# Sequence Pattern Mining with Variables

James S. Okolica, Gilbert L. Peterson, Robert F. Mills, *Senior Member, IEEE*, and Michael R. Grimaila, *Senior Member, IEEE*

**Abstract**—Sequence pattern mining (SPM) seeks to find multiple items that commonly occur together in a specific order. One common assumption is that the relevant differences between items are captured through creating distinct items. In some domains, this leads to an exponential increase in the number of items. This paper presents a new SPM, Sequence Mining of Temporal Clusters (SMTC), that allows item differentiation through attribute variables for domains with large numbers of items. It also provides a new technique for addressing interleaving, a phenomena that occurs when two sequences occur simultaneously resulting in their items alternating. By first clustering items temporally and only focusing on sequences after the temporal clusters are established, it sidesteps the traditional interleaving issues. SMTC is evaluated on a digital forensics dataset, a domain with a large number of items and frequent interleaving. Its results are compared with Discontinuous Varied Order Sequence Mining (DVSM) with variables added (DVSM-V). By adding variables, both algorithms reduce the data by 96 percent, and identify 100 percent of the events while keeping the false positive rate below 0.03 percent. SMTC mines the data in 20 percent of the time it takes DVSM-V and provides a lower false positive rate even at higher similarity thresholds.

**Index Terms**—Sequence pattern mining, variables, item attributes, interleaving, 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 [1], natural languages [2], biology [3], elder care [4], and digital forensics [5].

In general, SPM algorithms either ignore item attributes or use them to define constraints a priori. For instance, there are algorithms [6], [7] 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. In these cases, the SPM items are the computer and printer and the attributes are the prices. In general, an *attribute* is a quality or feature of an item that is of interest. 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 [8], [9].

At times, constraints are not known a priori. In these 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 and constraints may become prohibitive.

An alternative to creating items for each combination of attributes and constraints is keeping a single item and embedding the attribute values 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 SPM 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 [8], [9], where there are a small number of predefined items and interleaving is not an issue, has already addressed incorporating variables. This paper presents a new SPM, Sequence Mining of Temporal Clusters (SMTC), for domains with large numbers of items that are discovered during mining and where interleaving exists. SMTC also addresses interleaving in an innovative way by first clustering temporally close items and only creating sequences after the entire dataset is examined.

SMTC is tested on a digital forensics dataset, a domain that is very applicable to the issues that SMTC addresses. It is a domain with a large amount of data. Events occur every millisecond and the average computer contains in excess of a million files. In addition, with most computers having multiple processors, interleaving is also an issue. Using a virtual

- *The authors are with the Department of Electrical and Computer Engineering, Air Force Institute of Technology, WPAFB, OH 45433.*
  *E-mail: jsokolica@yahoo.com, {gilbert.peterson, robert.mills, michael.grimaila}@afit.edu.*

*Manuscript received 26 Jan. 2018; accepted 2 Nov. 2018. Date of publication 19 Nov. 2018; date of current version 5 Dec. 2019.*
*(Corresponding author: James S. Okolica.)*
*Recommended for acceptance by E. Keogh.*
*Digital Object Identifier no. 10.1109/TKDE.2018.2881675*

machine, the digital forensics dataset is constructed where the SPM items are the digital artifacts acquired during an incident response. The experiment tests if SMTC creates terms and sequences that can be used to create production rules to discover abstract human activities. It is compared against an existing SPM, Discontinuous Varied Order Sequence Mining (DVSM) [10], that has been extended to include variable assignment of item attributes (DVSM-V) to measure SMTC's new method for addressing interleaving.

When SMTC and DVSM-V are applied to the constructed dataset, they generate sequences that result in a 96 percent reduction in the data. SMTC mines the sequences in approximately two minutes while DVSM-V takes approximately ten minutes. The terms and sequences generated are successfully used to create production rules for discovering instances of users starting Microsoft Office applications. Furthermore, when these production rules are applied to unseen data using a "leave one out" methodology, 100 percent of the instances of the user activity are discovered while the false positive rate remains below 0.030 percent. SMTC is shown to produce a slightly lower false positive rate at a higher similarity threshold, suggesting it slightly outperforms DVSM-V.

## 2 Related Work

Aggrawal and Srikant [11] 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 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 [12]. The first, Apriori-Based Algorithms, extend the original algorithms developed by Agrawal [13]. 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 [14] 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) [10] is a recent Apriori-based SPM algorithm applied to the activities of daily living (ADL) domain where activities are the items being mined for sequences. It 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 the minimum amount the dataset must be reduced by subsituting a symbol for the sequence $a$, $D$ is the dataset, $a$ is a potential sequence and $DL$ is the number of bytes to encode the sequence. 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.

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 [15] and the Levenshtein edit distance [16] to measure similarity. Frequency is based on the minimum description length [17] 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 [8] and MiTempP [9]. Lee and Raedt [8] created SeqLog as a logical language that handles sequences of logical atoms with variables attached to them. Lattner et al. [9] 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 [6], [7]. 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.

### 2.1 Attaching Variables to SPM Algorithms

A *variable* is a symbol that may assume a given value subject to specified constraints and a *term* is an SPM item that contains one or more variables. 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 percent of the time he later purchases a video card (e.g., $S2 = \texttt{memory} + \texttt{storage} + \texttt{video card}$). That may be statistically significant, but does it merit targeted advertising? If a `price`

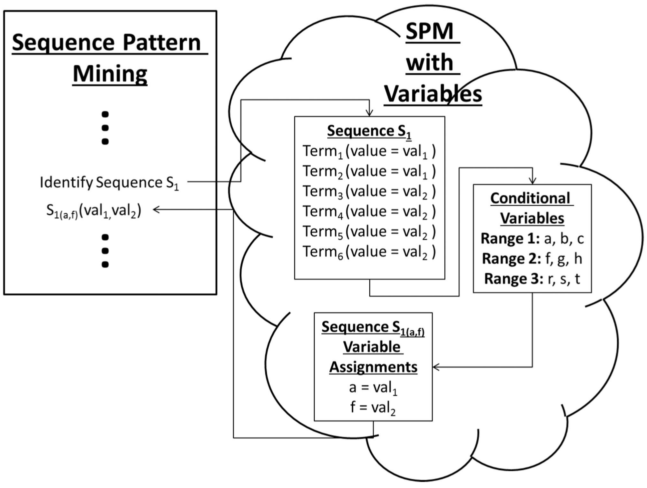

Fig. 1. Process for attaching variables to SPM.

