StreamWorks - A system for Dynamic Graph Search

Sutanay Choudhury
Pacific Northwest National
Laboratory, USA
sutanay.choudhury@pnnl.gov

Lawrence Holder
Washington State University,
USA
holder@wsu.edu

George Chin
Pacific Northwest National
Laboratory, USA
george.chin@pnnl.gov

Abhik Ray
Washington State University,
USA
abhik.ray@wsu.edu

Sherman Beus
Pacific Northwest National
Laboratory, USA
sherman.beus@pnnl.gov

John Feo
Pacific Northwest National
Laboratory, USA
john.feo@pnnl.gov

ABSTRACT

Acting on time-critical events by processing ever growing social media, news or cyber data streams is a major technical challenge. Many of these data sources can be modeled as multi-relational graphs. Mining and searching for subgraph patterns in a continuous setting requires an efficient approach to incremental graph search. The goal of our work is to enable real-time search capabilities for graph databases. This demonstration will present a dynamic graph query system that leverages the structural and semantic characteristics of the underlying multi-relational graph.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Query processing

Keywords

Continuous Queries; Dynamic Graph Search; Subgraph Matching

1. INTRODUCTION

Social networks, social media websites and mainstream news media are driving an exponential growth in online content and network traffic. This information barrage presents both a formidable challenge and an opportunity to applications that thrive on situational awareness. Domains such as emergency response, cyber security, intelligence and finance has many applications that continuously monitor the data stream to look for specific events. Timeliness of the detection carries paramount importance for such applications. The applications derive their competitive edge from fast detection as late detection may not have much value due to incurred damage to resources. Our work is motivated by queries that look for rare events, have a time constraint on the time to discovery and never need a bulk retrieval of historic data due to their monitoring nature.

The field of relational databases studied the topic of continuous queries to address applications with precisely the above characteristics. A continuous query (CQ) system is defined as one where a query logically runs continuously over time as opposed to being executed intermittently [2–4]. Many of the prominent news,
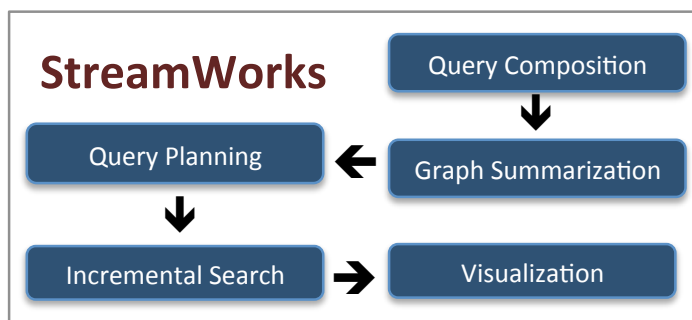
Figure 1: **Various components for graph mining and search.**

social media or cyber data streams can be represented as multi-relational graphs. Following the sprit of CQ systems, our work can be viewed as continuously searching a temporally evolving (henceforth referred as dynamic) graph for graph based patterns representing various events of interest.

Our proposed demonstration will showcase StreamWorks (Fig. 1) - an analytics framework for dynamic graphs. With StreamWorks, a user can register graph queries to find events as they emerge in the data graph. The novelty of StreamWorks lies in its incremental graph search algorithm based on a query decomposition approach. The registered queries are decomposed into sub-patterns using a novel data structure called the SJ-Tree [6] that systematically tracks the evolution of matches in the underlying graph. The query decomposition is performed by utilizing statistics and summaries about the data graph such as degree distribution, vertex and edge type distribution and multi-retlational triad distribution.

1.1 Demonstration features

We will present an interface to compose and execute graph queries, and query planning. Further, we will provide visualization of the evolving graph, results from graph queries, and relevant statistics.

2. BACKGROUND AND RELATED WORK

2.1 Problem Statement

Our theoretical contribution is the development of an incremental subgraph isomorphism algorithm for dynamic graphs [6]. Given a pattern or query graph (henceforth described as query graph) G_q and a larger input graph (henceforth described as the data graph) G_d, an isomorphism of G_q in G_d is defined as the matching that involves a one-to-one correspondence between the vertices of a sub-

graph of G_d and vertices of G_q such that all vertex adjacencies are preserved.

