Mure.js: Toward Flexible Authoring and Reshaping of Networks

Alex Bigelow[*] Carolina Nobre[†] Alexander Lex[‡] Miriah Meyer[§]

University of Utah

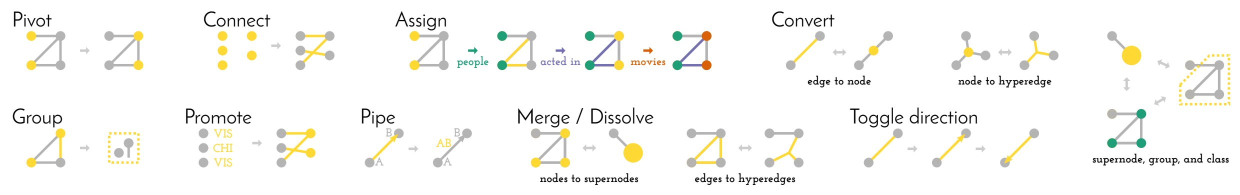

Figure 1: Examples of operations that our framework is meant to support. These include using graph topology for pivoting selections, connecting nodes, assigning classes to items, converting between constructs, grouping items, promoting values to nodes, piping attributes to or along edges, merging and dissolving supernodes and hyperedges, and toggling edge direction.

Index Terms: Human-centered computing—Visualization—Visualization systems and tools; Information systems—Data management systems—Database design and models—Graph-based database models

1 Introduction

When interpreting data as a graph for visualization, an analyst first assigns semantic meaning to graph concepts. For example, they may choose to represent actors and movies as nodes, and roles as edges. Alternatively, they may wish to represent movies as edges, connecting actor nodes when they collaborate. Data abstraction choices such as these are critical, because different data abstractions can limit—or inspire—different analysis questions, approaches, perspectives, and visualizations [2, 5].

However, current network modeling frameworks, systems, and databases narrowly define graph abstraction **constructs**—such as nodes, edges, node / edge classes, supernodes, hyperedges, etc.—in terms of how the data is stored in memory or on disk, rather than semantic, human-driven abstractions. Consequently, the ability of an analyst to iterate on a data abstraction becomes fundamentally limited by the implementation details of data wrangling software.

We present work-in-progress toward a broader framework for modeling network data that is less dependent on its underlying structure and storage. Our goal is to use semantic data abstraction constructs to inform how algorithms wrangle data, instead of allowing algorithmic concerns to define and constrain the semantics.

We also present *mure.js*, a software library that represents an initial implementation of this framework, allowing users to map semantic graph constructs to arbitrary data structures as metadata, that can, in turn, be used to select, navigate, and reshape the underlying data. Additionally, we discuss an early software prototype of a visual graph wrangling system based on this library.

2 Related Work

The framework for semantic network modeling that we are developing extends ideas from existing network modeling frameworks, systems, and databases that stress the role of the user in deciding what parts of the data are nodes, and what parts of the data inform connections [4, 6]. Like these existing efforts, we emphasize the fact that a data abstraction is designed, rather than naturally occurring.

Existing network modeling tools tend to define constructs in terms of specific data storage strategies, such as the shape of data objects in memory [3], as specific interpretations of relational database concepts [7], as specific interpretations of attribute values [6], or some specific combination of these interpretations [4]. These approaches have clear implementation advantages, in that it is easy and efficient to implement general-purpose algorithms [3], efficient queries in graph databases [7], powerful strategies for establishing connections [4, 6], or efficient visualizations of many nodes [1]. However, rigid, data-based construct definitions have two distinct disadvantages: they result in limited abstractions, and they are brittle to refactoring.

