InfoFilter: Complex Pattern Specification and Detection Over Text Streams

Laali Elkhalifa, Raman Adaikkalavan, and Sharma Chakravarthy

Technical Report CSE-2004-1
InfoFilter: Complex Pattern Specification and Detection Over Text Streams
Laali Elkhalifa, Raman Adaikkalavan, and Sharma Chakravarthy
Computer Science and Engineering Department
The University of Texas at Arlington
Arlington, TX 76019, USA
{elkhalif, adaikkal, sharma}@cse.uta.edu

ABSTRACT
With the marked emphasis on monitoring text streams to detect complex patterns or aberrant characteristics, Information filtering is fast becoming the focal point of research. Text stream monitoring and pattern detection have far reaching applications such as tracking information flow among terrorist outfits, web parental control, continuous monitoring of rival websites in e-commerce, and so forth. InfoFilter, a content-based information filtering system presented in this paper, detects complex patterns in text streams that include but are not limited to news feed, email, web pages and caption text from streaming videos. For many applications, pattern characterization is complex and requires an expressive specification than what is currently provided by Information Retrieval Query Languages (IRQLs). In essence, pattern specification and detection play a major role in information filtering. In this paper, we describe InfoFilter, which allows users to specify complex patterns such as sequential or structural patterns, wild cards, word frequencies, proximity, Boolean operators and synonyms using the proposed Pattern Specification Language Psnoop and to detect these patterns using the data flow paradigm over Pattern Detection Graphs (PDGs).

General Terms
Information Filtering, Content-based Filtering Information Retrieval, Pattern Specification Language

Keywords
Psnoop, Pattern Detection Graphs, InfoFilter, Complex patterns, Synonyms

1. INTRODUCTION
Users often find themselves swamped with colossal amount of information present in various resources, such as the World Wide Web, while retrieving task relevant data. Information filtering is the process of extracting relevant or useful portions of information from large data repositories or continuous streams of textual data based on user patterns (or queries). In this process, expressiveness of pattern (or query) specification by a user and its detection play a significant role. Typically, a user profile in the form of one or more patterns is created and submitted to the system, and patterns in such a profile are then compared to the incoming text streams for filtering. These patterns can be simple, such as the detection of individual keywords, or complex, such as multiple sequential occurrences of words or patterns. In order to extract useful or meaningful information, the user needs to have the flexibility to specify complex patterns using an expressive pattern specification language.

Current filtering systems can be classified into content-based/cognitive filtering and social/collaborative filtering. Content-based systems filter documents/text streams based on their content or characteristics. These characteristics can be the presence or absence of a given keyword or a phrase, or the presence or absence of a particular word sequence. A number of content-based systems have been developed; these include the SIFT [1] information system implemented at Stanford, the Information Lens [2] system from MIT and the SCISOR [3]. In Collaborative filtering, documents are filtered based on the recommendations or annotations of other users who share common interests. Examples include the Tapestry [2] developed at the University of Minnesota, and the MAXIMS [4] developed at the MIT Media Lab Autonomous Agents Group.

Most of the existing filtering tools such as personalized information filtering systems use Information Retrieval Query Languages (IRQLs) [5] whose primary components are Boolean queries and regular expressions. In Boolean queries, users can specify a set of keywords or phrases linked together using Boolean operators AND, OR, and NOT. With regular expressions, users can specify patterns that have certain properties, such as a text stream that begins with the word “information”. IRQLs are used by current filtering systems to formulate user information requirements. These languages support keyword-based (e.g., single-word queries), Boolean, context and natural language queries and their compositions in a restricted way. In addition, some query languages include pattern matching queries.

