manuscript received April 19, 2005; revised August 26, 2015.

with a user-specified minimum support, where the support of a pattern is the number of data-sequences that contain the pattern."

SPM algorithms fall into three main categories [8]. The first, Apriori-Based Algorithms, extend the original algorithms developed by Agrawal [9]. The second, Pattern Growth Algorithms, addresses Apriori's need for creating a large number of candidates during multiple passes through the dataset by creating a frequent pattern tree and then dividing the tree into a set of projected databases. Finally, the third type of algorithm, Temporal Sequence Mining Algorithms, mine datasets that are episodic in nature [10] where an episode is "a collection of events that occur relatively close to each other in a given partial order." Observe that while all three types of algorithms are appropriate to temporal data, i.e., data that is sequenced by a timestamp, the temporal sequence mining algorithms address a partial ordering, e.g., where an item may occur over a span of time that overlaps a span of time when a second and third item occur.

The general process the Apriori techniques use is to make multiple passes through a database of item. The first pass creates sequences of length 1 and stores the most frequent ones. The second pass extends the frequent length 1 patterns by one to length 2 patterns and stores the most frequent length 2 patterns. The process repeats until no new patterns are added.

Discontinuous Varied Order Sequential Miner (DVSM) [1] is a recent Apriori-baed SPM algorithm applied to the activities of daily living (ADL) domain where activities are the items being mined for sequences. DVSM handles two issues that arise when looking for sequences. The first is that the order that items occur within a sequence changes (e.g., one time when making a sandwich, they get a knife out first and the next time, they get the bread out first). The second is interleaving. Interleaving occurs when two sequences of items occur simultaneously (e.g., setting the table and fixing breakfast). DVSM handles both of these with sequence categorization. DVSM groups similar sequences together using the CLUSEQ algorithm [19] and the Levenshtein edit distance [20] to measure similarity. Frequency is based on the minimum description length [21] which argues that the best description of a data set is the one that maximally compresses it. One assumption that DVSM makes is that item attributes can be safely ignored.

SPM Algorithms that do handle item attributes through the use of variables include SeqLog [13] and MiTempP [14]. Lee and Raedt [13] created SeqLog as a logical language that handles sequences of logical atoms with variables attached to them. Lattner et. al. [14] extend this work with MiTempP which handles temporal data. Both SeqLog (and its associated algorithm, MineSeqLog) and MiTempP are designed to work in domains where there are a small number of items that are defined a priori and where interleaving is not an issue. To the best of our knowledge, no SPM approach handles large numbers of items, item discovery during mining, interleaving, and variables. Observe that there are algorithms that mine for sequences subject to constraints [11], [12]. These algorithms constrain the sequences they find based on item attributes. However, this differs from the current work in that these Constraint Frequent Set Queries search for sequences where the constraints are defined a priori. The current work searches for sequences where the constraints are defined as part of the mining process.

3 Application Domain

A domain that includes large numbers of items, interleaving, and variables is Digital Forensics. Digital forensics is the "discipline that combines elements of law and computer science to collect and analyze data from computer systems, networks, wireless communications, and storage devices in a way that is admissible as evidence in a court of law" [22]. A *digital artifact* is a digital trace found on digital media that is of evidentiary interest. Digital artifacts serve several purposes, including as a record of an activity, and as a record of the digital items on the digital media. A *digital object* is an organized collection of bits that provide content to either a digital device or a user of a digital device. Digital objects include several types of content, all of which must be predefined.

Consider digital objects in the Windows NT Master File Table (MFT). Each file stored on digital media has additional digital artifacts stored with it that include the time the file was created, the last time it was modified, the last time it was accessed, and the last time its meta information was changed. Each of these times corresponds to a separate activity, respectively the file being created, the file being modified, the file being accessed, and the file's meta information being changed. Consider the unique case where all four activities occurred at the same time (e.g., the file was created but nothing has been done with it since). Summarizing those four activities into an "A file was created at ___" idea provides the same amount of high level information to an examiner but with a 75% reduction in the amount of data that needs to be read.

Beyond the MFT, there is a large amount of temporal information available on a digital device. Office automation applications in Microsoft Windows keep track of the most recently used files in the Windows Registry to improve user productivity. Web browsers keep track of the websites visited and the files downloaded in web logs. And the operating system keeps track of important system events like user logins in the Microsoft Windows Event Log and active processes (e.g. volatile memory image).

As with activities of daily living (ADL) domain, the items being mined in the digital forensics domain are activities. While most of the issues with applying sequence pattern mining to digital forensic data are the same as applying it to activities of daily living (e.g., interleaving, items occurring in different orders), there is one specific to computer data. Digital forensics contains a large number of digital objects. For instance, the researcher's computer contains a 465 gigabyte hard drive with over 1.5 million files. It is unrealistic to create separate items for each activity performed on each digital object. Instead, these items must be considered attributes of a much smaller number of common items. Incorporating variables into SPM is one method for handling this.