When constructs are defined by data structure, they are limited by that structure. For example, NetworkX [3] requires that an entire graph be defined *a priori* to conform to a specific set of constraints, such as directed vs undirected. These constraints make sense from an algorithmic perspective, but because edges must follow one of these global patterns, there is no support for graphs that mix directed and undirected edges, or for other constructs such as hyperedges. Still other frameworks, such as Orion [4], define connections as simple value-based links, rather than distinct items, making it impossible for edges to have their own attributes. Data-defined constructs commonly result in workarounds, such as creating intermediate nodes as a place to store edge attributes, or as a proxy for hyperedges.

The other disadvantage of data-defined constructs is that iterating on a data abstraction is difficult or impossible in practice. Relatively simple semantic changes, such as swapping what is interpreted as a node, with what is interpreted as an edge, can be incredibly powerful in practice [5], yet support for this kind of graph wrangling is underexplored.

Our framework is distinct from existing work in that network modeling takes place at a more semantic level. It *maps* semantic constructs to data values and structures, rather than defining them in terms of data values and structures. As such, refactoring the mapping is far less complicated, and it enables a richer library of constructs.

3 Mapping Constructs to Data

Graph data can be challenging to store, especially as edges often introduce cyclical structures that do not map well to file system or data structure hierarchies. Consequently, many strategies exist for representing a graph in memory, including adjacency matrices,

[*]e-mail: abigelow@cs.utah.edu
[†]e-mail: cnobre@sci.utah.edu
[‡]e-mail: alex@sci.utah.edu
[§]e-mail: miriah@cs.utah.edu

node-link lists, and relational tables.

Our approach is to map semantic graph constructs to **items** in data structures as they already exist, at an instance level, rather than attempt to force all graph data into some canonical structure. As such, items could be rows in a CSV file or relational table, objects in a JSON file or NoSQL hierarchy, or elements in an XML document. At the data level, the only requirement is that an item must at least support attributes; it does not matter to our framework whether they are nested, or where in a file they exist. Once items are initially mapped to semantic constructs, they enable a set of operations that allow conversion from one construct to another—this process informs and drives the underlying data wrangling operations, making it possible to convert between different storage strategies.

We have hinted at one clear limitation of this approach: in attempting to avoid a single, canonical data structure, we can not rely on the strengths of any of them, such as the compactness of adjacency matrices for highly-connected graphs. This means that we risk the ability to scale to larger datasets. We suspect that there may be a fundamental trade-off in data wrangling tools—where greater semantic flexibility may come at the cost of scalability.

3.1 Constructs

Our framework currently identifies nodes, edges, supernodes, hyperedges, classes, and groups. This is not necessarily the only set of useful constructs that exist—we intend to refine this list, with feedback from the community. Meta-iteration on constructs themselves is another advantage of our more lightweight, semantic approach.

The simplest construct mapping that can be made is to identify an item as a **node**—an independent item in a graph. An **edge** is a dependent item that links at least two nodes, even if both nodes are the same (a self-edge). An edge may or may not be directed, independent of any other edges in the graph.

A **supernode** is a node that represents a set of nodes; it allows the user to summarize less-relevant network complexity. Supernodes also provide a target for edges that are meant to link to a set of items as a whole, rather than its individual members. A **hyperedge** is an edge that links more than two nodes, and may be fully directional, partially directional, or undirected.

Nodes and edges may be further identified with **classes**—a mechanism whereby the user organizes items by their semantic, real-world meaning, enabling item comparison across a common set of attributes. Additionally, **groups** allow a user to specify informal or *ad-hoc* item relatedness—distinct from the topological notion of relatedness represented by edges, supernodes, or hyperedges, and also distinct from the attribute-based relatedness implied by classes.

3.2 Operations

Once initial constructs are in place, they enable a rich set of operations that the user can perform to reinterpret and reshape the data. Similar to constructs themselves, available operations are also open-ended in our framework; the operations listed here, also shown in Figure 1, are meant to be illustrative, not comprehensive.

We describe each operation in terms of user **selections**—in our prototype interface, these manifest as interface selections, whereas in the `mure.js` library, they are more similar in spirit to D3 selections. One selection can yield another by **pivoting**—either to, from, or along edges; or from a set-like construct (supernode, group, or class) to its contents or members.