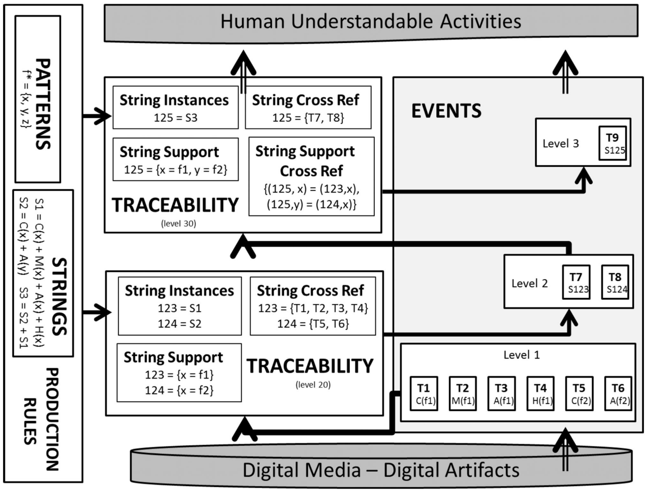

Fig. 2. Providing traceability for attaching variables.

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 percent of the time. Using variables as a receptacle to store attribute values allows better analysis of the mined sequences.

*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 $100.00. Any time these variables are associated with a term the value of the attributes that the variables are assigned to will be within the constrained space. Within the term, the relationship between the variable, attribute and value is called a *cross reference*. Conditional variables enable sequences based on the values of the attributes. In the previous example, variables enable sequences like `memory`$(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 focuses on items with a single attribute that can have a variable number of values; however, everything discussed generalizes to multiple attributes without loss of generality.

As shown in Fig. 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 Fig. 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 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 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 Fig. 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 Fig. 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.

**Algorithm 1.** SPM - Variable Attachment

```
Input
  $S_i$ - a sequence of terms
  $Conditions$ - 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
      $v = Conditions.getVariable(NULL)$
      $t_{new}.variable \leftarrow v$
    else if $t_{new}.isSingleValue()$ then
      $v = Used_v.getVariable(t.getAttributes()$
      if $v == NULL$ then
        $v = Conditions.$
            $getVariable(t.getAttributes())$
      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
          $v = Conditions.$
              $getVariable(t.getAttributes())$
        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
        $v_{old} \leftarrow Xref.getVariable(wattribute)$
        $v_{new} \leftarrow Used_v.getVariable(wattribute)$
        if $v_{new} == NULL$ then
          $v_{new} = Conditions.$
              $getVariable(wattribute)$
          $Used_v \leftarrow Used_v \bigcup \{v_{new}, wattribute\}$
        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
```

## 3 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 for sequences and then a second time for loops (i.e., sequences of terms where the items are all the same but the attribute values vary).

One initial complication that arises when performing abstraction on sequences with variables concerns the distance measurement. A common algorithm for measuring the distance between two sequences is the Levenshtein edit distance [16]. 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.

### 3.1 Sequence Mining of Temporal Clusters (SMTC)

While there are several SPM algorithms that handle interleaving, many handle it indirectly as part of determining whether a candidate sequence is a variation of a previously discovered one. Interleaving occurs when two or more sequences of terms occur simultaneously. For instance, if sequence 1 is $ABBCD$ and sequence 2 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 impossible 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 [16]. 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

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

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

**Algorithm 2.** Sequence Mining of Temporal Clusters (SMTC)

```
Input
  $D$ - the dataset - a sequence of terms
  $M_S$ - maximum terms in a sequence
  $M_\tau$ - maximum workspace time lag
  $\epsilon$ - 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, $\bigcup s_i$
          $s_n \rightarrow t_1 + t_2 + \cdots + t_n$ where
          $t_1, t_2, \ldots, t_n \in D$. getTerms()
          $\bigcup D$.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
  $S, W, C \leftarrow \emptyset$
  $D.gotoBeginning()$
  while $(t \leftarrow D.nextTerm()) \neq NULL$ do
    if $(|W| > M_S$ or $distance(w_1, t) > M_\tau)$ then
      $P \leftarrow powerset(W)$
      for $p \in P$ do
        $c \leftarrow t$
        if $c \in C$ then
          $c.occurrences++$
          $c.addInstantiations($
                  $p.getInstantiations())$
        else
          $C = C \bigcup c$
          $c.occurrences \leftarrow 1$
          $c.addInstantiations($
                  $p.getInstantiations())$
        end if
      end for
      Remove terms from workspace till
              new term fits in workspace
    end if
  end while
  $S \leftarrow KeepInteresting(C, \epsilon, threshold)$
```

There are several parameters associated with SMTC. The first parameter is the maximum number of terms in the workspace which is determined by considering how big of a power set the machine SMTC is executed on can handle. The second parameter is a weight term for the number of different variables in a sequence. Consider two sequences, $S1 = a(price \in (0, 100)) + b(price \in (0, 100)) + c(price \in (101, 200)) + a(price \in (101, 200)) + b(price \in (101, 200))$ and $S2 = a(price \in (0, 100)) + b(price \in (101, 200)) + c(price \in (201, 300)) + a(price \in (301, 400)) + b(price \in (401, 500))$. If these two sequences occur the same number of times, then $S1$ would be preferred over $S2$ because it contains fewer variables (e.g., $price \in (0,100)$ and $price \in (101,200)$ versus $price \in (0,100)$, $price \in (101,200)$, $price \in (201,300)$, $price \in (301,400)$, and $price \in (401,500)$ ). The remaining parameters are the maximum time lag for terms to be considered temporally close and the threshold score for SMTC to keep a cluster of terms. Finally, in Algorithm 2, an *instantiation* of a sequence is one of its location within the dataset. For instance, if sequence $S1$ occurs as items 3 through 7, items 87 through 91, items 118 through 122, and items 156 to 160, then it has four instantiations, $(3,7)$, $(87,91)$, $(118,122)$, and $(156,160)$.