A common requirement in Digital Forensics is to summarize the activities that occurred on a digital device [23]. Early work on this problem used pre-installed sensors [24] and

then mined the resulting data. While research involving pre-installed sensors continues [25], [26], there is also now research without them [7], [27]. This is important for applying digital forensics to non-corporate crimes, especially those that are non-digital. Both Zeitline [27] and Python Digital Forensic Timeline (PyDFT) [7] extract low level events and allow manual combination of these events into higher-level events. PyDFT does this with rules manually created by subject matter experts. [28] create a new comprehensive system, ParFor (Parallax Forensics) which includes extraction and uses forward chaining inference rules to develop abstractions. This previous work either deals with specific low level sequences of activities or involves significant effort by subject matter experts to create the sequences of activities, i.e., the rules. The current research endeavors to automate the creation of these sequences for more abstract user activities.

4 Definitions

This section defines several terms.

An *Attribute* is a quality or feature of an item.

A *Variable* is a symbol that may assume a given value subject to specified constraints.

A *Dataset* is an ordered sequence of items. For this research, each item will have one or more attributes associated with it.

A *Term* is an item with one or more attributes, each with a variable assigned to it.

An *Abstraction* is the replacement of a sequence of terms with a single term.

A *loop* is an abstraction where all of the terms in a sequence are the same and differ only in their attribute values.

A *single-item sequence* is an abstraction where the attributes of the terms all have the same value.

A *multi-item sequence* is an abstraction with no constraints on the terms.

Conditional Variables are variables that may only take on a subset of values. For instance, a, b, and c may only take on rational values between \$0.01 and \$100.00. Any time these variables are associated with a term the value of the attributes that the variables are assigned to will be within the constrained space. Within the term, the relationship between the variable, attribute and value is called a *cross reference*.

An *Instantiation* of a term is one of its location within the dataset. Associated with the instantiation is the values of the attributes for the term or sequence.

The *Workspace* is an ordered subset of the terms in a dataset that is currently being examined by an SPM.

5 Term and Sequence Generation

In the digital forensics domain, there are large number of sequences where the adjacent terms (i.e., activities) act on the same digital item. More generally, there are a large number of sequences where adjacent terms have attributes with the same variable instantiation. While DVSM-V would capture these single term sequences, using an algorithm specifically designed for single term sequences improves the overall results. Single Term Sequence and Loop Abstraction (STSLA) is a greedy algorithm that creates sequences based on adjacent terms with the same variable instantiation. Since DVSM-V allows for multi-object sequences, the sequences often have two parts of one object and two parts of a second object. For instance, given the terms $T_1(attribute = file_1) + T_2(attribute = file_1) + T_3(attribute = file_1) + T_4(attribute = file_1) + T_1(attribute = file_2) + T_2(attribute = file_2)$ where T_n represents term n, STSLA creates two sequences: $T_1T_2T_3T_4(file_1)$ and $T_1T_2(attribute = file_2)$ while DVSM-V is equally likely to produce $T_1T_2(attribute = file_1)$ and $T_3T_4(attribbute = file_1) + T_1T_2(attribute = file_2)$. Both sets of sequences are correct, but STSLA's is more understandable.

Discontinuous Varied Order Sequence Mining with Variables (DVSM-V), an extension of the DVSM algorithm [1], takes the STSLA output and mines sequences from it. While DVSM is described in [1], it is repeated here for convenience. DVSM proceeds sequentially through the dataset creating all possible sequences of size 1. It then saves those sequences it considers interesting, i.e.,

$$\frac{DL(D)}{DL(a) + DL(D|a)} > C \tag{1}$$

where C is a minimum compression value, D is the dataset, a is a potential sequence and DL is the number of bytes to encode the argument). It then looks at all of the interesting sequences and creates sequences of size 2 based on the items that occurred immediately preceding and following the size-1 sequences. Once it has created all of these size-2 sequences, it again removes all but those it considers interesting. This process repeats either until it does not consider any sequences of the new length interesting or until it reaches a previously defined maximum length. The algorithm concludes by producing sequences for all of the interesting sequences of size 2 or more.

5.1 Attaching Variables to SPM algorithms

Variables provide receptacles for attributes of mined sequences without the need to change the SPM algorithm itself. For instance consider a sequence of computer component purchases where if a customer purchases a memory card and storage, about 15% of the time he later purchases a video card (e.g., $S2$ = `memory + storage + video card`). That may be statistically significant, but does it merit targeted advertising? If a `price` attribute is added to each item, then analysis of the mined sequences may show that when the price of the memory card and storage are both above \$100, the purchase of the video card occurs 80% of the time. Using variables as a receptacle to store attribute values allows better analysis of the mined sequences.

Variables also enable sequences based on the values of the attributes. In the previous example, variables enable sequences like `memory`$(price > \$100)$ + `storage`$(price > \$100)$ + `video card`. In this case, when the SPM mines a sequence, it passes it off to a *variable attachment* algorithm for further processing. The result is that sequences that have the same items (e.g., $S1$ = `memory + storage + video card`) may become sequences with

different terms after variable attachment (e.g., $S2(p, q|p, q > 100)$ = `memory(`p`) + storage(`q`) + video card` and $S2(c, d|c < 50, d < 80)$ = `memory(`c`) + storage(`d`) + video card`). For consistency sake, the variable attachment algorithm pre-defines variables with different constraints (e.g., p and q are constrained to always be greater than 100, while c is always less than 50 and d is always less than 80). Thus, whenever these variables are assigned to attributes, it is clear what the constraints for those attributes are.

