Orion: A System for Modeling, Transformation and Visualization of Multidimensional Heterogeneous Networks

Jeffrey Heer* Stanford University
Adam Perer† IBM Research

Abstract

The study of complex activities such as scientific production and software development often require modeling connections among heterogeneous entities including people, institutions and artifacts. Despite numerous advances in algorithms and visualization techniques for understanding such social networks, the process of constructing network models and performing exploratory analysis remains difficult and time-consuming. In this paper we present Orion, a system for interactive modeling, transformation and visualization of network data. Orion's interface enables the rapid manipulation of large graphs — including the specification of complex linking relationships — using simple drag-and-drop operations with desired node types. Orion maps these user interactions to statements in a declarative workflow language that incorporates both relational operators (e.g., selection, aggregation and joins) and network analytics (e.g., centrality measures). We demonstrate how these features enable analysts to flexibly construct and compare networks in domains such as online health communities, academic collaboration and distributed software development.

Keywords: social network analysis, data management, data transformation, graphs, visualization, end-user programming.

1 Introduction

As social network analysis has gained popularity, researchers have developed novel statistical techniques, visualization designs, and user interfaces to better support the complex tasks of making sense of large networks. However, many of these recent advances take the process of assembling a network model for granted. Much data that is collected for analysis, whether scraped from online data sources or tabulated using traditional surveys, is not inherently in the form of a network but instead a raw list of data points and their corresponding attributes. This requires analysts to extract their own model of a network from the raw data. But for many data sets, networks can be modeled in as many different ways as analysts have hypotheses. For instance, after collecting a database of online community data, analysts may wish to examine the relationships between community members to measure collaborative support, or the relationships between thread posts to measure the dissemination of information, or the relationships between communities as a whole to measure comparative community success. To analyze each of these scenarios, completely different network models need to be extracted from the original data.

Typically, refactoring network data into such various models requires custom code that can take analysts days or even weeks. While it is also possible to express most of the necessary operations as database queries, this requires defining a schema, loading the data, and then forming the correct SQL queries — including complex queries such as multi-table joins. Repeating this level of effort as new questions emerge may undermine the exploratory process and even dissuade some analysts from testing all hypotheses.

*e-mail: jheer@cs.stanford.edu
†e-mail: adam.perer@us.ibm.com

To address these issues, we introduce *Orion*, a system for interactive modeling, transformation and visualization of network data. While many visualization and data mining techniques have been proposed for social network analysis, to our knowledge Orion is unique in its support for the early stages of data transformation and assessment when forming network models from source data. Orion enables iterative, exploratory analysis by reducing hours of programming and transformation to a few minutes of interactive, graphical specification. We make the following contributions:

A unified model and workflow language for network data. We use relational data tables as our fundamental model and represent networks as edge tables over a domain of integer node indices. We chose this model to correspond to those used by analytic databases and scalable network analysis packages. Our workflow language provides both relational operators and graph analysis routines and enables the generation of reusable analysis scripts. The language supports a range of analysis tasks including network definition, filtering, aggregation, and statistics computation.

A graphical user interface for iterative network manipulation and visualization. Orion translates user interface actions, such as drag-and-drop and menu commands, into operations in the underlying workflow language. Orion also supports the specification of complex linking relationships. The system first constructs a graph model of relationships among table columns; a traversal algorithm then identifies all feasible network models for a set of user-selected node types. This approach simplifies the otherwise difficult process of specifying a series of relational join and aggregation operations. Once a network has been defined, Orion enables visual exploration using tabular, matrix and node-link views. Networks can also be exported for use in other analysis tools.

The rest of the paper is structured as follows. After reviewing prior work, we describe our data model and present the Orion interface. Next, we discuss our enabling algorithms for network extraction and describe our workflow language. As a preliminary evaluation of Orion, we demonstrate its use in case studies of online health communities, academic collaboration and distributed software development. We then discuss future work and conclude.

2 Related Work

Orion draws on related work in graph visualization, analysis tools, and data management. We discuss selected relevant projects below.

2.1 Graph Visualization Techniques

Researchers have devised a variety of visualization techniques for networks [15]. Two common representations used in social network analysis are node-link diagrams (typically using force-directed placement) and adjacency matrix views [8]. Hybrids of the two have also been proposed [13, 14]. These approaches organize elements according to the linkage structure of the graph.

An alternative approach is to plot network data according to the attributes of the nodes (e.g., [27, 33]), as in a scatter plot or so-called "semantic substrates" [27]. Network links can then be drawn between nodes. This approach is well-suited for assessing potential correlations between node attributes and network structure.