Every edge in a dynamic graph has a timestamp associated with it and therefore, for any subgraph g of a dynamic graph we can define a time interval $\tau(g)$ which is equal to the interval between the earliest and latest edge belonging to g. Given a dynamic multi-relational graph G_d, a query graph G_q and a time window t_W, we report whenever a subgraph g_d that is isomorphic to G_q appears in G_d such that $\tau(g_d) < t_W$. The isomorphic subgraphs are also referred to as *matches* in the subsequent discussions. If $M(G_d^k)$ is the cumulative set of all matches discovered until time step k and E_{k+1} is the set of edges that arrive at time step $k+1$, we present an algorithm to compute a function $f(G_d, G_q, E_{k+1})$ which returns the incremental set of matches that result from updating G_d with E_{k+1} and is equal to $M(G_d^{k+1}) - M(G_d^k)$.

2.2 Related Work

Investigation of subgraph isomorphism for dynamic graphs did not receive much attention until recently. It introduces new algorithmic challenges because we can-not afford to index a dynamic graph frequently enough for applications with real-time constraints. In fact this is a problem with searches on large static graphs as well [8]. There are two alternatives in that direction. We can search for a pattern repeatedly or we can adopt an incremental approach. The work by Fan et al. [7] presents incremental algorithms for graph pattern matching. However, their solution to subgraph isomorphism is based on the repeated search strategy. Chen et al. [5] proposed a feature structure called the *node-neighbor tree* to search multiple graph streams using a vector space approach. They relax the exact match requirement and require significant pre-processing on the graph stream. Our work is distinguished by its focus on temporal queries and handling of partial matches as they are tracked over time using a novel data structure. There are strong parallels between our algorithm and the very recent work by Sun et al. [8], where they implement a query-decomposition based algorithm for searching a large static graph in a distributed environment. Our work is distinguished by the focus on continuous queries that involves maintenance of partial matches as driven by the query decomposition structure, and optimizations for real-time query processing.

3. INCREMENTAL QUERY PROCESSING

3.1 Our Approach

A simplistic approach to solving this problem would be to check, for every edge update, if that edge matches one in the query graph. Once an edge is considered as a matching candidate, the next step is to consider different combinations of matches it can participate in. While intuitively simple, this approach falls prey to combinatorial explosion very quickly. Our objective is to introduce an approach that guides the search process to look for specific subgraphs of the query graph and follow specific transitions from small to larger matches. Following are the main intuitions that drive this approach,

1. Instead of looking for a match with the entire graph or just any edge of the query graph, partition the query graph into smaller subgraphs and search for them.
2. Track the matches with individual subgraphs and combine them to produce progressively larger matches.
3. Define a *join order* in which the individual matching subgraphs will be combined. Do not look for every possible way to combine the matching subgraphs.
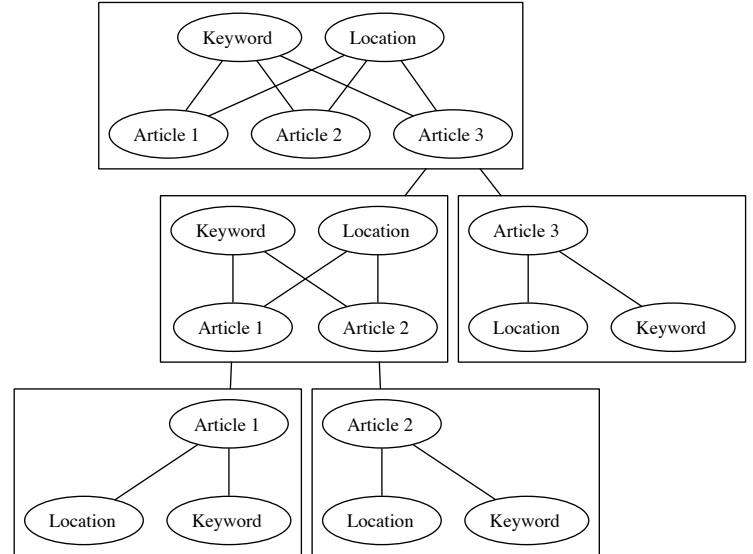

Figure 2: Illustration of query decomposition in SJ-Tree. The graph shown in the root node represents a query to find three articles or posts with a common keyword and location.

