LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-TR-225656

API Requirements for Dynamic Graph Prediction

B. Gallagher, T. Eliassi-Rad

October 27, 2006

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

API Requirements for Dynamic Graph Prediction

Brian Gallagher and Tina Eliassi-Rad

September 27, 2006

1 Problem at Hand

Given a large-scale time-evolving multi-modal and multi-relational complex network (a.k.a., a large-scale *dynamic semantic graph*),[1] we want to implement algorithms that discover patterns of activities on the graph and learn predictive models of those discovered patterns.

This document outlines the application programming interface (API) requirements for fast prototyping of feature extraction, learning, and prediction algorithms on large dynamic semantic graphs.

2 General Requirements

When selecting an API for access, manipulation, and transient storage of dynamic semantic graphs, we need to consider the following four general requirements:

1. Extensibility: Any API we choose will be extensible to some extent. However, ease of extensibility may be as issue.
2. Flexibility: Will the API (or the database system) lock us into a particular representation of time? Or force us to view every piece of data as a graph? Can these limitations hinder efficiency of algorithms?
3. Open source: We would like to have the option of "improving" the API such as API bug fixes and performance tuning, as necessary. Moreover, some changes are not possible by simple extension of an API, but require modifications to it. A major caveat to significantly modifying an existing API is the impractically of incorporating updates and fixes made by the third-party to the original API.
4. Licensing issues: Regardless of whether the chosen API is open source, we will need to consider software licensing issues.

There exist graph APIs that satisfy the above general requirements – such as CASOS tools[3], JUNG [7], and NetworkX [4]). However, none of these are specifically designed for large dynamic semantic networks, which require all of the following conditions :

1. Representation for multi-modal nodes and multi-relational links (with varying attributes)
2. Representation for an ontology graph that defines how to combine data from multiple sources
3. Mechanism for capturing time sequentially (e.g., Monday, Tuesday, Wednesday, ...) and hierarchically (e.g., day, week, month, year, ...)
4. Mechanism for dealing with large-scale graphs (such as through intelligent caching)

[1] A large-scale dynamic semantic graph can have on the order of 10^9 nodes. Its nodes are multi-modal and are connected via multi-relational links. The modality of nodes and links are defined by an *ontology graph*. The graph is dynamic because nodes and links are inserted/modified/deleted over time (as new data becomes available).

3 Specific Requirements

In this section, we discuss specific requirements for developing algorithms on large-scale dynamic semantic graphs.

First, we require basic graph operations on undirected, directed, and semantic graphs. When operating on a semantic graph, we will also need graph operations on the ontology graph. Examples include:

1. Accessing nodes and links: Retrieve nodes, links, and subgraphs by type and attribute values. Return degree matrix, adjacency matrix, and/or Laplacian matrix of a graph.
2. Graph traversal operations: Return all the links from and/or to a node. Return the end nodes of a link.
3. Accessing semantic graph data: Retrieve type or attribute values for a node or link.

Second, we require operations that allow modifications to (*i*) the graph structure, (*ii*) the semantic data, and (*iii*) the ontology. Examples include:

1. Modifying graph structure: Create/delete nodes and links. Add nodes and links to a graph. Remove existing nodes and links from a graph and (optionally) re-insert them elsewhere in the graph.
2. Modifying semantic graph data: Add/delete/modify type and attribute values of a node or link.
3. Modifying an ontology graph: Create/delete concept and relation types and add/delete attributes associated with those types.

Third, we require temporal operations. Most of these operations should return a time-series of some kind. Examples include time-series of graph metrics (such as average node degree over a month), time-series of nodes, links, or subgraphs that satisfy a set of conditions on either the structure or semantics of the graph, etc.

Fourth, we require flexible data representation. Given a dynamic semantic graph, the API should allow us to experiment with different views of the graph in terms of time, semantics, and structure. For instance, in addition to representing temporal information as timestamp attributes on nodes and links, the API should also allow us to operate on a time-series view of the graph. We also require flexibilities with representing nodes as links and vice versa. For example, in one algorithm we may consider an email as a concept (i.e., a node in the graph) and in another algorithm we may consider an email as a relation between people (i.e., a link). Furthermore, multigraph and hierarchical representations of concepts/relations will improve scalability and simplicity (such as in the case of type inheritance).

Finally, we require libraries that are built on top of the API and provide common functionalities on dynamic semantic graphs. Examples include:

1. Visualization [8, 1]
2. Sampling for the purpose of generating a smaller size graph that remains "faithful" to the original larger graph [6]
3. Metrics (such as degree distribution per type, clustering coefficient, etc) [2]
4. Operations on matrix and tensor representations of the graph [5]

4 Conclusion

Since our algorithms must operate on large-scale dynamic semantic graphs, we have chosen to use the graph API developed in the CASC Complex Networks Project. This API is supported on the back end by a semantic graph database (developed by Scott Kohn and his team). The advantages of using this API are (*i*) we have full-control of its development and (*ii*) the current API meets almost all of the requirements outlined in this document.

Acknowledgments

This work was performed under the auspices of the US Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.

References

[1] James Abello and Yannis Kotidis. Hierarchical graph indexing. In *Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management (CIKM)*, pages 453–460, November 2003.

[2] Mark Barthelemy, Edmond Chow, and Tina Eliassi-Rad. Knowledge representation issues in semantic graphs for relationship detection. In *Papers from the 2005 AAAI Spring Symposium on AI Technologies for Homeland Security*, pages 91–98, March 2005.

[3] Kathleen M. Carley et al. Computational analysis of social and organizational systems (CASOS): Tools for dynamic network analysis. http://www.casos.cs.cmu.edu/computational_tools/menu.html, 2001.

[4] Aric Hagberg, Dan Schult, and Pieter Swart. Networkx: High productivity software for complex networks. https://networkx.lanl.gov/, 2004.

[5] Wolfgang Hoschek. COLT: Open source libraries for high performance scientific and technical computing in java. http://dsd.lbl.gov/ hoschek/colt/, 1999.

[6] Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 631–636, August 2006.

[7] Joshua O'Madadhain, Danyel Fisher, Tom Nelson, Scott White, and Yan-Biao Boey. Java universal network/graph framework (JUNG). http://jung.sourceforge.net/, 2003.

[8] Zeqian Shen, Kwan-Liu Ma, and Tina Eliassi-Rad. Visual analysis of large heterogeneous social networks by semantic and structural abstraction. *IEEE Transactions on Visualization and Computer Graphics, Special Issue on Visual Analytics*, 2006. to appear.