Typically, a simple pattern contains only a keyword (e.g., “information”). In Boolean queries, a simple pattern can be linked with other simple patterns using AND, OR, and NOT operators. Thus a composite pattern like “information AND filtering”, can be specified and detected over a text stream. Patterns such as “information NEAR filtering WITHIN 100 WORDS” (i.e., words information and filtering occurring within a distance of 100 words) can be expressed using context queries. Basically, in context queries, a user can only specify a set of keywords that co-occur with another keyword. However, composition of Boolean, context, and some other queries in IRQLs are not usually supported. A user may be interested in detecting an entire pattern near another pattern within some distance (e.g., “information AND filtering” NEAR “information AND retrieval” WITHIN 100 WORDS). Current IRQLs cannot express composition of some complex patterns or specific number of pattern occurrences as indicated above.
Consider a real world example where a federal agent is tracking terrorist-related information streaming from various resources. He is interested in detecting the occurrence of 

\[ \text{"(bomb FOLLOWED BY ground zero) occurring twice AND} \]

automotive (or its synonyms)

\]. This pattern contains keywords, a sequential operator, phrase, cardinality, synonyms, and a Boolean operator. This pattern cannot be expressed using current query languages since IRQLs do not support the quantification of multiple occurrences (cardinality) of patterns. Moreover, a user cannot explicitly include the synonyms in his/her pattern, and is required to explicitly list the key words that are to be detected. Thus, current query languages are quite restrictive in their expressive power and hence needs to be extended and generalized to address the specification and detection of complex user patterns. This paper proposes such a framework in the form of an information filter.

Figure 1. InfoFilter with multiple users and incoming text streams

1.1 Our Contribution

In this paper we present InfoFilter, a content-based system that provides expressive pattern specification and detection. Figure 1 outlines the InfoFilter, which handles creation of user patterns using the Pattern Specification Language Psnoop, incoming text streams, WordNet [6] database tool (used for finding synonyms) and user notifications. Psnoop provides the pattern specification using the following operators and options: cardinality, synonyms, sequential, Boolean operators, structural, wild card, and proximity. Once specified, these patterns are detected using a data flow approach over Pattern Detection Graphs (PDG).

Outline: The paper is organized as follows, section 2 discusses related work, section 3 presents an overview of Pattern Specification Language Psnoop, section 4 outlines the InfoFilter, section 5 depicts the architecture of InfoFilter, section 6 discusses implementation issues and pattern detection modes, and section 7 outlines our conclusions and future work.

2. Related Work

SIFT [1] designed at Stanford University, is a content-based filtering system. Boolean queries are used to construct a user profile, allowing users to specify keywords that are to be included and those that are to be excluded, when filtering documents. To ensure efficiency, SIFT processes groups of similar user profiles, and it also allows users to apply candidate profiles (or existing profiles) against documents. On the other hand, SIFT does not consider structural information while filtering documents. It makes no distinction between positions of words in a structured text, such as those appearing in a title or the body. In the case of unstructured text, it does not take into consideration the text boundaries, such as sentences and paragraphs. It also does not take into account the word frequencies [7, 8] or its synonyms in a document.

Unlike SIFT, Infoscope [2], another content-based filtering system uses adaptive filtering to construct new user profiles efficiently. Initial user profiles are constructed using Boolean queries. It uses machine learning techniques to modify the initial user profile to adapt to the changes in the user’s interest. Infoscope also obtains implicit feedback in the form of actions suggested by user, such as the time spent by the user to read a newsgroup.

SIFT, Infoscope as well as other information filtering systems [9] use Boolean queries, and do not support proximity, regular expressions, cardinality, structural, and sequential operators. Boolean queries merely provide three operators, none of which allow the specification of the co-occurrence of words that has a certain order. In addition, number of occurrences cannot be included in Boolean queries.

IRQLs consist of various query languages such as keyword-based, context, Boolean, natural language, and pattern matching. A simple query can be formed using the keyword-based approach, where the query is a set of keywords that are to be identified in a document. A simple query can be extended using context query languages that allow the user to specify correlated words, (i.e., words that appear near each other). In addition, Boolean operators are used to compose keyword-based queries. Nevertheless, combinations of these query languages are not fully supported in current content-based filtering systems.