Although the current work is completely focused on temporal queries, the graph decomposition approach is suited for a broader class of applications and queries. The key aspect here is to search for substructures without incurring too much cost. Even if some subgraphs of the query graph are matched in the data, we will not attempt to assemble the matches together without following the join order. Thus, if there are substructures that are too frequent, joining them and producing larger partial matches will be too expensive without a stronger guarantee of finding a complete match. On the other hand, if there is a substructure in the query that is rare or indicates high selectivity, we should start assembling the partial matches together only after that substructure is matched.

3.2 Subgraph Join Tree (SJ-Tree)

We introduce a tree structure called *Subgraph Join Tree (SJ-Tree)* that supports the above intuitions for implementing a search and join order based on selectivity of substructures of the query graph. Fig. 2 shows an example decomposition of a query graph.

DEFINITION 4.1.1 A SJ-Tree T is defined as a binary tree comprised of the node set N_T. Each $n \in N_T$ corresponds to a subgraph of the query graph G_q. Let's assume V_{SG} is the set of corresponding subgraphs and $|V_{SG}| = |N_T|$. Additional properties of the SJ-Tree are defined below.

PROPERTY 1. The subgraph corresponding to the root of the SJ-Tree is isomorphic to the query graph. Thus, for $n_r = root\{T\}$, $V_{SG}\{n_r\} \equiv G_q$.

PROPERTY 2. The subgraph corresponding to any internal node of T is isomorphic to the output of the join operation between the subgraphs corresponding to its children. If n_l and n_r are the left and right child of n, then $V_{SG}\{n\} = V_{SG}\{n_l\} \bowtie V_{SG}\{n_r\}$. Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the join operation is defined as $G_3 = G_1 \bowtie G_2$, such that $G_3 = (V_3, E_3)$ where $V_3 = V_1 \cup V_2$ and $E_3 = E_1 \cup E_2$.

PROPERTY 3. Each node in the SJ-Tree maintains a set of matching subgraphs. We define a function $matches(n)$ that for any node $n \in N_T$, returns a set of subgraphs of the data graph. If $M = matches(n)$, then $\forall G_m \in M, G_m \equiv V_{SG}\{n\}$.

PROPERTY 4. Each internal node n in the SJ-Tree maintains a subgraph, CUT-SUBGRAPH(n) that equals the *intersection* of the query subgraphs of its child nodes.

4. SYSTEM OVERVIEW

4.1 Query Planning

With the subgraph join-tree data structure in mind, the next task is to automatically decompose a query graph G_q and create a subgraph join tree based on the decomposition. Broadly our aim is to decompose the query graph into a number of smaller graphs, which we refer to as *search primitives*, and perform local searches for these primitives. We use the term *local search* to refer to a subgraph search performed in the neighborhood of an edge in the data graph for a *small query subgraph*. The primitives are restricted to small and "selective" query subgraphs to keep the local search efficient. An important goal of the decomposition process is to push the most selective subgraph at the lowest level in the subgraph join-tree to reduce the number of partial matches.

4.2 Query Execution

Our proposed subgraph matching algorithm contains two primary tasks. First, for every incoming edge we perform a local search to detect a match with the smallest subgraphs associated with the leaves of the SJ-Tree. When a match is found with the subgraph corresponding to the leaf node of the SJ-Tree, we initialize a match structure and insert it into the collection maintained at that leaf node. Upon insertion of a match into a leaf node we check to see if it can be combined with any matches that are contained in the collection maintained at its sibling node. A successful combination of matching subgraphs between the leaf or intermediate node and its sibling node leads to the insertion of a larger match at the respective parent node. This process is repeated as long as larger matching subgraphs can be produced by moving up in the SJ-Tree. A complete match is found when two matches belonging to the children of the root node are combined successfully.

4.3 Summarization

Summarization involves collecting summary statistics about the data graph to use for query planning. We collect three different types of information 1) degree distribution 2) distribution of vertex and edge types, 3) the frequency distribution of multi-relational triad structures. Incorporation of triad statistics into the query decomposition process is a work in progress at the time of this writing. Continuously collecting the statistics information from the data stream and updating the query decomposition and search strategy remains an area for future work.