Items can be **grouped**, saving the selection for later use or conversion to a different construct; they can also be **assigned** to new or existing classes. Nodes can be **merged** into a supernodes; supernodes can be **dissolved** back into regular nodes; nodes can **connected** to each other, similar to (but not necessarily implemented as) a relational database join; or **converted** to hyperedges, merging the nodes' existing edges into a single item. Similarly, selected edges and hyperedges can be merged and dissolved; or converted to nodes—splitting an edge into separate entities, and creating an intermediate node.

Selected values or attributes can be **promoted** to distinct nodes, optionally connecting their original source nodes with new edges. Attributes can also be **piped** along edges, deriving new node or edge attributes based on connected properties.

4 Implementation

We are implementing an example of our framework, focusing on the above constructs and operations as an open-source library called *mure.js*[1]. Similar to other graph database approaches, it is a semantic layer on top of an existing database—in this case, PouchDB—but, unlike existing approaches, the layer is far more lightweight, and constructs do not rely specifically on the shape or structure of PouchDB documents. Instead, constructs are mapped to items in documents—or even the documents themselves—through a metadata layer.

In conjunction with the library, we are prototyping a visual tool for semantic graph wrangling. The interface exposes the constructs and operations made available by the library, in a way that non-programmers will be able to take advantage of them. Additionally, it will provide basic visualization capabilities to assist in inspecting the data before it is exported for deeper visualization in other programs.

5 Conclusions and Future Work

As mentioned above, we plan to continue refining the set of constructs and operations, and to continue developing and designing *mure.js* and its corresponding visual interface. We also intend to evaluate the expressiveness, usefulness, and/or the usability of the resulting systems.

In presenting this poster, we hope to solicit feedback from the community on our current directions. Perhaps most importantly, we seek opportunities to discuss how to best evaluate this kind of work. What kinds of usability tests are most appropriate? How—and should—data wrangling tool expressiveness be validated beyond case studies? How can we evaluate the usefulness of a wrangling tool in a way that can provide insight with respect to the set of constructs and operations that it supports?

References

[1] M. Bastian, S. Heymann, and M. Jacomy. Gephi : An Open Source Software for Exploring and Manipulating Networks. *Proceedings of the Third International ICWSM Conference*, p. 2, 2009.

[2] A. Bigelow, S. Drucker, D. Fisher, and M. Meyer. Reflections on How Designers Design with Data. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces*, AVI '14, pp. 17–24. ACM, New York, NY, USA, 2014. doi: 10.1145/2598153.2598175

[3] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring Network Structure, Dynamics, and Function using NetworkX. In G. Varoquaux, T. Vaught, and J. Millman, eds., *Proceedings of the 7th Python in Science Conference*, pp. 11 – 15. Pasadena, CA USA, 2008.

[4] J. Heer and A. Perer. Orion: A system for modeling, transformation and visualization of multidimensional heterogeneous networks. *Information Visualization*, 13(2):111–133, Apr. 2014. doi: 10.1177/1473871612462152

[5] C. B. Nielsen, S. D. Jackman, I. Birol, and S. J. M. Jones. ABySS-Explorer: visualizing genome sequence assemblies. *IEEE transactions on visualization and computer graphics*, 15(6):881–888, Dec. 2009. doi: 10.1109/TVCG.2009.116

[6] A. Srinivasan, H. Park, A. Endert, and R. C. Basole. Graphiti: Interactive Specification of Attribute-Based Edges for Network Modeling and Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):226–235, Jan. 2018. doi: 10.1109/TVCG.2017.2744843

[7] J. Webber. A Programmatic Introduction to Neo4j. In *Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity*, SPLASH '12, pp. 217–218. ACM, New York, NY, USA, 2012. doi: 10.1145/2384716.2384777

[1]The library, source code, and documentation are available at `https://github.com/mure-apps/mure-library`