Complex queries can be expressed as patterns (i.e., pieces of text that have some property such as regular expressions). [5] Systems using this approach allow users to specify patterns of interest, ranging from a simple word to a complex regular expression. Pattern matching [5] is the identification of syntactic features that may occur in a given document. However, this model does not consider the structural, sequential, cardinality, and NOT operator. There has been some work [5] about languages that integrate both content and structure of the text, but these systems support IRQLs in a restricted way. In structural languages, [5, 10] a user query can enforce structural constraints on the content. For instance, a user can specify the occurrence of correlated words within a structural element (e.g., title of a chapter).

3. Pattern Specification Language (Psnoop)

In InfoFilter, simple and composite patterns are specified using Psnoop. It supports the following operators and options: cardinality, synonyms, sequence, Boolean operators, structural, wild card, and proximity. Furthermore, any arbitrary complex pattern can be composed using the above operators. In Psnoop, the above operators along with others (such as WITHIN) are used to support information filtering. Psnoop has some similarity with Snoop, [11, 12] an event specification language.
Table 1. Psnoop Pattern Specification Operators

<table>
<thead>
<tr>
<th>Complex Pattern Operators</th>
<th>PSL Operators</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean Operators</td>
<td>OR, NOT, AND</td>
<td>Boolean operations</td>
</tr>
<tr>
<td>Proximity</td>
<td>NEAR</td>
<td>Two words within a certain distance</td>
</tr>
<tr>
<td>Sequential</td>
<td>FOLLOWED BY</td>
<td>Two words occurring in an order within in a certain distance</td>
</tr>
<tr>
<td>Structural</td>
<td>WITHIN</td>
<td>Within document/ paragraph/ sentence</td>
</tr>
<tr>
<td>Cardinality</td>
<td>FREQUENCY</td>
<td>Counting number of occurrences</td>
</tr>
<tr>
<td>Synonyms</td>
<td>SYN</td>
<td>Keyword synonyms</td>
</tr>
<tr>
<td>Wild Card</td>
<td>Regular Expressions</td>
<td>Stream starting with “x”</td>
</tr>
</tbody>
</table>

Patterns ::= Pattern (;Pattern)*
Pattern ::= Expression [Options] | (“ Expression “)” [Options]
Expression ::= Pattern Binary_Op Pattern | Term | Unary
Unary ::= NotExpression | CardExpression
NotExpression ::= NotOp (“ Expression “)”
CardExpression ::= CardOp (“ Expression “)”
CardOp ::= “FREQUENCY /” Frequency
Frequency ::= non-negative integer
Options ::= “WITHIN SENT” | “WITHIN PARA”
Binary_Op ::= “OR” | Proximal
Proximal ::= Proximal_Op [“/” Distance]
Proximal_Op ::= “NEAR” | “FOLLOWED_BY”
Distance ::= non-negative integer
NotOp ::= “NOT”
Term ::= Keyword [“SYNONYM “)”] | Phrase | RegExp
Phrase ::= Keyword Keyword (Keyword)*
Keyword ::= “any quoted string”
RegExp ::= StringLiteral Wildcard StringLiteral | Wildcard StringLiteral
StringLiteral ::= any string
Wildcard =* |?
SYNONYM = “SYN”

This BNF is left associative.

Figure 2. BNF for User Specification

used for the specification of events in a trigger or an event-condition-action (or ECA) rule. The semantics of the operators are, however, different in Psnoop.

A pattern is specified using the language Psnoop and can be classified into simple and composite patterns. A simple pattern is either a word or a simple regular expression (regular expression on a single word). A composite pattern is an expression constructed using simple patterns (or previously constructed composite patterns) and Psnoop operators and options.

A pattern occurrence is defined as an occurrence of a given pattern, which is assumed to be atomic (i.e., happens completely or not at all). Again, pattern occurrences can be classified into a simple pattern occurrence or a complex (or composite) pattern.
occurrence. We assume some pre-defined simple patterns, such as the beginning of a sentence, a paragraph, or a document. In general, some of the pre-defined simple patterns are domain specific and can be defined by the user. For example, structure of a word/latex document is different from the structure of a web page. They will require different pre-defined simple structural patterns. A composite pattern is detected using the simple pattern occurrences according to the semantics associated with the operators in the pattern.