## 4 Application Domain and Dataset

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" [18]. 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 a change to the digital media and as a record of the digital items on the digital media. For the purpose of this research, we are concerned with digital artifacts that contain temporal information. For instance, in the Windows NT Master File Table (MFT), each file stored on digital media has additional information stored with it that includes 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. 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.

In the digital forensics domain, we are interested in discovering user activities. These activities appear in the data as sequences of digital artifacts (i.e., an SPM item). 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 items. 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 action (e.g., create, access, download, etc.) performed on each file. Instead, the names of the files, registry keys, etc. must be considered attributes of a much smaller number of common items. Incorporating variables into SPM is one method for handling this.

### 4.1 Previous Work in Digital Forensics

A common requirement in Digital Forensics is to summarize activities that occurred on a digital device [19]. Early work on this problem used pre-installed sensors [20] and then mined the resulting data. While research involving pre-installed sensors continues [21], [22], there is also now research without them [5], [23]. This is important for applying digital forensics to non-corporate crimes, especially those that are non-digital. Both Zeitline [23] and Python Digital Forensic Timeline (PyDFT) [5] 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. [24] 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.

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 |
| 1f | User opens Google Chrome 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 |

### 4.2 The Dataset

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, and Microsoft PowerPoint. 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 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 master file table (MFT), along with 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 from 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.

TABLE 2
Locations of Extracted Data Files (excluding the MFT)

| Data Type | File System Location |
|---|---|
| Event Logs | /%WINDIR%/System32/winevt/Logs |
| Registry Files | /%WINDIR%/System32/config<br>/%USERPROFILE%/AppData/NTUSER.DAT<br>/%USERPROFILE%/AppData/Local/Microsoft/Windows/UsrClass.DAT |
| Browser History | /%USERPROFILE%/AppData/Local/Microsoft/Windows/History/history.ie5<br>/%USERPROFILE%/AppData/Local/Google/Chrome/User Data/Default/History<br>/%USERPROFILE%/AppData/Roaming/Mozilla/Firefox/Profiles/*/places.sqlite |

## 5 The Experiment

The first step in the experiment is taking the items found in the dataset and mining a set of constraints for them. For the digital forensics domain, the attribute value of interest is the item name, e.g., file name, registry key name, device name, uniform resource locator (url) name, etc. While parts of the item name are important to digital forensics analysts, other parts are not. For instance, a file name may be `c:/Users/user/AppData/Local/ Mozilla/Firefox/Profiles/ryuzhk.default /cache2/entries/00454D0AC349F-FA42D`. While part of the name is of interest (e.g., `AppData/Local/Mozilla/Firefox/Profiles`), the rest is not. So for this domain, the constraints are regular expressions for matching item names (e.g., `^*AppData/Local/Mozilla/Firefox/Profiles/ .*$`). To discover what these constraints are, the experiment begins by an exhaustive examination of the number of times each subset of words occur in the dataset (e.g., how many times `c:/`, `c:/Users`, `c:/Users/user`, `c:/.*/user`, etc) occurs. The number of occurrences is multiplied by the number of words in the subset to arrive at a value for subset. A threshold is then set and all subsets with a value above that threshold are included as constraints. Different numbers of constraints were tested and when the number rose above 800, the processing time was too long.

In the digital forensics domain, there are large number of sequences where adjacent terms are multiple actions on the same digital artifact (e.g., creating a file, accessing it and then changing its meta-information). More generally, there are a large number of sequences where adjacent terms have attributes with the same attribute values. Single Object Sequence and Loop Abstraction (SOSLA) is a greedy algorithm that creates sequences based on adjacent terms with the same attribute values. All possible sequences where adjacent terms have the same attribute values (single object sequences) are created, e.g., $S1 = create(fileName_1) + modify(fileName_1) + access(fileName_1) + changeMetaInformation(fileName_1)$ and $S2 = create(fileName_1) + access(fileName1)$ where create, modify, access, and changeMetaInformation are items and $fileName1$ is a variable. Given its simplicity, pseudocode for the algorithm is not shown. As shown in Fig. 3, the process begins by applying SOSLA to the extracted digital artifacts. Once SOSLA has created its single-object sequences, the initial dataset is abstracted using those sequences. Next, two copies of the resulting abstracted dataset are created.