While the previous examples have been restricted to a single attribute, attaching variables to SPM generalizes to multiple attributes. For instance, rather than limiting the attributes of the `memory`, `storage` and `video card` items to price, an additional attribute of size can be added. Perhaps it is not the price of memory and storage that is influencing whether video cards are part of the sequence but rather the size of memory and storage. Or, perhaps by adding in a variable with the size attribute to the video card, we can find an additional relationship between the price of memory and storage and the size of the video card. For simplicity, the remainder of this paper only considers items with a single attribute that can have a variable number of values; however, everything discussed generalizes to multiple attributes without loss of generality.

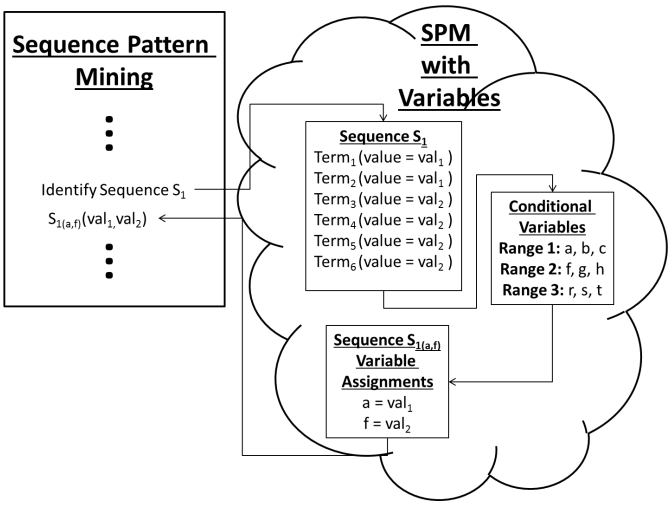

Fig. 1. Attaching Variables to SPM.

As shown in Figure 1, sequence mining of terms takes several steps. First the sequence of terms is identified. Then the number of variables needed is determined by considering the different attribute values in the different terms. Next, variables are assigned to the different values based on the categories the values fall into. Finally, the sequence with the variables assigned is fed back to the SPM for further processing. Observe that in Figure 1 that sequence S_1 is transformed into sequence $S_{1(a,f)}$. In other words, different instances of the same sequence of terms may transform into different sequences (e.g., $S_{1(a,f)}$, $S_{1(a,b)}$, or $S_{1(f)}$ based on the number and types of different values the attributes of those terms have.

Sequence Pattern Mining with Variables can handle mining sequences of terms where the terms are

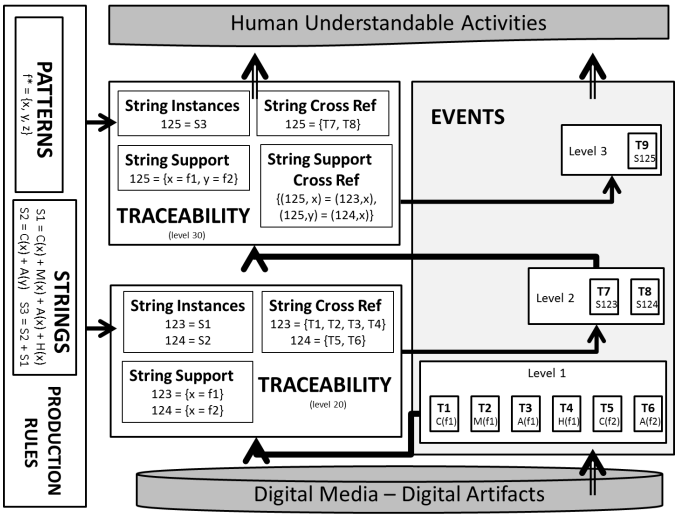

Fig. 2. Traceability.

themselves sequences. Consider a sequence A, B, C. A might be `memory1 ($120)`, B might be `memory2 ($123) + memory3($130) + memory4($127),` and C might be `memory1($120) + storage1($150) + video2($124)`. A is a simple term. B is known as a loop, a single term with a list of values assigned to a single variable , and C is a sequence of terms, each with one or more variables. The process described in Figure 1 becomes more involved. As shown in Algorithm 1, there are now three possibilities when considering a term (i.e., simple terms, loops, and sequences). Loops are handled like simple terms except that there is no attempt to find other terms in the sequence with the same list of values. Finally, there are sequences, a term that is itself a sequence of terms that already has a list of variable assignments. As shown in Figure 2, these variable assignments need to be combined with the variable assignments for the other items in the sequence. When the same value has already been assigned to a different variable, a cross reference between the new variable and the old variable needs to be made for traceability. When the variable assignments for a sequence is completed, it has the same structure whether it contains sequences or not. The only difference is the cross reference used for traceability.