3.1 Simple Patterns Occurrences

As mentioned above, a simple pattern occurrence is an atomic occurrence of a simple or basic pattern. Simple patterns form the basic building block of our pattern specification language. Possible simple patterns that are supported in InfoFilter are:

Single-word: A pattern occurrence is detected if the specified keyword or any of its synonyms (if specified) appear in the text. Synonyms are specified as an option [SYN] (see section 3.2) when the user specifies a keyword.

Multi-word/Phrase: A pattern occurrence is detected if the specified phrase appears in the text. Although a phrase can be specified using Psnoop operators, we treat it as a simple pattern for detecting it efficiently.

Regular expressions: A pattern is detected if the specified regular expression appears in the text.

3.2 Pattern Operators of Psnoop

The operators of Psnoop and the semantics of complex patterns expressed by these operators are defined below:

OR: Disjunction of two simple or complex patterns P₁ and P₂, denoted by (P₁ OR P₂), occurs when either P₁ or P₂ occurs. For example, “information OR filtering” will be detected whenever either one of the keywords occur. Since simultaneous occurrences are not possible in a stream (which is essentially a sequence), exclusive-OR semantics is applied to this operator.

NOT: Non-occurrence of the pattern P₂ in the range formed by P₁ and P₂, denoted by (NOT P₂ [N] P₁), occurs when both P₁ and P₂ occur, irrespective of their order of occurrence. “N” is the maximum distance allowed between the two patterns P₁ and P₂. Default value is the scope of the operator (which can be the entire document), and it refers to the AND operator of the Boolean model. For example, (Information NOT [20] Filtering) will be detected whenever information is followed by retrieval without the word filtering occurring once in between them. The default value of N is 1.

NEAR: Conjunction of two patterns P₁ and P₂, denoted by (P₁ NEAR [N] P₂), occurs when both P₁ and P₂ occur, irrespective of their order of occurrence. “N” is the maximum distance allowed between the two patterns P₁ and P₂. For example, “information FOLLOWED BY [10] Filtering” will be detected whenever the word information is followed by filtering within a distance of 10.

FOLLOWED BY: Sequence of two patterns P₁ and P₂, denoted by (P₁ FOLLOWED BY [N] P₂) occurs when the occurrence of P₁ is followed by the occurrence of P₂. “N” is the maximum distance allowed between the two patterns P₁ and P₂. For example, “information FOLLOWED BY [10] filtering” will be detected whenever the word information is followed by filtering within a distance of 10. If N is not specified, it is assumed to be scope of the operator (can be the entire document).

If the value of N is 1, this indicates that the patterns P₁ and P₂ form a phrase. Although phrases can be expressed using this operator technically, specification of a phrase is allowed in an intuitive manner to improve readability of the specification and its detection efficiently.

WITHIN: Occurrence of a pattern P₂ in the range formed by patterns P₁ and P₃, denoted by (P₂ WITHIN (P₁, P₃)). The pattern is detected each time pattern P₂ occurs in the range defined by patterns P₁ and P₃. For example, “information filtering WITHIN (BEGIN PARA, END PARA)” will be detected whenever the phrase information filtering occurs within a paragraph. When an expression is specified with out an outer WITHIN operator, the default structure (e.g., a document, a web page) is used as the default. This operator is extremely powerful in expressing scopes with in a stream being processed.

FREQUENCY: is a unary operator denoted by (FREQUENCY /[N] P)), where P refers to a simple or composite pattern. A pattern P is detected each time P occurs at least “N” times, where “N” is the minimum number of occurrences specified by the user. For example, “FREQUENCY /10 (information filtering)” will be detected whenever the phrase information filtering occurs 10 times. The first “N” occurrences are considered for this detection as the stream/document is processed sequentially.

