InfoFilter: A System for Expressive Pattern Specification and Detection Over Text Streams

Laali Elkhalifa, Raman Adaikkalavan and Sharma Chakravarthy
IT Laboratory and Department of Computer Science & Engineering
The University of Texas At Arlington, Arlington, TX 76019
{elkhalif, adaikkal, sharma}@cse.uta.edu

ABSTRACT
Information filtering includes monitoring text streams to detect patterns that are more complex than those handled by search engines. Text stream monitoring and pattern detection have far reaching applications such as tracking information flow among terrorist outfits, web parental control, and business intelligence. Pattern characterization requirements of applications entail an expressive language for specifying patterns than what is currently provided by Information Retrieval Query Languages (IRQLs) and current information filtering systems. Pattern specification alone does not suffice, as detecting these complex patterns is equally important in order to use these systems for real-world applications.

InfoFilter, a content-based information filtering system, presented in this paper, allows users to specify complex patterns and detects these patterns in incoming text streams from various sources such as news feed, emails, web pages and caption text from streaming videos. Complex patterns such as combinations of sequential, structural patterns, wild cards, word frequencies, proximity, Boolean operators and synonyms are formulated using the expressive pattern specification language, PSL, proposed in this paper.

Once specified, these complex patterns are detected using a data flow paradigm over Pattern Detection Graphs (PDGs).

1 INTRODUCTION
Recent advancements in computing have led to a digitized world with an ever increasing data available online. Users often find themselves swamped with colossal amount of information while retrieving task relevant data. Consider a news analyst who subscribes to an online news feed in order to perform an analysis on an interesting topic, a sales representative who receives email orders for different products, a federal agent who monitors terrorist information correspondence, and so forth. In all of the above scenarios, the ability to deal with information overloading is an important issue and better approaches for filtering information are needed.

Information filtering is the process of extracting relevant or useful portions of information/documents from large data repositories or continuous streams of textual data based on relatively static 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 filtered from the incoming text streams. 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 and meaningful patterns using an expressive pattern specification language.

Based on the similarity between information filtering and information retrieval [1, 2, 3], most of the existing filtering tools such as personalized information filtering systems use Information Retrieval Query Languages (IRQLs) [1] for user query specification. Thus, the work in this paper has been motivated by several observations on the IRQLs and current query languages and the amount of expressiveness or flexibility desirable in user pattern specification.

As observed from the characteristics of the query languages proposed in the literature, they support single-word, Boolean, context, natural language, pattern matching and structural queries, and their compositions in a very restricted manner. Typically, a simple user 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 complex pattern like "information" AND "filtering" can be specified and detected. On the other hand, in context queries, a user can only specify a set of keywords that co-occur with another keyword. They can be used to detect patterns such as "information" NEAR "filtering" WITHIN 100 WORDS (i.e., words "information" and "filtering" occurring within a distance of 100 words irrespective of the order). However, complex compositions of the above mentioned queries are not supported, thus limiting pattern specification.

Consider a real world example where a federal agent is tracking terrorist-related information streaming from various resources. He/she is interested in the occurrence of the word “bomb” followed by the word “ground zero” occurring twice, along with the word “automotive” or its synonyms (i.e., ("bomb" FOLLOWED BY "ground zero") occurring twice) AND “automotive” (or its synonyms). This pattern contains keywords, sequence (FOLLOWED BY), phrase, frequency, synonyms, and a Boolean operator. This pattern cannot be expressed using current query specifications as they do not support the following: 1) quantification of multiple occurrences (or frequency) of patterns and complex compositions, and 2) a user cannot include synonyms in the pattern, and is required to explicitly list all the synonyms as separate patterns. Thus, current query languages are quite restrictive in their expressive power and need to be extended and generalized to address the specification of meaningful complex user patterns. Pattern specification alone
does not suffice as detecting these complex patterns is equally important in order to use these systems for real-world applications.

The aim of this paper is to overcome the limitations mentioned above. In this paper, we propose an information filtering system that encompasses an expressive pattern specification language (PSL) to specify complex patterns and a detection mechanism in the form of a PDG (Pattern Detection Graph) for detecting these patterns. Figure 1 outlines the InfoFilter System handling various incoming text streams, creation of user patterns using PSL, synonym extraction using WordNet [4] database tool and user notifications. In InfoFilter, patterns are shared to reduce the pattern detection cost by merging common subgraphs in a PDG.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 introduces the Pattern Specification Language (PSL) and briefly discusses the various pattern operators and their semantics. Section 4 depicts the architecture of InfoFilter and discusses various components of InfoFilter, and section 5 illustrates the pattern detection modes and the pattern detection using PDGs. Section 6 discusses the implementation of InfoFilter. Finally, section 7 has the conclusions and future work.