6 Abstraction with Variables

Sequence pattern mining can be done iteratively. In this case, after each iteration of SPM, the "interesting" mined sequences in the dataset are replaced with a single term, either a sequence or loop, that represents the sequence. The result of iterative SPM, also known as abstraction, is a smaller sequence of terms that more compactly describes the dataset. Abstraction is done in two steps. First, the entire dataset is processed using for sequences and then a second time for loops (i.e., sequences of terms where the objects are all the same but the variable instantiations vary).

One initial complication that arises when performing abstraction on sequences with variables concerns the distance measurement. A common algorithm for measuring

Algorithm 1 SPM - Variable Attachment

Input
  S_i - a sequence of terms
  $ConditionalVariables$ - list of conditions and a list of variables for each condition
Output
  S_o - a sequence of terms
  X_v - cross reference of variables
  $Used_v$ - list of used variables
Variables
  t_{new} - a copy of S_i's current term
  v - variable being assigned to current term
  $attList$ - list of attribute values associated with term
  $wattribute$ - attribute value being examined for term
  $Xref$ - if term is a sequence, cross reference for the instance of that sequence in the dataset
  v_old - variable/value for an attribute in $Xref_i$
  v_new - variable/value for an attribute for S_o
$Used_v, X_v \leftarrow \emptyset$
Algorithm
  for $i \leftarrow 1, S_i.getLength()$ **do**
    $t_{new} \leftarrow S_i.getTerm(i)$
    $t_{new}.xref, t_{new}.variable \leftarrow NULL$
    if $t_{new}.isLoop()$ **then**
      $v = ConditionalVariables.getVariable(NULL)$
      $t_{new}.variable \leftarrow v$
    else if $t_{new}.isSingleValue()$ **then**
      $v = Used_v.getVariable(t.getAttributes()$
      if $v == NULL$ **then**
        $v = ConditionalVariables.$
          $getVariable(t.getAttributes())$
      end if
      $Used_v \leftarrow Used_v \bigcup \{v, t.getAttributes()\}$
      $t_{new}.variable \leftarrow v$
    else
      $attList \leftarrow t.getAttributes()$
      while $wattribute = attList.pop()$ **do**
        $v = Used_v.getVariable(t.getAttributes())$
        if $v == NULL$ **then**
          $v = ConditionalVariables.$
            $getVariable(t.getAttributes())$
        end if
        $Used_v \leftarrow Used_v \bigcup \{v, t.getAttributes()$
      end while
      $Xref \leftarrow S_i.getTerm(i).getXref()$
      for $j \leftarrow 1, Xref.getLength()$ **do**
        $witem \leftarrow Xref.getAttributes(j)$
        $v_{old} \leftarrow Xref.getVariable(wattribute)$
        $v_{new} \leftarrow Used_v.getVariable(wattribute)$
        if $v_{new} == NULL$ **then**
          $v_{new} = ConditionalVariables.$
            $getVariable(wattribute)$
          $Used_v \leftarrow Used_v \bigcup \{v_{new}, wattribute\}$
        end if
        $X_v \leftarrow X_v \bigcup \{v_{old}, v_{new}\}$
      end for
      $t_{new}.xref \leftarrow X_v$
    end if
    $S_o \leftarrow S_o \bigcup t_{new}$
  end for

the distance between two sequences is the Levenshtein edit distance [20]. This distance is a measure of how many changes must occur before one sequence is identical to the the other. Changes include inserting a term, deleting a term and replacing one term with another. When variables are added, the concept of replacing one term with another becomes more complex. If two terms are identical except for the constraints placed on the attributes (e.g., both terms are computer memory but one is memory with a price under $100 and the other is memory with a price over $100), are they "closer" than two terms that are different even without considering their attributes (e.g., memory versus storage). The general Levenshtein edit distance algorithm allows for a different costs for inserting, deleting, and swapping terms. When using it with sequences with variables, it needs to further allow for different costs if the only difference in the terms being swapped are variable constraints.

6.1 Sequence Mining of Temporal Clusters (SMTC)

While there are several SPM algorithms that handle interleaving, most handle it as an afterthought. Interleaving occurs when two or more sequences of terms occur simultaneously. For instance, if sequence 1 is $ABBCD$ and sequence two is $CBFFA$ then $ACBBBCFFDA$ is an example of those two sequences interleaved. If $ACBBBCFFDA$ is the only example of sequences 1 and 2, then it's impossile to mine them. However, if they occur frequently elsewhere as individual sequences where there isn't interleaving, it becomes possible to find them in $ACBBBCFFDA$.