In a related vein, PaperLens [22] uses multiple coordinated views of network attributes to explore publication databases. The NetLens system [21] generalizes this approach to support networks that fit a "content-actor" data model, i.e., bipartite networks such as publications and authors. In contrast, Orion supports an arbitrary number of linking relationships both within and between data tables.

Others have researched means of dealing with large graphs in excess of tens of thousands of nodes. Common strategies include filtering and aggregation. van Ham and Perer [31] introduce a degree-of-interest function [6] that reduces a graph to a small connected subset of nodes based on an input set of foci (e.g., search results). PivotGraph [33] and Honeycomb [32] aggregate networks by "rolling up" edges according to node attributes; for example, an analyst can collapse a social network of corporate employees to show the summed connection strengths between workers' geographic locations. ManyNets [5] enables comparison among multiple networks using a tabular view of summary graph statistics. If desired, users can still view standard (albeit less scalable) node-link diagrams on demand.

Orion draws on this prior work: it provides both node-link and matrix visualizations and supports network aggregation based on node attributes. However, with the sole exception of NetLens [21], each of the above systems assume a well-defined network is given as input to the tool. None of these tools help the user define and assess a variety of network models derived from arbitrary data tables.

2.2 Network Analysis Tools

Recent years have seen a proliferation of network analysis tools. Many of these tools combine visualization and statistics within an interactive environment [1, 7, 24, 26, 28]. Others are programming libraries [17, 18, 23] or menu-driven interfaces [3, 30] that provide access to analysis algorithms. While these tools support data import from common file formats (e.g., GraphML) and external data services (e.g., Twitter or Facebook), they do little to facilitate the flexible construction of network models from arbitrary data tables. Orion is not intended as a replacement for these systems; rather, it is designed to assist the unsupported early stages of network analysis. In the process, it enables the use of these downstream tools.

2.3 Managing Graph Data

Another domain of related work is graph data management. Database researchers [2, 9] have developed storage strategies and query languages [9] for large networks. Similarly, a number of commercial systems — including neo4j, InfiniteGraph, AllegroGraph, DEX, OrientDB, and sones/GraphDB — are now available. These systems facilitate storage, indexing, and querying of large graphs, but with goals different from Orion. Representative applications include managing social network web sites and querying motifs in biological networks. Orion instead supports the construction and assessment of network models from source data.

2.4 Interactive Data Transformation and Querying

Orion focuses on transforming data to create network models. In a related vein, other research systems have been designed to assist the early stages of data cleaning and reformatting. Google Refine [16] and Data Wrangler [19] enable analysts to reformat input data sets and correct data errors prior to analysis. D-Dupe [20] assists the process of finding and resolving duplicates within a data set. Any of these tools might be used to prepare data tables prior to network modeling with Orion. Similar to Wrangler, Orion produces as output not only data, but also a declarative transformation script that can be reapplied to new data and inspected to review data provenance. In this light, Orion can also be understood as an end-user programming tool for network manipulation.

Orion was particularly influenced by the Polaris system [29], now commercialized as Tableau. Polaris maps drag-and-drop operations of data variables into a formal algebra from which both database queries and resulting visualizations are derived. One key insight from this work is the value of deeply coupling visualization tools with rich facilities for data transformation. Orion similarly provides a user interface in which user actions are mapped to statements in an underlying data transformation language. While Polaris enables filtering and aggregation operations over a single data table, Orion instead enables manipulation of multiple tables, including linking relationships realized as relational joins.

3 Data Model

A variety of data models exist for handling graph data; common structures include adjacency lists and adjacency matrices. However, these representations alone are insufficient for network analytics, as in many cases networks must be derived from a prior data source permitting a number of models and parameterizations. As a simple example, a social network extracted from an email archive might include links only between senders and recipients, or might include links among all co-recipients.

Prior research on visualization toolkits has noted the value of representing networks as relational tables [10]: each row represents an edge in the graph, and columns contain *source* and *target* node values among other edge attributes. This format provides a sparse representation of the network, enables easy querying of attributes, and supports efficient edge iteration. On the other hand, this format is inefficient for path following and is thus ill-suited for many graph algorithms. As a result, we adopt a hybrid data model in Orion.