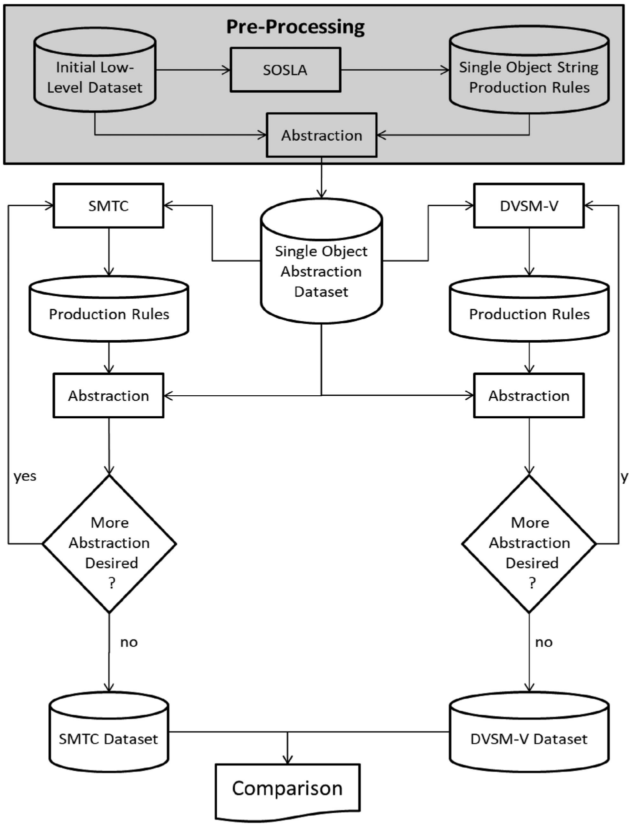
Pre-Processing
Initial Low-Level Dataset
SOSLA
Single Object String Production Rules
Abstraction
SMTC
Single Object Abstraction Dataset
DVSM-V
Production Rules
Production Rules
Abstraction
Abstraction
yes
yes
More Abstraction Desired ?
More Abstraction Desired ?
no
no
SMTC Dataset
DVSM-V Dataset
Comparison

Fig. 3. Abstraction methodology.

SMTC is applied to one copy of the abstracted dataset and DVSM [10] with variables is applied to the other. As shown in Fig. 1, to extend DVSM to include variables (DVSM-V), a hook is added in the DVSM algorithm after a sequence is identified that calls the Variable Attachment algorithm (algorithm 1). The Variable Attachment algorithm inspects the particular instantiation of the candidate sequence and assigns variables to it based on the variable constraints. It then returns to DVSM the sequence identifier tied to that sequence of items with those variables assigned. DVSM then continues its normal processing.

For both SMTC and DVSM-V, there are several parameters. There is a maximum size sequence that is considered. During testing of SOSLA, it was shown that no single term sequence was ever larger than eight terms and thus a maximum size of eight was used. For SMTC and DVSM-V, 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 Fig. 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 are shown in Tables 3 and 4. Both SMTC and DVSM-V show a 96 percent reduction in the number of items. Despite the dataset's size, both algorithms are able to quickly mine it. However, while DVSM-V takes ten minutes to mine the dataset, SMTC only takes one minute. Even accounting for the fact that SMTC is run twice, this is still an 80 percent reduction in elapsed time.

Once the dataset has been sufficiently abstracted, the final step is creating a set of production rules that describe

TABLE 3
Use Case Dataset Event Compression

| Use Case | Low Level Events | SMTC Events | DVSM-V Events |
|---|---|---|---|
| Use Case 1 | 18.6 M | 736 | 722 K |
| Use Case 2 | 16.9 M | 1.9 M | 1.00 M |
| Use Case 3 | 17.1 M | 664 K | 718 K |

TABLE 4
Use Case Dataset Term and Sequence Compression

| Use Case | SMTC Terms/Sequences | DVSM-V Terms/Sequences |
|---|---|---|
| Use Case 1 | 972/ 2,104 | 414/ 911 |
| Use Case 2 | 941/ 540 | 414/ 454 |
| Use Case 3 | 1,259/ 2,930 | 534/ 1,208 |

starting the appropriate application (e.g., MS Word, MS PowerPoint, MS Excel). First, the sequence of terms for each scenario is extracted by using the detailed times recorded during creating the use cases of when the application begins loading and when it finishes loading. The activities book-ended by these times are all considered to be part of starting the application. As an example, Tables 5 and 6 show the terms and sequences involved in starting Microsoft Word and Table 7 shows how they combine for the different scenarios. For instance, to start Microsoft Word, UC1a begins with sequence 667 which modifies and changes the meta information of a file in the `/Windows` directory. The next item in the sequence, term 230, modifies a registry key that has the name "Windows" in it followed by term 112 which modifies a registry key whose name can be anything. While these constraints suggest very loosely defined names, when the data is examined, the names show much more consistency. For instance, for all use cases, the initial instantiation of sequence 667 always has a file name constraint of `^.*/Windows/Prefetch/DLLHOST.EXE.*$` associated with it and the initial instantiation of term 230 always has a registry key name constraint of `^ntuser.dat/SOFTWARE/Microsoft/Windows/ CurrentVersion/Explorer/UserAssist/.*$`. The reason for the looser constraints is the initial mining of constraints. To keep the number of name constraints under 800 for the entire dataset, only looser constraints were kept. Even with these loose constraints, the number of terms and sequences that are in the sequence for starting the Microsoft applications result in a low number of false positives. However, to provide greater understandability to subject matter experts, additional constraints could be manually added in specific cases and the terms in the sequences manually replaced with new terms that use these constraints. Finally, there are several sequences (e.g., S735, S754, S757, etc.) 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).

Once the sequences for each scenario of a given use case are extracted, they are consolidated into a single set of production rules. Table 8 shows a subset of the production rules generated after the sequences in Table 7 are consolidated where rule S is the starting rule. This particular subset are the rules that fire when a sequence matching UC1a is encountered. Table 9 then shows an example of those production rules firing 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.