SYN: is specified as an option with a simple-word to indicate multiple patterns in a succinct manner. For example, if a simple pattern has 10 synonyms, it is equivalent to specifying 10 simple patterns for detection. The same is true for composite patterns with embedded synonym specification. However, different detection graphs (or nodes) are not constructed for detecting these patterns. They are represented once but detected using an efficient compile time mechanism as described later in this paper.

Using the above operators, users can specify complex patterns. For example, pattern “((bomb FOLLOWED BY ground zero) occurring twice) AND automotive (or its synonyms)” can be specified using the BNF provided in Figure 2 for the Psnoop (Figure 3) as “((FREQUENCY /2 (bomb) FOLLOWED BY “ground zero”)) NEAR “automotive” [SYN]).

4. InfoFilter Architecture

InfoFilter analyzes text streams based on the content and structural information, and notifies the users when their patterns of interest are detected. This system is based on the client/server architecture, wherein a group of clients register with a server to specify patterns of interest. As shown in Figure 3, users submit patterns to the system with Psnoop, and these patterns are validated and processed by the pattern validator and processor respectively. Once processed these patterns are sent to the pattern evaluator where it is used to extract meaningful information from the incoming text streams. Figure 3 shows the various modules of the InfoFilter server: Pattern Detector, Processor, and Evaluator. Other modules shown are external to the system.

The patterns can be associated with different types of streams as shown in Figure 3 (e.g., documents, web pages, and video captions). The InfoFilter uses a separate pattern detector for each type of input and accumulates all patterns for a stream type in that pattern detector. This allows us to detect and exploit
common patterns and create the PDG for them only once. The InfoFilter will continuously monitor multiple types of streaming text, detect simple pattern occurrences and generate notifications for them to the corresponding pattern detector. The pattern detector in turn will detect composite patterns using the semantics of the operators and notifies a user when his/her pattern is detected.

**Figure 3. Architecture of InfoFilter**

4.1 Pattern Validator and Processor
Pattern Validator (PV) accepts the input patterns from the users according to the Psnoop BNF. The PV then checks these patterns for correctness and once validated, the pattern processor (PP) processes them. The PP extracts keywords, phrases, regular expressions and operators, and passes them to the Pattern Evaluator (PE). In the PP, WordNet [6] Database tool is used to determine the synonyms.

WordNet is a lexical database tool. The database is partitioned into nouns, verbs, adjectives and adverbs using the parts of speech. Nouns are organized as a lexical hierarchy of nodes. Each node contains the meaning of a word, or a synset (a list of synonymous words). WordNet is utilized by the pattern processor to extract synonyms of words, whenever it is specified by the user. The synonyms are stored with the corresponding keywords specified by the user. For example, if the user specifies the keyword "bomb" to be detected on a text stream, the pattern processor extracts synonyms such as explosive device, flop, and bust. If any of these synonyms are detected in any incoming text stream, the pattern comprising the keyword "bomb" will be detected.

4.2 Pattern Evaluator
This module consists of the graph generator, stream processor, and notifier. Graph generator uses the extracted keywords, phrases, regular expressions and operators to construct the Pattern Detection Graph (PDG) in the Pattern Detector (PD). For each incoming text stream, pattern evaluator will interact with a PD. PD is a library that will provide the means to define simple and composite patterns. It provides the necessary APIs to generate the patterns. The graph generator constructs a PDG, establishing the relationship among the basic constituent patterns. Leaf nodes of the PDG represent simple patterns, while internal nodes represent composite patterns. The stream processor accepts the incoming text streams as inputs. It parses these input streams to detect the occurrence of simple patterns (keywords, phrases, regular expressions and synonyms). Since these simple patterns correspond to leaf nodes, as soon as they are detected, the stream processor triggers the PDG. Once the pattern detector detects the composite patterns in the PDG, it alerts the notifier, which in turn notifies the user.