5. TARGET APPLICATIONS

We focus on two major application domains: cyber-security and news/social media monitoring. The following subsections present a quick snapshot of some motivating queries.

5.1 Cyber-Security

A cyber system is naturally described as a graph with physical machines, IP addresses, users, and software services as entities (vertices). The relationships between these entities such as communication between machines, association of a physical machine and an IP address, login of a user on a machine etc are modeled as edges in the graph. From a security perspective, updates to this dynamic graph can be constantly monitored to detect events such
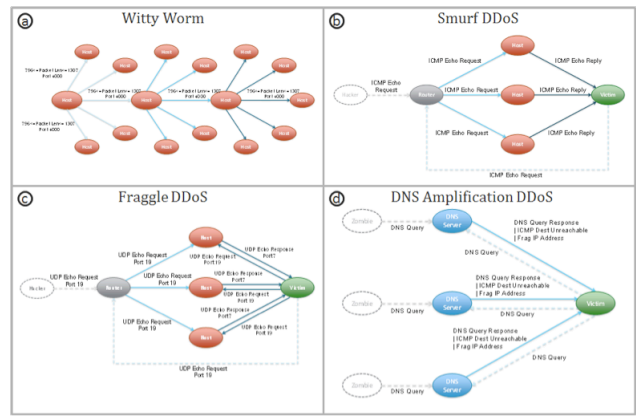

Figure 3: Examples queries to detect cyber attacks.

as worm spread, virus attack, denial-of-service attack etc.. We construct graph-based representation of these events (Fig. 3) and query the data graph to detect occurrences of malicious events.

5.2 News and Social Media

Various online news or social media data sources can be represented as multi-relational graphs. Entities such as articles, events, people, location, organizations and keywords can be represented as vertices in the graph. Next, graph based queries can be executed to detect the occurrence of various events in the news stream. Fig. shows some example queries and Fig. 5 shows a map-based visualization of the a series of queries executed on New York Times data [1].

6. DEMONSTRATION SETUP

6.1 Setup

Dataset: We will demonstrate the query capabilities on internet traffic data obtained from www.caida.org. The number of records in these datasets typically varies between 50-100 million/hour.

Software/Hardware The queries will be executed on a 48-core shared memory system running Linux 2.6.18 and comprising 2.3 GHz AMD Opteron 6176 SE processors and 256 GB memory. Each system node has 32 GB memory attached to it. The graph query engine is implemented in C++.

6.2 User Interface

There are three major focus areas for visualization and UI design.

- Our primary target audience includes journalists, emergency responders, intelligence professionals who are not expected to use StreamWorks using an API. Fig. 4 shows an experimental user interface for visual query composition. The user interface will retrieve metadata information such as vertex and edge types and their attributes to assist in drawing a query graph.

- The query graphs are a representation of events of interest; hence, we are developing map (Fig. 5) and tabular views (Fig. 6) that show occurrence of events in a geospatial and temporal context. The goal is to keep the underlying graph representation transparent to the user. Query results from any graph with location information available as a vertex attribute can be displayed on the map view.
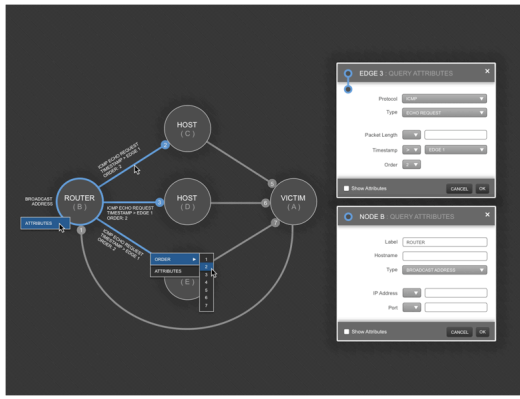
---

[1] http://data.nytimes.com

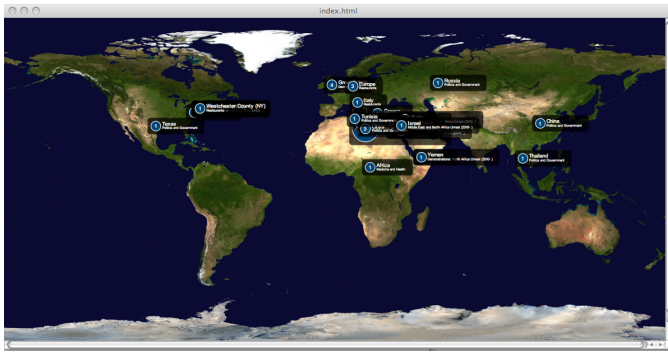**Figure 4: Prototype of an interface for visual graph query composition.**