TABLE 5
Representative Terms for Starting Microsoft Word

| Term ID | Description | Conditional Variable |
|---|---|---|
| 1 | Create File | ^.*/Windows/.*$ |
| 2 | Modify File | ^.*/Windows/.*$ |
| 7 | Create File | ^.*$ |
| 8 | Access File | ^.*$ |
| 13 | Modify File | ^.*$ |
| 22 | Modify File | ^.*/Users/user/AppData/.*/ Microsoft/.*$ |
| 29 | Create File | ^.*/Users/user/AppData/.*/ Microsoft/.*$ |
| 30 | Access File | ^.*/Users/user/AppData/.*/ Microsoft/.*$ |
| 36 | Modify File | ^.*/Users/user/AppData/ Local/.*$ |
| 40 | Access File | ^.*/Windows/.*$ |
| 47 | Create File | ^.*/Windows/Logs/.*$ |
| 48 | Modify Registry Key | ^Software/Microsoft/Windows CurrentVersion/Component Based Servicing/.*$ |
| 74 | Access File | ^.*/Users/.*/AppData/.*/ Microsoft/.*$ |
| 90 | Modify File | ^.*/Windows/.*/Windows/.*$ |
| 112 | Modify Registry Key | ^.*$ |
| 131 | Modify File | ^.*/Windows/Logs/.*$ |
| 132 | Access File | ^.*/Windows/Logs/.*$ |
| 145 | Modify Registry Key | ^.*/Windows/.*$ |
| 152 | Modify File | ^.*/$OrphanFiles/.*$ |
| 150 | User Modifies Registry Key | ^.*$ |
| 199 | Create File | ^.*/Users/user/AppData/ Local/.*$ |
| 200 | Access File | ^.*/Users/user/AppData/ Local/.*$ |
| 202 | Change Meta Information | ^.*/Windows/.*$ |
| 203 | Change Meta Information | ^.*/Windows/.*/Windows/.*$ |
| 205 | Change Meta Information | ^.*$ |
| 207 | Change Meta Information | ^.*/Windows/Logs/.*$ |
| 211 | Create File | ^.*/Windows/WinSxS/.*$ |
| 212 | Access File | ^.*/Windows/WinSxS/.*$ |
| 213 | Modify File | ^.*/Windows/WinSxS/.*$ |
| 214 | Change Meta Information | ^.*/Windows/WinSxS/.*$ |
| 228 | Changes Meta Information | ^.*/Users/user/AppData/Local/.*$ |
| 229 | Change Meta Information | ^.*/Users/user/AppData/.*/ Microsoft/.*$ |
| 230 | User Modifies Registry Key | ^.*/Windows/.*$ |
| 310 | User Accesses File | ^.*$ |
| 313 | Create File | ^.*/$OrphanFiles/.*$ |
| 314 | Access File | ^.*/$OrphanFiles/.*$ |
| 315 | Change Meta Information | ^.*/$OrphanFiles/.*$ |

TABLE 6
Representative Sequences for Starting Microsoft Word

| Seq ID | Sequence of Terms | Conditional Variable |
|---|---|---|
| 342 | 313 ($v_1$) + 152 ($v_1$) + 314 ($v_1$) + 315 ($v_1$) | ˆ.*/$OrphanFiles/.*$ |
| 347 | 90 ($v_1$) + 203 ($v_1$) | ˆ.*/Windows/$ |
| 377 | 22 ($v_1$) + 228 ($v_1$) | ˆ.*/Users/user/AppData/ Local/.*$ |
| 428 | 211 ($v_1$) + 211 ($v_1$) | ˆ.*/Windows/WinSxS/.*$ |
| 490 | 131 ($v_1$) + 207 ($v_1$) | ˆ.*/Windows/Logs/.*$ |
| 643 | 1 ($v_1$) + 40 ($v_1$) + 1 ($v_1$) + 40 ($v_1$) | ˆ.*/Windows/.*$ |
| 667 | 2 ($v_1$) + 202 ($v_1$) | ˆ.*/Windows/.*$ |
| 676 | 2 ($v_1$) + 40 ($v_1$) + 202 ($v_1$) | ˆ.*/Windows/.*$ |
| 677 | 1 ($v_1$) + 2 ($v_1$) + 40 ($v_1$) + 202 ($v_1$) + 1 ($v_1$) + 2 ($v_1$) + 40 ($v_1$) + 202 ($v_1$) | ˆ.*/Windows/.*$ |
| 686 | 47 ($v_1$) + 131 ($v_1$) + 132 ($v_1$) + 207 ($v_1$) | ˆ.*/Windows/Logs/.*$ |
| 694 | 13 ($v_1$) + 8 ($v_1$) + 205 ($v_1$) + 13 ($v_1$) + 8 ($v_1$) + 205 ($v_1$) | ˆ.*$ |
| 706 | 13 ($v_1$) + 8 ($v_1$) + 205 ($v_1$) | ˆ.*$ |
| 735 | 211($v_1$ + 213 ($v_1$) + 212($v_1$) + 214 ($v_1$) + 211($v_1$ + 213 ($v_1$) + 212($v_1$) + 214 ($v_1$) | ˆ.*/Windows/WinSxS/.*$ |
| 754 | 199 ($v_1$) + 36 ($v_1$) + 200 ($v_1$) + 228 ($v_1$) + 199 ($v_1$) + 36 ($v_1$) + 200 ($v_1$) + 228 ($v_1$) | ˆ.*/Users/user/AppData/ Local/.*$ |
| 757 | 29 ($v_1$) + 22 ($v_1$) + 30 ($v_1$) + 229 ($v_1$) + 29 ($v_1$) + 22 ($v_1$) + 30 ($v_1$) + 229 ($v_1$) | ˆ.*/Users/user/AppData/ .*/Microsoft/.*$ |
| 759 | 22 ($v_1$) + 229 ($v_1$) | ˆ.*/Users/user/AppData/ .*/Microsoft/.*$ |
| 766 | 22 ($v_1$) + 30 ($v_1$) + 229 ($v_1$) | ˆ.*/Users/user/AppData/ .*/Microsoft/.*$ |
| 768 | 29 ($v_1$) + 29 ($v_1$) | ˆ.*/Users/user/AppData/ .*/Microsoft/.*$ |
| 777 | 22 ($v_1$) + 30 ($v_1$) + 229 ($v_1$) 22 ($v_1$) + 30 ($v_1$) + 229 ($v_1$) | ˆ.*/Users/user/AppData/ .*/Microsoft/.*$ |
| 836 | 213 ($v_1$) + 212($v_1$) + 214 ($v_1$) | ˆ.*/Windows/WinSxS/.*$ |
| 940 | 36 ($v_1$) + 200 ($v_1$) + 228 ($v_1$) | ˆ.*/Users/user/AppData/ Local/.*$ |
| 1047 | 131 ($v_1$) + 132 ($v_1$) + 207 ($v_1$) | ˆ.*/Windows/Logs/.*$ |
| 1050 | 2 ($v_1$) + 202 ($v_1$) + 2 ($v_1$) + 202 ($v_1$) | ˆ.*/Windows/.*$ |
| 1063 | 313 ($v_1$) + 152 ($v_1$) + 314 ($v_1$) + 315 ($v_1$) + 313 ($v_1$) + 314 ($v_1$) | ˆ.*/$OrphanFiles/.*$ |
| 1070 | 213 ($v_1$) + 214 ($v_1$) | ˆ.*/Windows/WinSxS/.*$ |
| 1080 | 29 ($v_1$) + 22 ($v_1$) + 30 ($v_1$) + 229 ($v_1$) + 29 ($v_1$) + 30 ($v_1$) | ˆ.*/Users/user/AppData/ .*/Microsoft/.*$ |