5. Implementation
All the InfoFilter server modules, pattern validator, processor, evaluator and detector have been implemented in Java. The pattern validator, a JavaCC parser, accepts user patterns in the form of a linear text in Psnoop BNF, such as “(FREQUENCY / 2 (bomb FOLLOWED BY ground zero)) NEAR automotive [SYN]”. It parses, validates, and tokenizes the patterns based on the syntax of the Psnoop language. The validator outputs the infix notation (Figure 4) of tokens to a queue. Subsequently the pattern processor, a java package, retrieves the infix notation from the queue and converts it to a postfix notation [13] and places it in a stack (Figure 4), which in turn is passed to the pattern evaluator. In the pattern evaluator, user’s patterns are matched against the incoming streams and the users are notified if there is a match. The graph generator, stream processor and the notifier carry out these actions.

![Figure 4. Infix and Postfix Notations](image)

The graph generator uses the postfix notation provided by the pattern processor to construct the PDG in the pattern detector. Pattern detector is a Java library that provides the APIs necessary to construct the detection graphs to detect complex patterns.
The pattern detector APIs support information filtering operators that can be used to construct pattern detection graphs (PDGs). The graph generator reads the postfix notation stored in the stack to construct PDGs using these information filtering operators.

Figure 5. Pattern Detection Graph (PDG)

In the PDG shown in Figure 5, leaf nodes represent simple patterns, and the internal nodes represent composite patterns. In Figure 1, simple patterns $P_1$ and $P_2$ are the leaf nodes, and the composite operator FOLLOWED BY in the internal node represents a complex pattern. A composite pattern node subscribes to its constituent pattern node, and is notified when the constituent patterns are detected. Flow and consumption of pattern occurrences (from incoming text streams) over the nodes in a PDG is crucial, since user patterns are represented using PDGs. Pattern occurrences flow in a bottom-up fashion in the PDG, so that simple pattern occurrences can be used to detect composite (or complex) pattern occurrences. Since the internal nodes of the PDGs form the composite (or complex) pattern, pattern occurrences from constituent nodes should be consumed avoiding duplicate composite pattern occurrences. Since pattern occurrences can flow from any constituent node to an internal node, the order of the pattern occurrences should be considered.

In a FOLLOWED BY operator, the pattern occurrences from the left child should happen before the right child. But, for a NEAR operator the order does not matter. Thus, the constituent node that starts the detection of a composite pattern is known as an initiator of that composite operator and the child that terminates composite pattern detection is known as the terminator. The semantics of each operator is implemented with each node, which ensures that the incoming pattern occurrences are combined properly and forwarded to its subscribers. Note that sharing of leaf nodes as well as sub expressions are possible and are exploited in the InfoFilter. This is done using a uniform name generation convention from the input and using these generated names for detecting same leaf and non-leaf nodes from two different expressions. This reduces the number of nodes and makes the pattern detection efficient.

To detect the pattern “$P_1$ FOLLOWED BY $P_2$” (Figure 5), there should be at least one occurrence of $P_1$ and $P_2$. With the following pattern occurrences: $P_1^1$, $P_1^2$, $P_1^3$, $P_2^1$, $P_2^2$ (where, $P_i^j$ represents the $j^{th}$ occurrence of the $i^{th}$ pattern), pattern $P_1$ has three occurrences ($P_1^1$, $P_1^2$, $P_1^3$), of which one should be paired with an occurrence of pattern $P_2$ to detect the composite pattern “$P_1$ FOLLOWED BY $P_2$”. Since the words or phrases that appear closer to each other are considered to be highly correlated, the pattern occurrence $P_1^j$ is paired with the first occurrence of $P_2$ (i.e., $P_2^1$), detecting the pattern “$P_1$ FOLLOWED BY $P_2$”. This can be achieved in the PDG by the following naïve approach. When $P_1^1$ occurs, it is propagated from the leaf node corresponding to $P_1$ to its parent node. Then, it gets replaced by the occurrence $P_1^2$. Similarly, $P_1^2$ replaces $P_1^1$. When $P_2^1$ occurs, it is propagated from the leaf node $P_2$ to its parent node and gets paired with $P_1^3$. Subsequent occurrence of pattern $P_2$ (i.e., $P_2^2$) gets paired with $P_1^1$, since there is no other initiator between $P_2^1$ and $P_2^2$. Even though, the occurrences $P_1^1$, $P_2^1$, and $P_1^3$ are considered to be the initiators, only the most recent initiator $P_1^3$ is used. Unlike the initiators, both the terminator occurrences $P_2^1$ and $P_2^2$ are used in the detection. As shown above duplicate composite patterns are detected in the naïve approach using the same pattern occurrence $P_1^3$. Thus, pattern detection modes are necessary in order to decide which pattern occurrences should be used to detect a composite pattern.