2 RELATED WORK

Commercially and freely available information filtering systems such as SIFT [5], InfoScope [6], Glimpse [7], Igrep [8] were developed to provide solutions to assist users in extracting relevant information. Various techniques have been applied ranging from simple approaches such as rule-based approaches to complex ones such as machine learning algorithms and probabilistic approaches.

SIFT [5, 9] designed at Stanford University, is a content-based filtering system. Boolean queries and Vector Space Model [3] are used to construct user profiles, allowing users to specify keywords that are to be included and those that are to be excluded, when filtering USENET articles. However, 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 body. It does not support proximity, regular expressions, frequency, structural, and sequence operators.

GLIMPSE (Global Implicit Search) [7], another content-based filtering system utilizes indexing and query schema for retrieving files. It supports Boolean queries and approximate matching such as regular expressions. However, it does not support proximity, frequency, structural, sequence, and compositions of queries.

Igrep [8] is an approximate matching tool for large data collections. It accepts words, phrases, and set of characters such as wild cards, ranges, etc. Regular expressions are supported to provide approximate matching. Again, it does not support frequency, structural, sequence, and not all the compositions of queries.

Inquiry [10], is a ranked retrieval system, and Lemur [11], a toolkit based on Inquery is used for Language Modeling and Information Retrieval. Both the above are typical information retrieval (IR) systems and are not information filtering systems, that can filter patterns over text streams. Even though Inquery’s language supports structural queries it does not have the capability to search within a paragraph, or within a range of words, which is possible using InfoFilter’s WITHIN operator. It also does not support frequency.

Search engines [1, 9] such as Google, Altavista are used to retrieve documents or Web pages online. Fundamentally, most search engines utilize traditional Information Retrieval techniques such as variations of the Vector Space Model [3] and ranking algorithms. They support user queries comprising list of keywords, Boolean operators, phrase search, proximity search and wild cards. Nevertheless, they do not support complex compositions of those queries. The mechanism of answering user queries in search engines are different than the typical IR systems. Search engines model the Web as a database and all the user queries are answered using the stored web pages.

In summary, both search engines and current information filtering systems do not allow arbitrary compositions of various types of queries.

3 USER SPECIFICATION

In InfoFilter, users can specify simple and complex patterns using the Pattern Specification Language, PSL. It supports the following operators and options: frequency, synonyms, sequence, Boolean operators, structural, wild card, and proximity. Furthermore, any arbitrary complex pattern can be composed using the above operators. Some of the operators in PSL have some similarities with event specification languages [12] used for the specification of events. Even though some of the operators are similar, semantics of pattern operators are different as it includes the notion of proximity, which is crucial in information filtering. In addition, PSL supports pattern operators such as regular expressions, frequency and synonyms. Also, PSL allows composition of all the above operators for specifying complex patterns.

3.1 Pattern Types

In PSL, a pattern $P$ is formally represented as $P^i_j$, where $i$ is the pattern identifier and $j$ is the instance of the pattern identifier. A pattern $P$ is a function that maps from the offset interval domain onto the boolean values, “True/False” corresponding to the occurrence or non-occurrence of the pattern. $O_1$ is the start offset, and $O_2$ is the end offset of the pattern, where offset is the position of the word relative to the beginning of the text stream. For example, in the phrase “user pattern”, if “user” occurs at offset 50 then “pattern” occurs at 51, and $O_1$ is 50 and $O_2$ is 51.

According to the semantics of PSL, patterns are classified as:

Simple patterns: These are the basic building blocks and can be either System-defined (i.e., built into the system), or User-defined. System-defined patterns are pre-defined and they correspond to the structural elements present in text streams, such as the beginning of a sentence, a paragraph, or a document/stream. For example, two system-defined patterns BeginPara and EndPara are used to define the beginning and end of a paragraph. On the other hand, possible user-defined patterns include a single word or any of its synonyms (e.g., “filter”), multi-word or phrases (e.g., “information filtering system”), or simple regular expressions (e.g., “filter*”).