A common method for decoupling interleaved sequences is to compare the sequence being examined with previously extracted sequences. If the sequence in question is "close enough" to a previously extracted sequence, then it is considered another example of the sequence. There are several methods for determining "closeness". One measurement of distance that is often used is the Levenshtein edit distance [20]. This determines the minimum number of simple edit operations necessary to transform one sequence to another. In the above example, transform $ACBBBCFFDA$ into $ABBCD$ or $CBFFA$ would require five delete operations. Thus, the distance between the sequences is five. There are several issues with this. The first is determining what a reasonable threshold for closeness is. In the above example, five operations are required to transform $ACBBBCFFDA$ into $ABBCD$ and $CBFFA$ (four if we can ignore the final or first characters respectively). Using that threshold would mean that sequence 1 and 2 are also the same sequence (since they can be transformed into each other with only 4 swap operations). One way around this issue is to put different costs on different types of operations (e.g., the delete operation costs 1 distance while the insert and swap operations cost 2). A second issue in separating interleaved sequences is that the same candidate sequence needs to be compared to every previously extracted sequence. For instance, it is not sufficient to simple transform $ACBBBCFFDA$ into $ABBCD$; it needs to be transformed into $ABBCD$ and $CBFFA$. Thus, once one sequence is extracted, the terms remaining in the candidate (i.e., the noise) needs to be examined again to see if there are more sequences hidden in it.

An alternative method for resolving interleaving is to focus first on what terms occur together. By first clustering terms that occur close together, we avoid several complications. First, there is no need to worry about the order of terms varying (e.g., one time when making a sandwich, they get a knife out first and the next time, they get the bread out first). Terms are first considered without order, only as temporally close sets. Second, since temporally close terms are first placed in clusters, there is no concept of interleaving. Each time two or more temporally close terms occur, all possible subsets of those terms are placed in clusters. For instance, if v_1, v_2 and v_3 occur temporally close together, the subsets $\{v_1\}, \{v_2\}, \{v_3\}, \{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}$, and $\{v_1, v_2\, v_2, v_3\}$ are all created. Then the existing set of clusters is checked to see if these subsets already exist. If they do, the new instance of the subset is added to the cluster. In this way, once the entire dataset has been examined, the resulting clusters are free from interleaving. Any cluster that exceeds a threshold (discussed below) is considered interesting. Only after the clusters have been created and the uninteresting ones removed is sequencing considered (discussed below).

Sequence Mining of Temporal Clusters (SMTC) is based on this second option for resolving interleaving. There are two implementation decisions to be considered when implementing the SMTC class of sequence pattern mining algorithms. The first is how are interesting clusters determined. For this implementation, the score used is

$$occurrences(C) * |C|^{variableCount(C)*\epsilon} \quad (2)$$

where $occurrences(C)$ is the number of times this cluster of terms occurs in the dataset, $variableCount(C)$ is the number of distinct variables in C, and ϵ is a user specified weighting. Observe that using a different formula, e.g., Equation 1, doesn't change the SMTC class of algorithms at all. The second implementation decision is how to determine the optimal sequence for a given cluster. In this implementation, the sequence that occurred most frequently in the dataset (i.e., the mode) is considered the representative sequence for the cluster. However, other choices can be made (e.g, using the median sequence) without changing the SMTC class of algorithms. Algorithm 2 shows the SMTC class of algorithsm. Parameters associated with SMTC include the maximum number of terms in the workspace (i.e., how big of a power set can the machine handle), how much weight to give to fewer different variables in a sequence, the maximum time lag for terms to be considered temporally close, and the threshold score for SMTC to keep a cluster of terms.

7 The Experiment

The dataset used in this research is constructed from a collection of virtual machine snapshots created in a controlled environment with Microsoft Windows 10 and Office 2013 installed to create the terms and sequences for Microsoft Word, Microsoft Excel, Microsoft PowerPoint, Google Chrome, and Mozilla Firefox. Table 1 is a summary of the five use cases and the five to eight scenarios for each. When performing each scenario, there are several guidelines. First, after starting the virtual machine, the examiner

Algorithm 2 Sequence Mining of Temporal Clusters (SMTC)

Input
- D - the dataset - a sequence of terms
- M_S - maximum terms in a sequence
- M_τ - maximum workspace time lag
- ϵ - the weight of more variables in a sequence
- $threshold$ - the minimum score necessary for a sequence to be interesting

Output
- s_i - a generated sequence and all its instantiations in D of it
- S - the set of generated sequences, $\cup s_i$
  - $s_n \rightarrow t_1 + t_2 + ... + t_n$ where
  - $t_1, t_2, ..., t_n \in D$.getTerms()
  - $\cup D$.getSequences()

Variables
- t - the current term
- W - the workspace - the subset of D being examined
- w_i - the i^{th} entry in the workspace
- P - the power set generated by W where each $p \in P$ has all its instantiations attached
- C - the set of clusters. For $c \in C$, c is an unordered set, but the instantiations attached to c are ordered as in D
- $distance(x, y)$ - the temporal distance between x and y

Algorithm

```
S, W, C ← ∅
D.gotoBeginning()
while (t ← D.nextTerm()) ≠ NULL do
    if (|W| > M_S or distance(w_1, t) > M_τ) then
        P ← powerset(W)
        for p ∈ P do
            c ← t
            if c ∈ C then
                c.occurrences++
                c.addInstantiations(
                        p.getInstantiations())
            else
                C = C ∪ c
                c.occurrences ← 1
                c.addInstantiations(
                        p.getInstantiations())
            end if
        end for
        Remove terms from workspace till
                new term fits in workspace
    end if
end while
S ← KeepInteresting(C, ε, threshold)
```

TABLE 1
Use Cases.