Figure 6. PDG for user pattern

The recent unique pattern detection mode described below overcomes the limitations of the naïve approach. It allows detection of patterns using a unique pair of an initiator and terminator. In this approach, initiators are removed once they take part in the detection. For example, consider a user pattern “FREQUENCY /2 (“bomb” FOLLOWED BY “battery”)”. In Figure 6, keywords “bomb” and “battery” are simple patterns, the FOLLOWED BY operator in the internal node forms the complex pattern “bomb FOLLOWED BY battery”, and the FREQUENCY operator in the internal node forms the entire complex pattern denoted by “FREQUENCY /2 (bomb FOLLOWED BY battery)”. In Figure 6, the composite pattern node FREQUENCY subscribes to the FOLLOWED BY composite pattern node. Thus, whenever there is a pattern detected at the FOLLOWED BY composite pattern node, it will be propagated to the FREQUENCY node. This subscription/notification mechanism is used to facilitate the detection of patterns and the propagation of associated parameters.
in the graph. The pattern occurrences, used to detect the subpatterns, are propagated up the graph to the subscribing nodes. Reusability of patterns is supported and common subpatterns are represented only once in the PDG, effectively reducing the number of nodes.

Single occurrence of a subpattern can detect the same complex pattern twice in a PDG using the naïve approach. Conversely, in recent unique pattern detection mode, duplicate detections are avoided by removing the recent initiator after it takes part in pattern detection. This may be illustrated with the example “FREQUENCY / 2 (bomb FOLLOWED BY battery)” Figure 6. The pattern detection process is illustrated using the text shown in Figure 7.

In the text (from [14]) shown in Figure 7 the word “bomb” occurs twice and the word “battery” occurs three times. The first occurrence of the word “bomb” (shown as bomb in Figure 7) raises the pattern P1, and the pattern occurrence P1 is propagated to the parent node (i.e., FOLLOWED BY). This pattern occurrence is considered the initiator of the composite pattern FOLLOWED BY since it initiates the pattern detection. Subsequently, the word “battery” (shown as battery in Figure 7) has occurred raising pattern P2, and pattern occurrence P2 is propagated to the parent node (i.e., FOLLOWED BY). Thus, a composite pattern FOLLOWED BY is detected. Pattern occurrence P2 acts as the terminator since it terminates the FOLLOWED BY pattern detection. Since the PDG detects patterns in the naïve approach, it does not remove pattern occurrences from the graph that have already been consumed.

Figure 7. Online article that shows how to make a bomb.

Figure 8. Suffix trie for the words “bomb” and “battery”

When the word “bomb” occurs for the second time (shown as bomb in Figure 7) it detects the composite pattern FOLLOWED BY with the occurrences of bomb and battery (i.e., P12 and P23). Here the occurrence of word “battery” triggers the detection of two composite patterns. This should not be allowed since the unique occurrence of keywords that appear adjacent to each other is favorable, as it provides correct results. Thus, pattern detection modes are necessary to remove the consumed patterns and to detect accurate patterns. The pattern detection modes are based on the notion of pairing an initiator with a terminator. An initiator is the pattern occurrence that starts the detection of a composite pattern, and a terminator is the pattern occurrence that ends the detection of a composite pattern. With the pattern detection modes, the recent occurrence of an initiator for a pattern under detection is used. The recent initiator is paired with the recent terminator forming a unique pair. In the above example for the occurrences of patterns “bomb” “battery”, “bomb”; “battery”; “battery”, the following pairs are detected: “bomb” FOLLOWED BY “battery”; “bomb” FOLLOWED BY “battery”. Occurrence of “battery” does not detect any composite pattern since there are no occurrences of the word “bomb”.