Complex patterns: These are composed of simple patterns, complex patterns, pattern operators and options. PSL provides a comprehensive set of pattern operators and they are: Boolean (OR, NOT, NEAR), sequence (FOLLOWED BY), structure (WITHIN), frequency (FREQUENCY), proximity (NEAR/N, FOLLOWED BY/N) and the option synonym (SYN).
3.2 Operators for Pattern Specification

Semantics of PSL operators and options are explained below (formal definitions for these operators and options are not provided in this paper due to lack of space).

OR: Disjunction of two simple or complex patterns $P_1$ and $P_2$, denoted by $(P_1 \text{ OR } P_2)$, occurs when either $P_1$ or $P_2$ occurs. For example, “information” OR “filtering” will be detected whenever either one of the keywords occurs. Since simultaneous occurrences of the same patterns are not possible in a stream (essentially a sequence), exclusive OR semantics is used.

NOT: Non-occurrence of the simple or complex pattern $P_2$ in the range formed by the end offset of $P_1$ and the start offset of $P_2$, where $P_1$ and $P_2$ can also be simple or complex patterns, is denoted by $(\text{NOT } /F)(P_1)(P_2)$, “F” indicates the minimum number of occurrences and its default value is 1. For example, NOT (“filtering”) (“information”, “retrieval”) will be detected whenever “information” is followed by “retrieval” without the word “filtering” occurring at least once in between them.

NEAR: Conjunction of two simple or complex patterns $P_1$ and $P_2$, denoted by $(P_1 \text{ NEAR } /D)(P_2)$, occurs when both $P_1$ and $P_2$ occur, irrespective of their order of occurrence. “D” is the maximum distance allowed between the patterns $P_1$ and $P_2$. Default value of “D” is the scope of the operator (which can be the entire document), and it refers to the AND operator of the Boolean model. The minimum value of $D$ is 1. For example, “information” NEAR/10 “filtering” will be detected whenever both these words co-occur within a distance of 10.

FOLLOWED BY: Sequence of two simple or complex patterns $P_1$ and $P_2$, denoted by $(P_1 \text{ FOLLOWED BY } /D)(P_2)$, occurs when the occurrence of $P_1$ is followed by the occurrence of $P_2$. The end offset of $P_2$ is less than the start offset of $P_1$; that is, the occurrence interval of $P_1$ and $P_2$ should not overlap. “D” is the maximum distance allowed between the two patterns $P_1$ and $P_2$. If “D” is not specified, the distance is bounded by the scope of the operator (can be the entire document). If the value of “D” is 1 (minimum value), this indicates that the patterns $P_1$ and $P_2$ form a phrase. For example, “information” FOLLOWED BY /10 “filtering” will be detected whenever the word “information” precedes “filtering” within a distance of 10 words.

WITHIN: Occurrence of a simple or complex pattern $P_2$ in the range formed by the end offset of the pattern $P_1$ and the start offset of $P_2$, denoted by $(P_2 \text{ WITHIN } /D)(P_1)$. The pattern is detected each time pattern $P_2$ occurs in the range defined by patterns $P_1$ and $P_2$. For example, “information filtering” WITHIN (BeginPara, EndPara) will be detected whenever the phrase “information filtering” occurs within a paragraph. When an expression is specified without a system-defined pattern, the default structure (e.g., a document) is used as the default. This operator is crucial while expressing the scope of the stream being processed.

FREQUENCY: Multiple occurrences of a simple or complex pattern that exceed or equal to $F$, denoted by $(\text{FREQUENCY } /F)(P)$. A pattern $P$ is detected each time $P$ occurs at least $F$ times, where “F” is the minimum number of occurrences specified by the user. The default value of $F$ is 1, which is the minimum value. All the occurrences that are used for detection should be disjoint (i.e., the end offset of each pattern occurrence should precede the start offset of the subsequent pattern occurrence). The same set of occurrences is not used for detecting multiple patterns. For example, FREQUENCY/10 (“information filtering”) will be detected whenever the phrase “information filtering” occurs at least 10 times. Frequency can be applied to any pattern expression.

SYN: This is an option and is specified along with a single-word pattern (currently), denoted by $(P \text{ SYM})$, to indicate multiple single-word patterns that have the same meaning, in a succinct manner. In PSL, specifying a single-word pattern with SYM option is equivalent to specifying $N$ simple patterns that carry the same meaning (synonyms) as the original pattern. For example, if you specify the word “bomb”[SYN] is equivalent to specifying “bomb” OR “explosives device” OR “weaponry” OR “arms” OR “implements of war” OR “weapons system” OR “munition”...