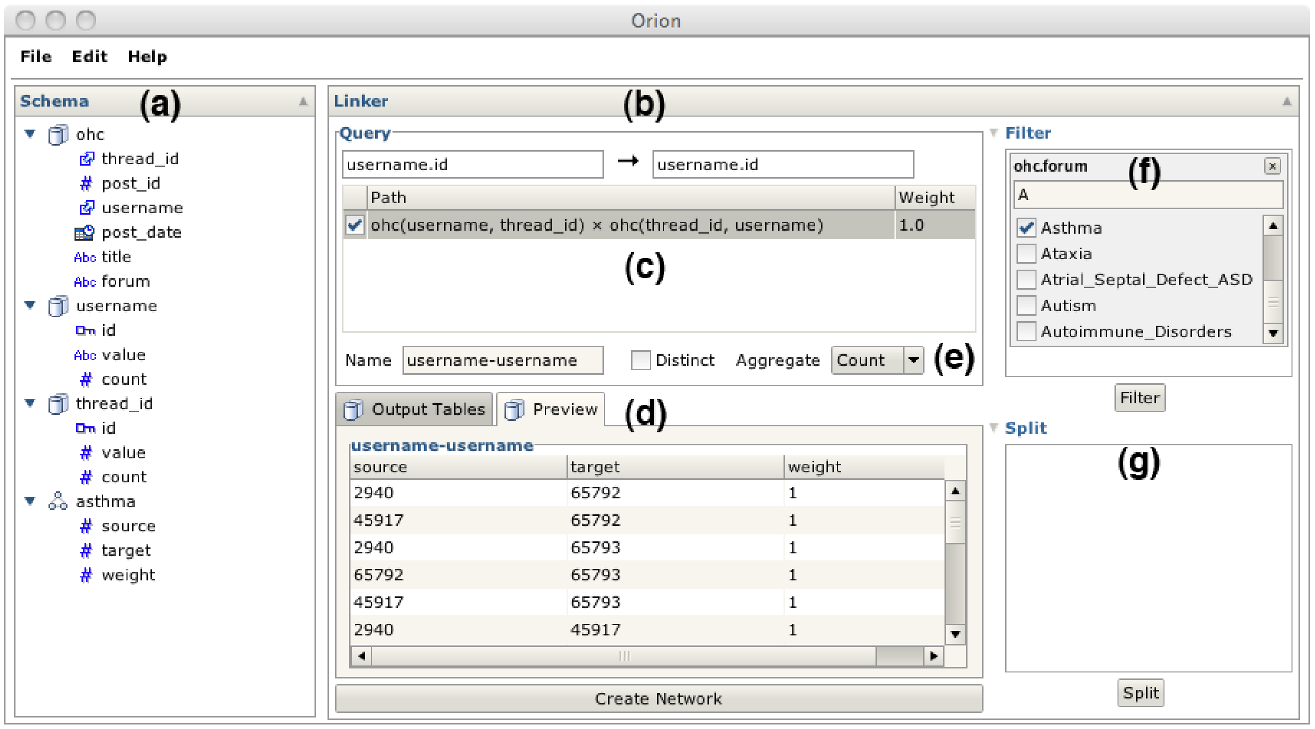
We use relational data tables as our base representation. Tables can represent individual node types or linking relationships. At times, node types may be implicit within the attributes of a table; Orion provides methods to promote these values to their own table. Networks can be inferred from the foreign key relations among tables. This design allows us to support arbitrary node types and linking relations while facilitating integration with relational databases.

Once a specific linking relationship has been chosen (as described in subsequent sections), Orion models the network using a specialized edge table format. Source and target columns represent incident nodes using zero-based integer indices. For efficient processing, these indices default to the row index in the corresponding node table. This scheme works well for edges involving a single node table, but leads to index collisions among different tables. To ensure distinct keys we bias the indices for a given table by the total size of any previous tables. The mapping from node tables to index ranges is stored as metadata for the edge table.

Some graph analysis routines, such as force-directed layout or clustering coefficient calculation, can be performed by simply iterating over edges. However, other methods — including shortest path and betweenness centrality algorithms — must traverse the graph by following paths. Accordingly, our edge tables support the construction and caching of adjacency lists, represented as an array of sorted integer arrays for in-links, out-links or both.

This integer-based representation provides multiple benefits. In particular, it allows rapid access of associated node data via index-based table lookups and facilitates the creation of efficient network analysis routines. Representing nodes as zero-based integers enables the use of simple arrays to keep state within graph algorithms, avoiding the overhead of associative data structures.

Our data model, like the rest of Orion, is implemented in the Java programming language. We have implemented our own data structures and processing routines, but our data model was intentionally chosen to correspond to those used by modern analytic databases and scalable network analysis packages (e.g., [23]). In future work, we want to exploit this correspondence to implement our workflows on massively scalable platforms. We use relational operators for as much of our workflow as possible so that we can later leverage shared-nothing parallel databases. That said, we will show shortly that our own implementation already scales to networks involving millions of elements, and so supports a broad class of data sets.



	Path	Weight
✓	ohc(username, thread_id) × ohc(thread_id, username)	1.0

username-username

source	target	weight
2940	65792	1
45917	65792	1
2940	65793	1
65792	65793	1
45917	65793	1
2940	45917	1

Figure 1: The Orion User Interface, consisting of (a) a schema viewer for manipulating data tables and (b) a linker interface for creating network models. Analysts drag-and-drop desired node types to the linker and Orion responds with (c) a table of possible linking paths. The (d) preview display shows the resulting network data. Analysts can also specify (e) aggregation, (f) filtering, and (g) splitting (subdivision) criteria.