The detection modes can be nicely encapsulated into the algorithm that deals with the semantics of each operator. This keeps the system manageable and makes it easy to implement seemingly complex pattern detection modes.

The stream processor extracts keywords, phrases, synonyms and regular expressions from the postfix notation and stores them in a suffix trie (a trie constructed over all the suffixes of the text) so that these can be efficiently matched against the incoming text streams. Suffix tries are characterized as space efficient and have less search time for small text. [5, 15, 16]

In the suffix trie, each position in the text is considered to be a text suffix. The start and end position of the suffixes, represented by the subtitles, are stored in the nodes of the trie. [17] Suffix trie for the keywords “bomb” and “battery” are shown in Figure 8. In this suffix trie, the branches connected to the root node are labeled in alphabetical order. For instance, the branch that is labeled “b” is connected to the node that contains pointers and indices of the suffixes “battery”, “omb”, and “$”.

To look for the patterns in the input stream, stream processor matches the tokens, generated by the parsing process, with the patterns stored in the suffix trie. Consider the text in Figure 7, the stream processor processes the first line of the text and generates the following tokens, “bomb”, “from”, “a”, and “battery”, respectively. To search for “bomb” in the suffix trie, the token “bomb” gets numbered from left to right starting at 1. The search starts with the root node and it follows the branch that is labeled with “b”. Then, it compares the remaining indices with the pattern indices. If the indices are equal, it compares the second character of the token “o”, if there is a match it continues comparing the remaining characters in the sub-branch that starts with “o”.

Bomb from a Battery

Alright, so you wanna make an inexpensive pressure bomb without messing with drano and sulfuric acid and all those other dangerous chemicals (which can burn the crap out of your skin if your not careful.) All you need is a battery, a 2 liter plastic bottle, and hydrogen peroxide (3% by solution is fine), a hammer and a sharp nail. The size of the battery doesn’t matter, although I don’t recommend some of the D because the new ones aren’t dry cell anymore, and the concentrated sulfuric acid inside can burn you.
Hence, “o” matches with “o”, “m” with “m”, and so forth, until the label and the token are consumed.

Thus the match is considered to be successful, since the node has a pointer that corresponds to the word “bomb”. Finally, the stream processor triggers the PDG mapped to the word “bomb”. In addition to generating the notification, when simple pattern occurrences are detected, the stream processor is also responsible for detecting structural pre-defined simple patterns such as begin-sentence and end-sentence. These detections will also trigger leaf nodes in the PDG. Once the PDG detects the complex pattern as explained in the pattern detection modes, the notifier is notified. The notifier then alerts the user about the pattern occurrence.

6. Conclusion and Future work

In this paper, we have presented the InfoFilter a content-based system for filtering text streams. InfoFilter has been developed with an intent to support expressive user patterns using Psnoop and to provide on-the-fly filtering and notification on multiple streams. InfoFilter can be used in many contexts. We are currently preparing for its release to general public. Psnoop, proposed in this paper, overcomes the limitations of the Information Retrieval Query Languages currently used for specifying and detecting user patterns.

InfoFilter can be extended in a number of ways. Psnoop can be further extended by allowing complex regular expressions, and synonyms for phrases. Higher level specification of patterns that can be converted into Psnoop is another direction. We plan on linking InfoFilter with the web monitoring system (WebVigiL) to filter web contents in a selective manner. In order to process XML streams, only the stream processor and pattern validator modules of the InfoFilter architecture need to be extended to handle the structure of XML elements. The system can also be extended to search for patterns in an entire web and for the generation of web ontology based on the patterns detected in web pages in a web server.

7. References