4 INFOFILTER

InfoFilter analyzes text streams based on the content and structural information, and notifies users when their patterns of interest are detected. The system is based on the client/server architecture, wherein a group of clients register with a server to specify patterns of interest. Figure 2 shows various modules of the InfoFilter server: Pattern Validator, Graph Generator, Stream Processor, Pattern Detector, Notifier and other external components. Patterns can be associated with different types of text streams (e.g., documents, web pages, and video captions). User patterns are handled by certain modules and incoming streams are handled by other modules. Pattern flow and its corresponding modules are shown in Figure 3, and stream flow and its corresponding modules are shown in Figure 4, and both are explained below.

Pattern flow in the InfoFilter is illustrated in Figure 3. On the client side, users submit patterns using PSL to the InfoFilter server. In the InfoFilter server, these patterns are validated and processed upon submission by the pattern validator. Once processed, these patterns are sent to the graph generator. The graph generator constructs the PDGs corresponding to the patterns in the pattern detector, and interacts with the WordNet database tool to extract synonyms of single words if specified. It also stores the keywords, phrases and regular expressions, embedded in these patterns in a shared suffix trie.

Figure 4 depicts the stream flow in InfoFilter. As shown, stream processor interacts with the shared suffix trie for pattern matching.
5 PATTERN DETECTION

User patterns are required to be detected as the incoming text streams flow into the system. For simple patterns, detecting the pattern occurrence is straightforward. For example, the keyword “Language” is detected just by simple pattern matching. However, for complex patterns composed of complex sub-patterns, the detection should conform to the semantics of PSL pattern operators where the order of pattern occurrence is significant (e.g., FOLLOWED BY operator). For example, in order to detect a complex pattern “Information” FOLLOWED BY/10 “Filtering”, the flow of pattern occurrences should be preserved. Thus, a data flow paradigm in the form of a PDG is used to detect the patterns according to the operator semantics and to maintain the flow of pattern occurrences. In this section we address how patterns are detected using pattern detection graphs and modes. How patterns are detected effectively using shared approaches are explained in [13].

5.1 Pattern Detection Graphs (PDGs)

For each pattern or sub-pattern, a corresponding PDG is constructed. In a PDG (Figure 5), leaf nodes represent simple patterns and internal nodes represent PSL pattern operators. For example, Figure 6(b) shows the PDG corresponding to the complex pattern “Query” FOLLOWED BY “Language”. Both the leaf and internal nodes store information about their parent nodes in a subscriber list. Typically, during the construction of a PDG, the parent node subscribes to its child nodes by placing its reference in the subscriber list of the child nodes. In addition to the subscriber list, the leaf node contains the name of the simple pattern it corresponds to. For complex patterns, the internal node contains the name of the complex pattern, references to the nodes of the sub-patterns and their parameters such as the offset of the pattern occurrence and reference to the text stream in which it occurs.

The pattern occurrences flow through the PDG in a bottom-up manner. Once a simple pattern occurrence is detected, the corresponding leaf node propagates it with the associated parameters to the parent node as leaf nodes do not have storage capabilities. For example, in Figure 6(b) when an occurrence of the simple pattern “Query” is detected, it is propagated to the FOLLOWED BY node. Each parent node allocates a space for storing the pattern occurrences that belong to its child nodes. Similarly, when a sub-pattern occurrence is detected, the corresponding internal node propagates it to the parent nodes, using the subscriber list.

5.2 Pattern Detection Modes

According to the characteristics of the PDGs, the detection of a complex pattern requires the detection of its sub-patterns. The sub-pattern that starts the detection of a complex pattern is termed the “initiating sub-pattern/pattern”. Similarly, the sub-pattern that ends the detection of a complex pattern is termed the “terminating sub-pattern/pattern”. In the most general case (termed unrestricted mode), the complex pattern occurrences are detected using all the occurrences of the sub-patterns. This may generate a large number of pattern occurrences which can contain duplicates and may not be meaningful.

Let us take a sample pattern “Query” FOLLOWED BY “Languages”, represented as Q FOLLOWED BY L. Simple pattern occurrences \{Q^1 (P^1), Q^2 (P^2), L^1 (P^3), L^2 (P^4)\} are shown in Figure 6(a), where P^i_j indicates the pattern occurrence and offset. For these simple pattern occurrences, four combinations of complex pattern occurrences are generated and they are:

\{Q^1, L^1\}, \{Q^2, L^1\}, \{Q^1, L^2\}, \{Q^2, L^2\}.