4 The Orion User Interface

With the Orion user interface, analysts can import source data from multiple formats, specify a variety of network models, compute statistics, and visualize the results. Analysts can then export either the resulting data or a declarative script defining the transformation workflow. In this section, we first describe the design of the Orion interface through a concrete usage scenario. We then provide more detailed descriptions of Orion's user interface components.

4.1 Usage Scenario

Consider the real-world example of a researcher studying online health communities organized around medical conditions (e.g., asthma, lupus, lyme disease, etc). Driving questions include: How do community dynamics and structure vary across conditions? Can we gain new insights from the co-occurence of symptoms and conditions? To explore these questions, our analyst collected over 3 million discussion posts from MedHelp.org, a public online health site. The initial database consists of a single table where each row represents a post on the site. Table columns include a forum (community) name, the user name of the poster, the post date, the title of the discussion thread, and the post text.

From this data, the analyst would like to analyze the social networks of the individual communities. She begins by importing the data table (a large CSV file) into Orion. The table and its columns are displayed in the Schema Viewer in Figure 1a.

Next, she must define the entities of interest that might form the nodes of her graph. Currently, these entities reside implicitly as values within the table. The analyst right-clicks the *username* field and selects "Promote" in the resulting context menu. This operation causes all username values to be extracted from the table: a new table is constructed with one row for each unique user and the username field in the original table is replaced with a foreign key referencing the new table. As the analyst wishes to model a social network based on co-participation within a discussion thread, she similarly promotes the *thread_id* field as an entity of interest.

The analyst would now like to construct a social network among users. She drags the *id* field (the primary key) from the *username* table and drops it on the Linker interface in the center of the Orion window (Figure 1b). The interface allows analysts to specify desired source and target node types. In response, Orion calculates all feasible network definitions involving *username* entities as nodes (Figure 1c). In this case, there is only one feasible result: linking users by shared threads. Should the analyst wish to consider alternatives, she could promote other entities (e.g., individual forums).

When the analyst clicks the check box to include the linking path, Orion responds by showing a preview of the resulting graph (Figure 1d). Orion previews include both a list of tables that will be generated, and an inspector for individual table values. For now the analyst would like to limit her exploration to a single community. She drags the *forum* field from the Schema Viewer to the Filter region of the Linker; she then selects the "Asthma" forum using the resulting search box (Figure 1f). The preview updates in turn. Satisfied, the analyst clicks the "Create Network" button to add the network to the data set; the Schema Viewer updates with a new edge table containing links between all posters to the "Asthma" forum who have posted to the same thread; by default, edge weights indicate the number of shared threads between two users.

By right-clicking the "Asthma" edge table, the analyst reveals additional options. She can choose to visualize the network using both matrix and node-link diagrams. In a matrix overview (Figure 2a), the disjoint structure of the community becomes evident which suggests that newcomers arrive into the community, ask a question, and it gets handled by one of a handful of leaders. This also suggests that the community may serve as an "answer mill" rather than a place of prolonged discussion. The analyst can dig deeper by filtering the visualization to only show a highly active cluster and pivots to a node-link visualization (Figure 2b). Here, with nodes sized by their post count and colored according to their betweenness centrality, the analyst can focus on specific nodes of interest for further analysis.

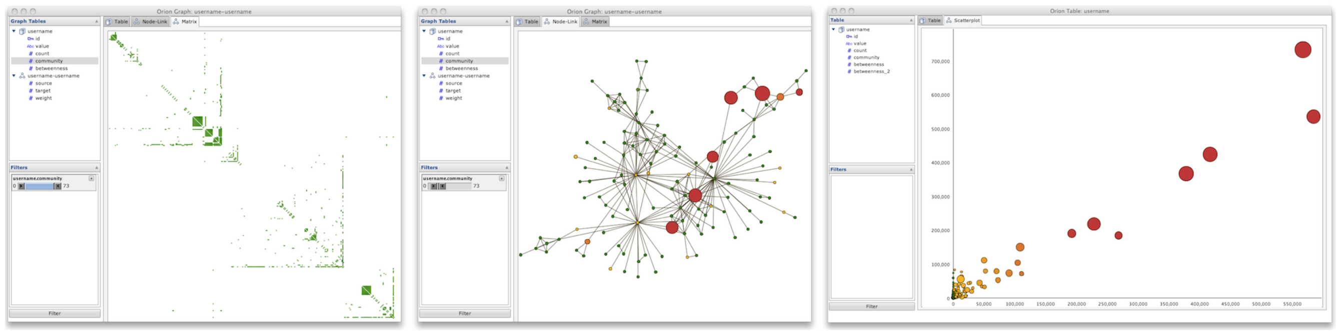


Figure 2: Orion Visualizations of Online Health Communities. From left-to-right: (a) A sorted matrix view of an online asthma forum. A few central leaders divide up responses among incoming questions. (b) Node-link diagram of highly active cluster of the same forum. (c) Plot of betweenness centrality values for two different network models, sized by number of posts. The models have similar centrality distributions.