Figure 5: A visualization of the output from a collection of graph queries. The queries are similar to Fig. 2. Each query graph specifies a label (such as "politics", "accident" etc.) on the keyword vertex to indicate the event of interest.
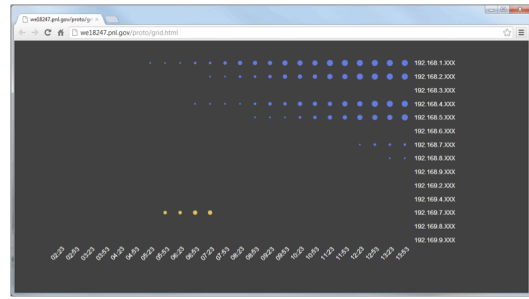

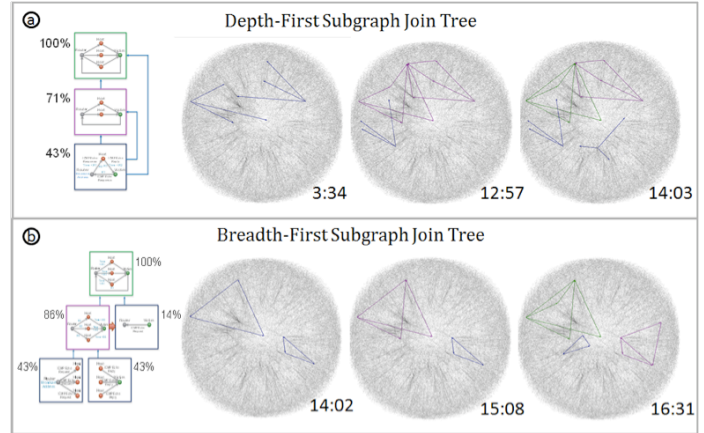**Figure 6: Grid-based visualization showing cascading effect of a Smurf DDoS attack across subnetworks (blue dots).**

Figure 7: Emerging matches for Smurf DDoS subgraph patterns in a dynamic computer network using different query plans.

- Graph-based visualization of the results from subgraph queries is critical for developers and API users. Therefore, we are adapting and applying the Gephi graph visualization and manipulation software [1] to render snapshots of the data graph and encode the partial and complete matches. This is also useful to observe the choice of different query decomposition strategies. To illustrate, Fig. 7 shows snapshots of emerging subgraph patterns in a computer network that are identified and tracked using different SJ-Tree structures. The percentages show the fraction of query graph being matched as measured by the number of edges. Each SJ-Tree is shown next to its associated emerging subgraph pattern snapshots. The colors of the subgraph patterns in the snapshots correspond to particular partitions in the associated SJ-Tree to indicate the level or degree of partial matching to the query graph.

7. ACKNOWLEDGMENTS

8. REFERENCES

[1] Gephi, an open source graph visualization and manipulation software, www.gephi.org.

[2] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12:120–139, August 2003.

[3] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. Telegraphcq: continuous dataflow processing. SIGMOD '03.

[4] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. Niagaracq: a scalable continuous query system for internet databases. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, pages 379–390, New York, USA, 2000. ACM.

[5] L. Chen and C. Wang. Continuous subgraph pattern search over certain and uncertain graph streams. *IEEE Trans. on Knowl. and Data Eng.*, 22(8):1093–1109, Aug. 2010.

[6] S. Choudhury, L. Holder, A. Ray, G. Chin, and J. Feo. Continuous queries for multi-relational graphs. *Pacific Northwest National Laboratory technical report, PNNL-SA-90326, http://arxiv.org/abs/1209.2178*, 2012.

[7] W. Fan, J. Li, J. Luo, Z. Tan, X. Wang, and Y. Wu. Incremental graph pattern matching. SIGMOD '11, 2011.

[8] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. Efficient subgraph matching on billion node graphs. *PVLDB*, 5(9), 2012.