This poses a question as to which pattern occurrences can participate in the detection of the complex pattern (i.e., which occurrence
of \( Q^1 \) should be paired with \( L^1 \). As singularity (i.e., a single pattern cannot detect two different patterns, thus creating duplicates) and proximity (i.e., words that co-occur near each other are considered highly correlated providing a semantic meaning) of pattern occurrences play a significant role in information filtering domain, Proximal–Unique detection mode was developed to provide an accurate pattern detection.

**Proximal-Unique**: Only the initiating sub-pattern/pattern occurrence that is closest to the terminating sub-pattern/pattern occurrence is used for pattern detection, ensuring the proximity property. From the above example, pattern occurrence \( Q^1 (P^1) \) is discarded as \( Q^2 (P^2) \) will be the closest to the terminating sub-pattern/pattern occurrence \( L^1 (P^2) \). On the other hand, the initiating sub-pattern/pattern occurrence is discarded immediately after it has been used in any pattern detection, ensuring that no duplicate pattern occurrences are generated. In other words, this mode entails that an occurrence of a sub-pattern can participate only once in detecting a complex pattern. Therefore, once the pattern occurrence \( Q^2 \) is paired with \( L^1 \) it cannot be paired with \( L^2 \). In this mode only the pattern occurrences \( \{Q^2, L^1\} \) are used to detect the complex pattern shown in Figure 6(a). Since this detection mode emphasizes the proximity and uniqueness of pattern occurrences, it is termed as Proximal-Unique.

6 IMPLEMENTATION

The InfoFilter server has been implemented as a stand-alone Java application. It consists of pattern validator, graph generator, stream processor, pattern detector, notifier, and suffix trie modules. The pattern validator, a JavaCC parser, accepts user patterns in the form of a linear text using PSL BNF (see [13]), such as (“Information FOLLOWED BY ”Filtering”). It parses, validates, and tokenizes the patterns based on the syntax of the PSL language. Then, it places the tokens into a stack in postfix order and passes it to the graph generator. The graph generator reads the postfix notation and constructs the PDG. It interacts with the WordNet database tool to extract synonyms of words. It uses JavaWordNet Library (JWNL) [14] to provide this interaction. The extracted words, synonyms, phrases, and regular expressions are stored in a shared suffix trie. Suffix tries are constructed over all the suffixes of the text. They are characterized as space efficient and have less search time for small text [1, 15, 16]. Pattern detector is a Java library that provides the APIs necessary to construct the PDGs that detect complex patterns.

Stream processor is used to handle various incoming text streams and is implemented using several tokenizers. For handling unstructured text, a parser has been implemented using the BreakIterator Class in Java. The BreakIterator Class can detect text boundaries such as word boundaries, sentence boundaries, and so forth. The Java Mail API [17] has been used to parse emails. The parser should extract the textual information present in the emails, which is then tokenized using the BreakIterator. All the tokens generated by these parsers are then matched against the extracted keywords, phrases, and regular expressions stored in the suffix tries. If there is a match, the stream processor notifies the pattern detector, which alerts the notifier. The notifier is implemented using Java Mail APIs to send messages to users subscribed to the system.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we have presented the InfoFilter system, a content-based system for filtering text streams. InfoFilter has been developed with an intent to support expressive user patterns using PSL and to provide filtering on streams and notification. We have implemented the InfoFilter server that consists of pattern validator, pattern detector, graph generator, stream processor, notifier, and suffix trie modules. PSL, proposed in this paper, with its expressiveness and well-defined semantics, overcomes the limitations of the current information filtering systems used for specifying and detecting user patterns. It provides a complete set of pattern operators and options such as frequency, synonyms, followed by, Boolean operators, structural, wild card, and proximity.

We are currently extending the InfoFilter in a number of ways. PSL is being extended to support complex regular expressions, and synonyms for phrases. Higher-level specification of patterns that can be converted into PSL is another direction. InfoFilter can be linked with the web monitoring systems to filter web contents in a selective manner. The system can also be extended to search for patterns in the entire web and in the generation of web ontology based on the patterns detected in web pages in a web server. Patterns that need to be detected over other types of streams such as XML can be supported by modifying the stream processor and pattern validator modules of the InfoFilter architecture. Handling partial pattern matching can be explored further. In the long term, this work can be extended to incorporate adaptive filtering, and discovering patterns that can be of interest to the user.

8 ACKNOWLEDGMENTS

This work was supported, in part, by NSF grants (IIS-0123730 and IIS-0326505).

9 REFERENCES