Individual tables can be inspected and visualized using bar or scatterplot charts. Each visualization also supports interactive filtering controls. After filtering the graph to highlight interesting patterns, the analyst can save the filtered edge table as an additional entry in the Schema Viewer. The analyst can also compute statistics, including node degrees, betweenness centrality and clustering coefficients. Statistical operators add additional columns to the implicated edge and/or node tables.

Now the analyst would like to assess the effects of using a different network model. The current model includes edges connecting all posters to the same thread. What happens if instead thread respondents are connected only to the thread initiator? The analyst follows the same modeling path as before, but this time adds a join predicate: she right-clicks the linking path of the network and chooses to filter how the *ohc* table is linked to itself. In this particular data, a poster has a *post_id* of 0 if they initiated the thread and a *post_id* greater than 0 if they responded to the thread. As these data characteristics are specific to this particular community data, the analyst enters a customized formula in the resulting dialog:
`INT1('post_id')==0 && INT2('post_id')>0`

The formula ensures that the source node always corresponds to the thread initiator and that the target node is a respondent. The analyst creates this network, computes betweenness centrality values, compares values for the two models in a scatter plot (Figure 2c) and notes a high degree of correlation. She decides to proceed with the simpler model connecting only initiators to respondents.

The analyst would now like to start comparing the various health communities. She revisits her previous steps, but instead of filtering the *forum* field, she drags it to the Split region. The preview display then shows entries for multiple networks – one for each forum. Upon completion, these networks are grouped together within a subtree of the Schema Viewer. Context menus for the grouping element enables batch invocation of statistics for all contained networks (see Figure 7 for an example). The analyst can now continue analyzing the diverse characteristics of health communities.

4.2 User Interface Design

The previous scenario illustrates a subset of the modeling and visualization functionality supported by Orion. We now describe the user interface components in more detail. Along the way, we outline additional functionality, such as the ability to merge multiple sets of edges and construct "roll-up" graphs via node aggregation.

4.2.1 Schema Viewer

The Schema Viewer (Figure 1a) provides an overview of all data tables in the current data set and supports data manipulation. Source data tables and generated edge tables are indicated by icons. Table attributes are displayed using icons indicating their data type, with special annotations for primary and foreign key fields. Context menus enable analysts to rename and remove both tables and columns, create derived columns using an expression language, specify primary keys, and promote values in one or more columns to new node tables. Analysts can also access statistics and visualization options via context menus. Drag-and-drop interactions allow analysts to specify foreign key relations (by dragging a field on to a primary key with a matching type), import data (by dragging external data files from the operating system), and querying for network models (by dragging fields to the Linker interface).

4.2.2 Link Specification

The Linker interface (Figure 1b) is the primary means of defining networks. Analysts start by dragging desired node types to fields for source and target nodes. Orion responds by computing the possible linking paths between the source and target nodes and displays the results in a table. Users can select the resulting paths to include those edges within the resulting network model.

Filtering. Analysts can drag-and-drop fields to specify filtering criteria (Figure 1f). Filters can be created for any table involved in the network. Orion generates dynamic query widgets — selection lists and range sliders — based on the data type. Corresponding predicates are then applied during network construction to limit the nodes and edges included in the final graph. In addition to single table predicates, analysts can specify filtering criteria directly on joins. Filterable joins are presented in a context menu when an analyst right clicks a linking path. Currently, Orion only supports user-defined join predicates specified as Java code statements.

Splitting. An alternative to filtering is to split a network into a collection of subgraphs. Examples include inspecting time slices and splitting on categorical variables (e.g., health forums). Orion supports splitting by categorical variables or using window functions over quantitative variables (see Figure 7). Of course, not all such splits are useful: naïvely splitting on a node's primary key results in a collection of singleton graphs. In this special case, Orion instead interprets the split as a request for subgraphs centered at each node and provides a graph distance control. Analysts can extract all nodes and edges within a specified graph distance to isolate egocentric networks. Networks generated by splitting appear as grouped collections that support batch operations.

Rollup. When specifying desired node types via drag-and-drop, typically the primary key of a node table is used. If analysts instead drag-and-drop a non-key field, an aggregated network will be constructed that uses the unique field values as individual nodes (c.f., PivotGraph [33]). The underlying nodes are grouped according to the requested field; edges between groups are tallied to provide an aggregate representation of the underlying graph.