TABLE 7
Example Sequences for Starting Microsoft Word

| Scenario | Sequence |
|---|---|
| UC1a | S667 + T230 + T112 + T150 + S1080 + T766 + T757 + S667 + T112 + S667 + S667 + S676 + S677 + S667 + S230 + T145 + S667 |
| UC1b | S667 + S667 + S667 + S310 + T230 + S667 + T230 + T112 + T150 + T145 + S667 + T112 + S667 + S667 + T112 + S676 + S677 + S667 + T145 |
| UC1c | T230 + S667 + T112 + S667 + S667 + S667 + T150 + T230 + T112 + T145 + S667 + S667 + S676 + S677 + S667 + S667 + T112 |
| UC1d | T13 + T230 + S706 + S768 + T205 + T150 + T230 + S667 + T150 + T230 + S667 + T112 + T145 + S667 + S676 + S677 + S667 + S667 + T112 + S766 + S777 + S759 + S766 + S759 + S777 + T230 + T150 + T112 + S766 + S757 + S759 + T310 + T230 + T230 + S759 + S759 + T150 + T230 + T150 + S766 + T150 + T112 + T145 + S940 + T150 |
| UC1e | S768 + T205 + T230 + S766 + S757 + T230 + S667 + S766 + S777 + S759 + T150 + S667 + T112 + T145 + S667 + S676 + S677 + S667 + S754 + S667 + T150 + S766 + S759 + S777 + T150 + T230 |
| UC1f | T150 + S766 + S759 + S766 + S676 + S677 + S1070 + T150 + S836 + T214 + T150 + T145 + T112 + S428 + S735 + S940 + T228 + S667 + S735 + S1196 + S735 + T214 + T150 + S667 + S347 |
| UC1g | S667 + T230 + T112 + T150 + T112 + S667 + T150 + T145 + S676 + S677 + T145 + S667 + S706 + T145 + S1070 + T112 + S667 + S667 |
| UC1h | S940 + S377 + T228 + S667 + T112 + T150 + S759 + T150 + T112 + S667 + S643 + S676 + S1050 + S667 + T150 + S667 + S1063 + T112 + T40 + S766 + T315 + S490 + T202 + T112 + S667 + T150 + T230 + T145 + S766 + S757 + S667 + T230 + S667 + S490 + T48 + S759 + S667 + T13 + T74 + T8 + T40 +S667 + S342 + S677 + S757 + S766 + T230 + S667 + T112 + S677 + S667 + T230 + S754 + S667 + T230 + S777 + S667 + T112 + T112 + S676 + S686 + T112 + S1047 + T112 + S676 +S667 + T202 + S667 + S706 + T145 + T145 + S766 + S777 +S759 + S694 + T150 + S667 + S759 + T145 + S706 + T150 + S777 + T150 + S759 + T230 + T112 + S766 + S759 + S777 + T230 + T112 + S754 + S766 + S940 + S766 + T150 + S377 + S667 |

TABLE 8
Mined Production Rules for Starting Microsoft Word (UC1a)

| ID | | Rule |
|---|---|---|
| S | → | R285 + R1012 |
| R3 | → | T112 + T150 + S1080 + S766 + S757 |
| R12 | → | S676 + S677 |
| R21 | → | T112 + S667 |
| R28 | → | T230 + T145 |
| R84 | → | R21 + S667 + R12 |
| R252 | → | S667 + T230 + R3 |
| R284 | → | R252 + S667 + R84 |
| R285 | → | R284 + S667 + R28 |
| R1012 | → | S667 |

## 6 RESULTS

Once the terms and sequences are generated from the Use Case dataset, they are then tested using a "leave one out" technique to see how well they generalize to new data. Each use case is composed of between six and eight scenarios. For each use case, $N$ tests are done where $N$ is the number of scenarios. For each test, one scenario is left out when the production rules are generated. Then the production rules are tested on that scenario to see if the actual user activity (e.g., starting Microsoft Word) is found and if the production rules find any other instances of the user activity. These are called *true positives* and *false positives* respectively. If the true positive is missed, this is called a *false negative*. The similarity

TABLE 9
Example of Production Rules Generating UC1a's Sequence for Starting Microsoft Word