Use Case	Scenarios for Starting and Using Applications in Windows 10
1a	User creates a MS Word document
1b	User opens MS Word from the start menu, opens an existing file from within MS Word, edits and saves it, and exits
1c	User open File Explorer, clicks on an existing file, edits and saves, it, and exits MS Word
1d	User opens Command Line, types the file name, edits and saves it, and exits MS Word
1e	User opens MS Internet Explorer and downloads and opens a document and saves it
1g	User opens Mozilla Firefox and downloads and opens a document and saves it
1h	User opens two files and copy/pastes between them
2	Same scenarios for MS Excel
3	Same scenarios for MS PowerPoint
4a	User starts Google Chrome by selecting it from the MS menu and visits a website
4b	User starts Google Chrome by selecting it from the MS menu, visits several websites and downloads/saves content
4c	User starts Google Chrome by selecting it from the MS menu, visits Facebook, logs in, and starts instant messaging
4d	User starts Google Chrome by selecting it from the MS menu, visits Twitter, logs in, and makes a tweet
4e	User starts Google Chrome by selecting it from the MS menu, visits Instagram, logs in, and views pictures
5	Same scenario for Mozilla Firefox

TABLE 2
Extracted Data Files

Data Type	**File System Location**
Event Logs	/windows/system32/winevt/logs
Registry Files	/windows/system32/config
User Data	/%USERPROFILE%/appdata/NTUSER.DAT /%USERPROFILE%/appdata/local/microsoft/windows/UsrClass.DAT /%USERPROFILE%/appdata/local/microsoft/windows/history/history.ie5 /%USERPROFILE%/appdata/local/google/chrome/user data/default/History /%USERPROFILE%/appdata/roaming/mozilla/firefox/profiles/xxxx/places.sqlite

waits 30 minutes before performing the first step of the scenario. The second is that there is at least a two minute delay between each step in the scenario. Both of these guidelines are to minimize the chances of the operating system running background processes that will complicate generating the grammar. Once the images have been created, the files shown in Table 2 are extracted and used to create the dataset. Each scenario takes approximately 90 minutes to create and extract the data form and another 90 minutes to load into a database for use. A separate database is used for each use case and all scenarios for a single use case are in the same database.

First, STSLA is applied to the dataset to create an abstracted dataset. Next, DVSM-V and SMTC are applied to further abstract the dataset. In each case, there is a maximum size sequence that is considered. During testing of STSLA, it was shown that no single term sequence was ever larger than eight terms and thus a maximum size of eight was used. For DVSM-V and SMTC, the constraints of the machine used to abstract the data limits us to a maximum sequence size of thirty and six respectively. The six term limitation for SMTC has resulted in it being run twice for a total maximum sequence size of thirty-six. As shown in Figure 2, the terms and sequences are applied to items at a specific level of abstraction to create items at a higher abstraction level as well as traceability back to the original items, terms and sequences. The result of applying these algorithms to the dataset is the creation of 4,709 terms and 7,710 sequences in the DVSM-V dataset and 61,778 terms and ‘8,759 sequences in the SMTC dataset. In both cases, there is a 96% reduction in the number of terms.

Once the dataset has been sufficiently abstracted, the final step is creating a set of production rules that describe starting the appropriate application (e.g., MS Word, MS PowerPoint, MS Excel, Chrome, Firefox). This occurs by using the detailed times recorded during creating the use cases of when the application begins loading and when it finishes loading. The activities bookended by these times are all considered to be part of starting the application. As an example, Table 3 shows the terms and sequences involved in starting Microsoft Word and Table 4 shows how they combine for the different scenarios. Note that there are some terms that seem to perform the Create, Modify, Access, Change on the same file twice. In reality, this is part of establishing exclusive access to the file (and is actually the file and a lock file with a similar name). Table 5 shows the production rules generated after the sequences in Table 4 are consolidated. Table 6 shows an example of the production rules used to generate the sequence for scenario UC1A. There are three cases where a specific scenario for a specific use case produced so few (three - five) terms and sequences that it was excluded. In all cases, when the generated production rules are applied to the constructed datasets, the activities used to generates the rules are found and no other sequences of activities are found.

8 Conclusions and Future Work

The Variable Attachment algorithm discussed is the first attempt at attaching items to sequence pattern mining (SPM) algorithms in domains with large numbers of items discovered during SPM where interleaving exists. By attaching it to DVSM [1], we show that it can be used with existing SPM algorithms. Sequence Mining of Temporal Clusters (SMTC) is a new perspective on SPM that focuses on resolving interleaving at the expense of either pattern size or number of passes through the dataset. The utility of variable attachment has been shown by applying DVSM-V and SMTC to the a constructed dataset in the digital forensics domain. Applying these algorithms to a constructed dataset has resulted in a 96% reduction in the number of terms and the creation of production rules for recognizing user activities.