Multiple Edge Sets. By selecting multiple linking paths, Orion allows analysts to construct networks with multiple edge sets.

When multiple paths are selected, the linking interface enables controls for choosing an aggregate function for merging edge sets (Figure 1e); options include basic logical (or, and) and arithmetic (count, sum, product) operations. For arithmetic operations, analysts can also provide numerical weights for each edge set.

Preview & Confirmation. As analysts manipulate settings within the linker display, a preview panel updates in response (Figure 1d). Analysts can review the number and size of all tables generated, and inspect the values of individual tables. Once an analyst is satisfied with the linking definition, they can click the 'Add Network' button to add all resulting tables to the schema viewer.

4.2.3 Visualization

Orion also supports visualizations: table displays, basic data graphics (bar and scatterplot charts), node-link diagrams, and matrix views (see Figure 2). Visualizations are shown in a separate window with different visualization types accessible via tabbed panes. These windows include a schema viewer showing only the data tables implicated in the current visualization. Orion uses the Java implementation of the Protovis specification language [11] to generate these visualizations.

Analysts can parameterize a display using filtering, sorting, zooming, and visual encoding controls. Node-link diagrams use a force-directed layout algorithm based on a physical simulation. Matrix rows and columns can be sorted by node attributes or by linkage to hunt for patterns within the data. Layout and sorting facilities are included among the analytic operators described in §6.5. As our interactive data transformation methods constitute the primary contributions of this paper, we leave consideration of additional visualization facilities to future work.

5 Network Definition and Extraction

Having introduced the Orion interface, we now discuss some of the underlying algorithms enabling interactive network modeling. While relational tables provide a flexible model for representing data, network extraction involves creating linking queries that regularly include one or more join operations. As a result, defining networks via a query language such as SQL can be tedious and error-prone. To simplify the process, Orion models the connections among data tables and analysts request networks simply by specifying the desired node types. The system then enumerates the possible network definitions, from which an analyst can choose.

We describe the steps of this process in the following subsections. First, we construct a *linking graph* that models the foreign key relations among tables. In response to user queries (i.e., desired node types), we then run a search algorithm over this graph to identify valid *linking paths*. Linking paths are then translated into relational algebra statements for extracting network edge tables.

5.1 Linking Graph Construction

To aid network definition, Orion builds a *linking graph*: a data structure that supports user queries over possible network models. Nodes within a linking graph correspond to data table fields (columns); primary key fields are assumed to represent a specific node type. Edges in the graph represent relationships among fields (e.g., foreign key references) that might be used to define a network among node types. Given input schemas for a set of data tables, Orion constructs a directed graph containing three types of edges:

1. *Key reference* (R) edges link all primary and foreign keys representing the same node type.

2. *Intra-table* (T) edges link all foreign keys within a table. The edges represent potential linking paths between node types.

3. *Conjugate* (C) edges link a foreign key F of one node type to a primary key P of a different node type if and only if the table containing F has an additional foreign key that references P.

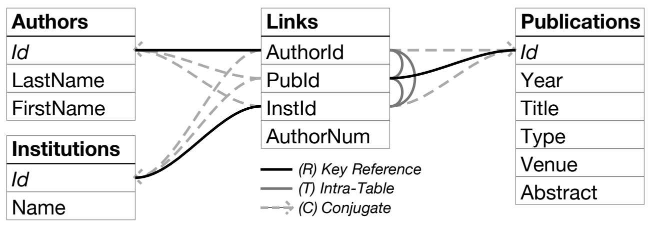


Figure 3: Schema and linking graph for publications data. Primary keys are italicized. Links are styled according to the edge type; links without arrows are bi-directional (R & T edges). The graph is a data structure for finding all networks involving a pair of node types.

Query: $Authors.Id \times Publications.Id$ (ties between people and papers)

$$Authors.Id \xrightarrow{R} Links.AuthorId \xrightarrow{T} Links.PubId \xrightarrow{R} Publications.Id$$
$$\Rightarrow \pi_{AuthorId,PubId}(Links)$$

Query: $Authors.Id \times Authors.Id$ (social ties between people)

$$Authors.Id \xrightarrow{R} Links.AuthorId \xrightarrow{T} Links.PubId \xrightarrow{C} Authors.Id$$
$$\Rightarrow \pi_{AuthorId,PubId}(Links) \bowtie_{PubId=PubId} \pi_{PubId,AuthorId}(Links)$$
$$Authors.Id \xrightarrow{R} Links.AuthorId \xrightarrow{T} Links.InstId \xrightarrow{C} Authors.Id$$
$$\Rightarrow \pi_{AuthorId,InstId}(Links) \bowtie_{InstId=InstId} \pi_{InstId,AuthorId}(Links)$$

Figure 4: Example linking queries and results returned by Orion's search algorithm. Linking path edges are annotated by type: key reference (R), intra-table (T) or conjugate (C) edges. Paths are mapped to relational algebra statements to extract network edge tables.