S
R285 + R1012
R284 + S667 + R28 + R1012
R252 + S667 + R84 + S667 + R28 + R1012
S667 + T230 + R3 + S667 + R84 + S667 + R28 + R1012
S667 + T230 + T112 + T150 + S1080 + S766 + S757 + S667 + R84 + S667 + R28 + R1012
S667 + T230 + T112 + T150 + 1080 + S766 + S757 + S667 + R21 + S667 + R12 + S667 + R28 + R1012
S667 + T230 + T112 + T150 + S1080 + S766 + S757 + S667 + T112 + S667 + S667 + R12 + S667 + R28 + R1012
S667 + T230 + T112 + T150 + S1080 + S766 + S757 + S667 + T112 + S667 + S667 + S676 + S677 + S667 + R28 + R1012
S667 + T230 + T112 + T150 + S1080 + S766 + S757 + S667 + T112 + S667 + S667 + S676 + S677 + S667 + T230 + T145 + R1012
S667 + T230 + T112 + T150 + S1080 + S766 + S757 + S667 + T112 + S667 + S667 + S676 + S677 + S667 + T230 + T145 + S667

TABLE 10
SMTC Performance on Use Case Dataset

| Use Case Scenario | Similarity Threshold | True Positives | False Positives | False Negatives |
|---|---|---|---|---|
| UC1a | 3 | 1 | 1 | 0 |
| UC1b | 3 | 1 | 1 | 0 |
| UC1c | 3 | 1 | 0 | 0 |
| UC1d | 3 | 1 | 2 | 0 |
| UC1e | 3 | 1 | 1 | 0 |
| UC1f | 3 | 1 | 1 | 0 |
| UC1g | 3 | 1 | 0 | 0 |
| UC1h | 3 | 1 | 0 | 0 |
| UC2a | 3 | 1 | 0 | 0 |
| UC2b | 3 | 1 | 0 | 0 |
| UC2c | 2 | 1 | 0 | 0 |
| UC2e | 3 | 1 | 0 | 0 |
| UC2f | 3 | 1 | 1 | 0 |
| UC2h | 4 | 1 | 21 | 0 |
| UC3a | 4 | 1 | 17 | 0 |
| UC3b | 4 | 1 | 15 | 0 |
| UC3d | 4 | 1 | 18 | 0 |
| UC3e | 4 | 1 | 25 | 0 |
| UC3f | 4 | 1 | 21 | 0 |
| UC3g | 4 | 1 | 20 | 0 |
| UC3h | 4 | 1 | 22 | 0 |

TABLE 11
DVSM-V Performance on Use Case Dataset

| Use Case Scenario | Similarity Threshold | True Positives | False Positives | False Negatives |
|---|---|---|---|---|
| UC1a | 3 | 1 | 1 | 0 |
| UC1b | 3 | 1 | 0 | 0 |
| UC1c | 3 | 1 | 1 | 0 |
| UC1d | 3 | 1 | 2 | 0 |
| UC1e | 3 | 1 | 2 | 0 |
| UC1f | 3 | 1 | 0 | 0 |
| UC1g | 3 | 1 | 0 | 0 |
| UC1h | 3 | 1 | 0 | 0 |
| UC2a | 4 | 1 | 9 | 0 |
| UC2b | 4 | 1 | 5 | 0 |
| UC2c | 2 | 1 | 3 | 0 |
| UC2e | 4 | 1 | 24 | 0 |
| UC2f | 3 | 1 | 8 | 0 |
| UC2h | 3 | 1 | 16 | 0 |
| UC3a | 3 | 1 | 29 | 0 |
| UC3b | 3 | 1 | 27 | 0 |
| UC3d | 4 | 1 | 26 | 0 |
| UC3e | 2 | 1 | 19 | 0 |
| UC3f | 2 | 1 | 19 | 0 |
| UC3g | 2 | 1 | 9 | 0 |
| UC3h | 3 | 1 | 20 | 0 |

threshold is the maximum number of edit operations needed using the Levenshtein edit distance to transform one sequence into the other. The threshold was set empirically to ensure that there are no false negatives.

As can be seen in Tables 10 and 11, the similarity threshold was consistent at between 2 and 4 edit operations. This threshold enabled all user activities to be identified while keeping the false positives low. To put the number of false positives in context, each scenario has over 100,000 events. That means that the number of false positives could be as high as 100,000 (assuming each event is the first event in exactly one sequence). Thus, 30 false positives is equivalent to a false positive rate of no more than 0.03 percent.

The highest number of false positives for a specific scenario was 29 which is sufficiently low to allow a digital forensic examiner to perform their work efficiently. 29 false positives results in a maximum false positive rate of 0.03 percent. However, the average false positive rate is 0.0079 percent for SMTC and 0.0104 percent for DVSM-V. While setting the similarity threshold to guarantee all positive instances are found prevents us from knowing how well the algorithm generalizes to previously unseen data, the fact that in all cases, a threshold of 4 is sufficient suggests that a similar threshold will likely work for new datasets.

## 7 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. 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 SMTC and DVSM-V to the a constructed dataset in the digital forensics domain. Applying these algorithms to a constructed dataset results in a 96 percent reduction in the number of terms and the creation of production rules for recognizing user activities. When those production rules are tested using a “leave one out” methodology and the similarity threshold is set to ensure the true positive rate is 100 percent, the false positive rate is kept below 0.03 percent, with SMTC having a lower average false positive rate. Furthermore, SMTC mines the data in 10 percent of the time that it takes DVSM-V. Even taking into account that SMTC must mine the data twice, due to its restricted maximum sequence size, this is still an 80 percent reduction in elapsed time.

There are two separate directions that future work may go. The first concerns SPM and the second concerns its application to digital forensics. Further SPM testing should look to apply SMTC and the Variable Attachment algorithm to other domains and other (potentially non-Apriori) SPM algorithms. Furthermore, future testing of SMTC should include parallelizing it to allow mining longer sequences in a single pass. In the digital forensics domain, additional use cases should be developing 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.

## ACKNOWLEDGMENTS

The views expressed in this paper are those of the authors and do not represent the views or policies of the United States Air Force or the Department of Defense.