The current results only show the algorithms applied to the use cases for starting the applications. Subsequent tests should be run using the existing data for performing simple tasks in the applications. Once all of these user activities are modeled as well as some system processes (e.g., automated software updates), a technical narrative of a “day in the life” can be presented. Further use cases would provide

TABLE 3
Representative Terms and Sequences

Term/Sequence	Description
T13	Modify File ^.*$
T112	System Modify Registry Key ^.*$
T150	User Modify Registry Key ^.*$
T205	Change File ^.*$
T145	System Modify Registry Key ^.*/Windows/.*$
T230	User Modify Registry Key ^.*/Windows/.*$
T310	User Access File ^.*$
S347	Modify and Change File ^.*/Windows/.*/Windows/.*$
S667	Modify and Change File ^.*/Windows/.*$
S676	Modify, Access and Change File ^.*/Windows/.*$
S677	Create, Modify, Access, Change, Create, Modify, Access, and Change File ^.*/Windows/.*$
S706	Modify, Access and Change File ^.*$
S757	Create, Modify, Access, Change, Create, Modify, Access, and Change File ^.*/Users/user/AppData/.*/Microsoft/.*$
S754	Create, Modify, Access, Change, Create, Modify, Access, and Change File ^.*/Users/user/AppData/Local/.*$
S759	Modify and Change File ^.*/Users/user/AppData/.*/Microsoft/.*$
S766	Modify, Access, and Change File ^.*/Users/user/AppData/.*/Microsoft/.*$
S768	Create ^.*/Users/user/AppData/.*/Microsoft/.*$
S777	Modify, Access, Change, Modify, Access and Change File ^.*/Users/user/AppData/.*/Microsoft/.*$
S940	Modify, Access, and Change File ^.*/Users/user/AppData/Local/.*$
S1080	Create, Modify, Access, Change, Create, and Access File ^.*/Users/user/AppData/.*/Microsoft/.*$

TABLE 4
Example Sequences for Starting Microsoft Word

Scenario	Sequence
UC1A	667 230 112 150 1080 766 757 667 112 667 667 676 677 667 230 145 667
UC1B	667 667 667 310 230 667 230 112 150 145 667 112 667 667 112 676 677 667 145
UC1C	230 667 112 667 667 667 150 230 112 145 667 667 676 677 667 667 112
UC1D	13 230 706 768 205 150 230 667 150 230 667 112 145 667 676 677 667 667 112 766 777 759 766 759 777 230 150 112 766 757 759 310 230 230 759 759 150 230 150 766 150 112 145 940 150
UC1E	768 205 230 766 757 230 667 766 777 759 150 667 112 145 667 676 677 667 754 667 150 766 759 777 150 230
UC1F	150 766 759 766 676 677 1070 150 836 214 150 145 112 428 735 940 228 667 735 1196 735 214 150 667 347
UC1G	667 230 112 150 112 667 150 145 676 677 145 667 706 145 1070 112 667 667
UC1H	940 377 228 667 112 150 759 150 112 667 643 676 1050 667 150 667 1063 112 40 766 315 490 202 112 667 150 230 145 766 757 667 230 667 490 48 759 667 13 74 8 40 667 342 677 757 766 230 667 112 677 667 230 754 667 230 777 667 112 112 676 686 112 1047 112 676 667 202 667 706 145 145 766 777 759 694 150 667 759 145 706 150 777 150 759 230 112 766 759 777 230 112 754 766 940 766 150 377 667

TABLE 5
Representative Production Rules

ID	Rule
S3	T112 + T150 + T1080 + T766 + T757
S12	T676 + T677
S21	T112 + T667
S28	T230 + T145
S84	S21 + T667 + S12
S252	T667 + T230 + S3
S284	S252 + T667 + S84
S285	S284 + T667 + S28
S1010 (b)	S285 + S1012
S1011	T0 + S1010 + T0
S1012 (f)	T667

TABLE 6
Example of Production Rules Generating a Sequence

S1010 (b)
S285 + S1012 (f)
S284 + T667 + S28 + S1012 (f)
S252 + T667 + S84 + T667 + S28 + S1012 (f)
T667 + T230 + S3 + T667 + S84 + T667 + S28 + S1012 (f)
T667 + T230 + T112 + T150 + T1080 + T766 + T757 + T667 + S84 + T667 + S28 + S1012 (f)
T667 + T230 + T112 + T150 + T1080 + T766 + T757 + T667 + S21 + T667 + S12 + T667 + S28 + S1012 (f)
T667 + T230 + T112 + T150 + T1080 + T766 + T757 + T667 + T112 + T667 + T667 + S12 + T667 + S28 + S1012 (f)
T667 + T230 + T112 + T150 + T1080 + T766 + T757 + T667 + T112 + T667 + T667 + T676 + T677 + T667 + S28 + S1012 (f)
T667 + T230 + T112 + T150 + T1080 + T766 + T757 + T667 + T112 + T667 + T667 + T676 + T677 + T667 + T230 + T145 + S1012 (f)
T667 + T230 + T112 + T150 + T1080 + T766 + T757 + T667 + T112 + T667 + T667 + T676 + T677 + T667 + T230 + T145 + T667

additional rules for better results when applied to unseen datasets.