While the first two edge types are straightforward, conjugate edges merit further explanation. These edges represent paths in which one can join a linking table with itself to form a unipartite graph — a process analogous to multiplying a bipartite adjacency matrix by its transpose. Key reference (R) and intra-table (T) edges are bidirectional; conjugate (C) relationships are unidirectional, from a foreign key to a primary key with a different node type. Figure 3 shows the schema and linking graph for publication data extracted from the ACM digital library.

5.1.1 Automated Key Finding

To facilitate accurate key assignments—and thus accurate linking graph models—Orion includes mechanisms to automatically infer single-column primary and foreign key relations. To identify primary keys, the algorithm finds columns with distinct elements in each row and then ranks the candidates according to data type (e.g., integers are preferred to strings or dates) and position (left-most columns — those with a minimal index position — are preferred). The top-ranking result for a table is then chosen as the key, though within the interface users are free to override this choice.

For a selected primary key, Orion identifies candidate foreign keys by first finding all table columns with a matching data type. It then scores each candidate using a classifier based on the following features, where P is the primary key column, F the candidate foreign key column, $dist$ returns a set of distinct column values, and lcs returns the longest common subsequence within two strings:

$$f_a = \frac{|dist(P) \setminus dist(F)|}{|dist(P)|} \qquad f_b = |dist(P) \setminus dist(F)|$$
$$f_c = \frac{|\{i : F_i \in dist(P)\}|}{|F|} \qquad f_d = \frac{|lcs(name(P), name(F))|}{max(|name(P)|, |name(F)|)}$$

In other words, each candidate is classified using features concerning (a,b) how many distinct primary key values occur in the candidate column, (c) how many elements in the candidate column occur in the primary key column, and (d) the similarity of the pri-

Algorithm 1 FindPaths(*source*, *target*)

```
$maxOccurrences \leftarrow source = target\ ?\ 3\ :\ 1$
$paths \leftarrow \{\}$
$queue \leftarrow new\ Queue([source])$
**while** $queue$ is not empty **do**
    $path \leftarrow dequeue(queue)$
    $len \leftarrow length(path)$
    $t \leftarrow len < 2\ ?\ null\ :\ previous(path)$
    $u \leftarrow current(path)$
    **for all** $e \in edges(u)$ **do**
        $v \leftarrow v : v \in e \wedge v \neq u$
        $a \leftarrow |\{n \in path : sameBase(n,v)\}| > maxOccurrences$
        $b \leftarrow sameBase(t,u,v) \wedge (len > 2 \vee \neg sameTable(u,v))$
        $c \leftarrow type(e) \neq R \wedge sameTable(t,u,v)$
        $d \leftarrow type(e) = C \wedge \neg(sameTable(t,u) \wedge sameBase(t,v))$
        **if** $\neg(a \vee b \vee c \vee d)$ **then**
            $newPath \leftarrow append(copy(path), v)$
            **if** $v \neq target$ **then**
                $enqueue(queue, newPath)$
            **else if** $reverse(newPath) \notin paths$ **then**
                $paths \leftarrow paths \cup \{newPath\}$
            **end if**
        **end if**
    **end for**
**end while**
**return** $path$
```

mary key and candidate column names. We trained a logistic regression classifier on a corpus of test data, including all examples in this paper. We have not run a comprehensive study, but currently achieve perfect classification accuracy for tests on our corpus.

5.2 Linking Path Search

Given a linking graph and desired *source* and *target* fields (node table primary keys), Orion searches the graph to identify valid linking paths. These paths can be translated into relational algebra statements (e.g., projections and joins) to create a network edge table.

Orion's path-finding method (Algorithm 1) performs a breadth-first traversal starting from the *source* field. The traversal algorithm allows repeated visits, but greedily prunes the search at each step by testing the validity of candidate path segments. For a given path segment *path*, we denote the most recently added field by u, the previously added field by t, and a newly encountered candidate field by v. All fields have a corresponding *base* field indicating the node type: primary key fields reference themselves, while foreign key fields reference a primary key. With these definitions in place, valid paths are defined by the following conditions, which roughly correspond to the boolean variables a, b, c, d within Algorithm 1:

1. Excluding *source* and *target* fields, paths can not contain the same *base* more than twice.
2. The same *base* field can not occur three times consecutively unless u and v are in the same table.
3. Three consecutive fields can not be from the same table, unless the third field is reached by a *key reference* edge (R).
4. A field reached through a *conjugate* edge (C) can not be added to a path unless (a) t and u are in the same table and (b) t and v share the same *base* field, which differs from that of u.