## REFERENCES

[1] S. Brin, R. Motwani, J. Ullman, and S. Tsur, "Dynamic itemset counting and implications rules for market basket data," in *Proc. ACM Special Interest Group Manag. Data Conf. Manag. Data*, 1997, vol. 26, pp. 255–264.

[2] N. Jindal and B. Liu, "Identifying comparative sentences in text documents," in *Proc. 29th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2006, pp. 244–251.

[3] G. Bejerano and G. Yona, "Modeling protein families using probabilistic suffix trees," in *Proc. 3rd Annu. Int. Conf. Comput. Molecular Biol.*, 1999, pp. 15–24.

[4] D. J. Cook, N. C. Krishnan, and P. Rashidi, "Activity discovery and activity recognition: A new partnership," *IEEE Trans. Cybern.*, vol. 43, no. 3, pp. 820–828, Jun. 2013.

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

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

[7] C. Bucilă, J. Gehrke, D. Kifer, and W. White, "Dualminer: A dual-pruning algorithm for itemsets with constraints," *Data Mining Knowl. Discovery*, vol. 7, no. 3, pp. 241–272, 2003.

[8] S. Dan Lee and L. De Raedt, *Constraint Based Mining of First Order Sequences in SeqLog*. Berlin, Germany: Springer, 2004, pp. 154–173.

[9] A. D. Lattner and O. Herzog, "Constraining the search space in temporal pattern mining," in *Proc. LWA*, 2006, pp. 314–321.

[10] P. Rashidi, D. J. Cook, L. B. Holder, and M. Schmitter-Edgecombe, "Discovering activities to recognize and track in a smart environment," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 4, pp. 527–539, Apr. 2011.

[11] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proc. Int. Conf. Data Eng.*, 1995, pp. 3–14.

[12] C. H. Mooney and J. F. Roddick, "Sequential pattern mining–approaches and algorithms," *ACM Comput. Surveys*, vol. 45, no. 2, 2013, Art. no. 19.

[13] R. Agrawal, R. Srikant, et al., "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases*, 1994, vol. 1215, pp. 487–499.

[14] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovering frequent episodes in sequences extended abstract," in *Proc. 1st Conf. Knowl. Discovery Data Mining*, 1995, pp. 210–215.

[15] J. Yang and W. Wang, "Towards automatic clustering of protein sequences," in *Proc. IEEE Comput. Soc. Bioinf. Conf.*, 2002, pp. 175–186.

[16] V. Levenshtein, "Binary codes capable of correct deletions, insertions and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[17] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, pp. 465–471, 1978.

[18] A. Marcella and D. Menendez, *Cyber Forensics: A Field Manual for Collecting, Examining, and Preserving Evidency of Computer Crimes*, 2nd ed., Boca Raton, FL, USA: Auerbach Publications, 2007.

[19] S. L. Garfinkel, "Digital forensics research: The next 10 years," *Digital Investigation*, vol. 7, pp. S64–S73, 2010.

[20] S. T. King and P. M. Chen, "Backtracking intrusions," in *Proc. 19th ACM Symp. Operating Syst. Principles*, 2003, pp. 223–236.

[21] M. N. A. Khan, "Performance analysis of Bayesian networks and neural networks in classification of file system activities," *Comput. Security*, vol. 31, no. 4, pp. 391–401, 2012.

[22] Y.-C. Liao and H. Langweng, "Resource-based event reconstruction of digital crime scenes," in *Proc. IEEE Joint Intell. Security Informat. Conf.*, 2014, pp. 129–136.

[23] F. P. Buchholz and C. Falk, "Design and implementation of zeitline: A forensic timeline editor," in *Proc. Digital Forensics Res. Workshop*, 2005, pp. 1–7.

[24] 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.

**James S. Okolica** received the BS degree in computer science in applied mathematics from Drew University, in 1990, and the MS and PhD degrees in computer science from the Air Force Institute of Technology, in 2006 and 2018. He is a research engineer with the Center for Cyberspace Research at the Air Force Institute of Technology. His current research interests include digital forensics, text mining, serious games, and machine learning.

**Gilbert L. Peterson** received the BS degree in architecture, and the MS and PhD degrees in computer science from the University of Texas at Arlington. He is an professor of computer science with the Air Force Institute of Technology, and chair of the IFIP Working Group 11.9 Digital Forensics. He teaches and conducts research in digital forensics, statistical machine learning, and autonomous robots. His research has been sponsored by the NSF, DARPA, AFOSR, AFRL, and JIEDDO. He has more than 90 peer reviewed publication, and six edited books. In 2008, he received the Air Force Junior Scientist of the Year Category I award.

**Robert F. Mills** is a professor of electrical engineering with the Department of Electrical and Computer Engineering, Air Force Institute of Technology (AFIT), Wright-Patterson AFB OH. He retired from Air Force active duty after serving 21 years as a communications/radar engineer. Since 2003, he has been on the AFIT faculty and has taught numerous courses covering topics in a wide variety of technical areas, including systems engineering, digital avionics, electronic warfare, computer networks, and cyber security. His current research interests include cyber physical security, electronic warfare, and secure systems engineering. He is a member of Eta Kappa Nu and Tau Beta Pi, and is a senior member of the IEEE.

**Michael R. Grimaila** received the BS, MS, and PhD degrees from the Texas A&M, in 1983, 1995, and 1999, respectively. He is a professor and head of the Systems Engineering and Management Department at the Air Force Institute of Technology (AFIT), Wright-Patterson AFB, Ohio. He is a fellow of the Information Systems Security Association (ISSA), a senior member of the IEEE, a member of the Association for Computing Machinery (ACM), the International Council on Systems Engineering (INCOSE), and serves as a National Research Council (NRC) Research Advisor. His research interests include computer engineering, computer and network security, data analytics, mission assurance, quantum communications and cryptography, and systems security engineering.