BIBLIOGRAPHY

REFERENCES

[1] P. Rashidi, D. J. Cook, L. B. Holder, and M. Schmitter-Edgecombe, "Discovering activities to recognize and track in a smart environment," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 4, pp. 527–539, 2011.

[2] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proceedings of the International Conference on Data Engineering*. IEEE, 1995, pp. 3–14.

[3] S. Brin, R. Motwani, J. Ullman, and S. Tsur, "Dynamic itemset counting and implications rules for market basket data," in *Proceedings of the ACM Special Interest Group on Management of Data Conference on the Management of Data*, vol. 26. ACM, 1997, pp. 255–264.

[4] N. Jindal and B. Liu, "Identifying comparative sentences in text documents," in *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2006, pp. 244–251.

[5] G. Bejerano and G. Yona, "Modeling protein families using probabilistic suffix trees," in *Proceedings of the Third Annual International Conference on Computational Molecular Biology*. ACM, 1999, pp. 15–24.

[6] D. J. Cook, N. C. Krishnan, and P. Rashidi, "Activity discovery and activity recognition: a new partnership," *IEEE Transactions on Cybernetics*, vol. 43, no. 3, pp. 820–828, 2013.

[7] C. Hargreaves and J. Patterson, "An automated timeline reconstruction approach for digital forensic investigations," *Digital Investigations*, vol. 9, pp. S69–S79, 2012.

[8] C. H. Mooney and J. F. Roddick, "Sequential pattern mining–approaches and algorithms," *ACM Computing Surveys (CSUR)*, vol. 45, no. 2, p. 19, 2013.

[9] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, pp. 487–499.

[10] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovering frequent episodes in sequences extended abstract," in *1st Conference on Knowledge Discovery and Data Mining*, 1995.

[11] L. V. Lakshmanan, R. Ng, J. Han, and A. Pang, "Optimization of constrained frequent set queries with 2-variable constraints," in *ACM SIGMOD Record*, vol. 28, no. 2. ACM, 1999, pp. 157–168.

[12] C. Bucilă, J. Gehrke, D. Kifer, and W. White, "Dualminer: A dual-

pruning algorithm for itemsets with constraints," *Data Mining and Knowledge Discovery*, vol. 7, no. 3, pp. 241–272, 2003.
[13] S. Dan Lee and L. De Raedt, *Constraint Based Mining of First Order Sequences in SeqLog*. Springer Berlin Heidelberg, 2004, pp. 154–173.
[14] A. D. Lattner and O. Herzog, "Constraining the search space in temporal pattern mining," in *LWA*, 2006, pp. 314–321.
[15] M. J. Zaki, "Spade: An efficient algorithm for mining frequent sequences," *Machine learning*, vol. 42, no. 1-2, pp. 31–60, 2001.
[16] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo, "Fast discovery of association rules," *Advances in Knowledge Discovery and Data Mining*, vol. 12, no. 1, pp. 307–328, 1996.
[17] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu, "Freespan: frequent pattern-projected sequential pattern mining," in *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2000, pp. 355–359.
[18] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth," in *Proceedings of the 2001 International Conference on Data Engineering*. IEEE, 2001, pp. 215–224.
[19] J. Yang and W. Wang, "Towards automatic clustering of protein sequences," in *Proceedings of the IEEE Computer Society Bioinformatics Conference*. IEEE, 2002, pp. 175–186.
[20] V. Levenshtein, "Binary codes capable of correct deletions, insertions and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.
[21] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, pp. 465–471, 1978.
[22] A. Marcella and D. Menendez, *Cyber Forensics: A Field Manual for Collecting, Examining, and Preserving Evidency of Computer Crimes, Second Edition*. Auerbach Publications, 2007.
[23] S. L. Garfinkel, "Digital forensics research: the next 10 years," *Digital Investigation*, vol. 7, pp. S64–S73, 2010.
[24] S. T. King and P. M. Chen, "Backtracking intrusions," in *Proceedings of the Nineteenth ACM Symposium on Operating System Principles*. ACM, 2003, pp. 223–236.
[25] M. N. A. Khan, "Performance analysis of bayesian networks and neural networks in classification of file system activities," *Computers & Security*, vol. 31, no. 4, pp. 391–401, 2012.
[26] Y.-C. Liao and H. Langweng, "Resource-based event reconstruction of digital crime scenes," in *IEEE Joint Intelligence and Security Informatics Conference*. IEEE, 2014, pp. 129–136.
[27] F. P. Buchholz and C. Falk, "Design and implementation of zeitline: a forensic timeline editor." in *Proceedings of the 2005 Digital Forensics Research Workshop*, 2005, pp. 1–7.
[28] B. Turnbull and S. Randhawa, "Automated event and social network extraction from digital evidence sources with ontological mapping," *Digital Investigation*, vol. 13, pp. 94–106, 2015.