The algorithm returns a set of valid linking paths with which an analyst can define a network model. To simplify the results, the algorithm culls paths that are identical to a previously found path if reversed. In addition, Orion sorts the returned paths such that shorter paths with less variation in *base* field types are listed first.

5.3 Network Extraction

Once an analyst has selected a set of desired linking paths, Orion translates these paths into relational algebra statements that when evaluated provide a network edge table. Figure 4 provides examples of input queries and the resulting linking paths and relational algebra statements (using the data and linking graph in Figure 3).

Mapping paths to relational algebra is straightforward. In most cases, each pair of fields (ignoring *source* and *target* fields) maps to two columns of the same table, with adjacent pairs related by an equijoin on the shared inner field. Special cases include *conjugate* edges, for which an encountered pair is instead joined against itself, and "self-edges" within a table that result in an odd number of path elements (e.g., tree data with paths of the form $T.P \xrightarrow{R} T.F \xrightarrow{R} T.P$).

When an analyst selects a network definition, Orion executes the resulting queries and constructs an edge table with integer node indices (§3). Orion similarly turns filtering criteria specified in the user interface into relational selection predicates that are incorporated into the queries. If an analyst selects multiple linking paths, Orion will construct multiple edge tables. Orion then forms the union of these edge tables; analysts can further specify aggregation functions in the Orion user interface to control if and how multiple edge sets should be merged.

6 Orion Workflow Operators

In addition to transforming data, one goal of Orion is to enable the construction of editable and reusable analysis workflows. These workflows are realized in a declarative language incorporating both relational operators and network statistics. By mapping user interface actions into statements in this language, Orion supports not only data manipulation and visualization, but can also export reusable scripts that keep a record of data provenance. In this section we describe the operations supported by our language and how they can be used to model and analyze network data.

Each Orion workflow task is an operator accepting one or more named parameters. These operators modify a *data set*: a collection of named tables. A workflow is simply a sequence of tasks. Figure 5 shows an example workflow for the case study in §7.1. Orion externally represents workflows using a simple XML format.

6.1 Data Import and Export

Orion's input / output operators read in data from different sources and write results in a variety of formats. The ***read*** operator imports data tables from external sources such as delimited text files (e.g., comma- or tab-separated values), relational databases, and GraphML or Dot (GraphViz) network files. Network files are translated into tables, typically one node table and one edge table. For delimited text files, Orion infers column data types based on their contents. The ***write*** operator writes either individual tables or extracted networks to a database or files in these same formats.

6.2 Schema Modification

Orion includes a handful of operators for modifying table schemas. The ***rename*** operator renames tables or individual columns. The ***key*** operator indicates that a column serves as a primary key, while the ***references*** operator assigns foreign key relations. Foreign key relations are particularly important, as they are used as the basis for determining feasible linking paths. While tables pulled from relational databases often have the appropriate key relations defined in their schemas, data in common formats such as CSV regularly lack this metadata. To add this metadata, our key finding algorithm (§5.1.1) generates a sequence of *key* and *references* statements.

6.3 Table Transformation

To aid the creation of network models, Orion provides operators for manipulating single tables. The ***remove*** operator simply drops a column or table. The ***derive*** operator allows analysts to create new columns by writing a user-defined function over existing columns. Orion currently accepts user-defined functions as snippets of Java code that are dynamically compiled at run-time.

6.3.1 Ranking Table Rows

The ***rank*** operator adds a new column containing rank-ordered integer indices. *Rank* statements must include sorting criteria: one or more fields to sort in ascending or descending order. The sort order determines a set of non-repeating indices. Statements can also take group-by fields; each group is then rank-ordered separately. The *rank* operator can be used to create indices enabling nuanced join predicates (e.g., the *post_id* field used in the scenario of §4.1).

6.3.2 Promoting Column Values to Node Tables

Tabular data often contains implicit linking relationships via values embedded in a column. For example, a single table of research grant awards might contain fields for a principal investigator (PI), a co-PI, and the institutions of each. From this data one may wish to form social networks of researchers and/or institutions. To help model such networks explicitly, Orion provides the ***promote*** operator. Given one or more field names, the *promote* operator first identifies and counts all distinct values in those columns and then populates a new table with the schema $(id, value, count)$. Values in the input table are replaced with foreign key references to the new "promoted" table. This operation allows analysts to extract implicit node types into explicit node tables. Returning to the grants example, an analyst might *promote* the PI and co-PI columns to create a new node table for people; the original table now serves as a linking table defining a network among people nodes.

6.4 Network Modeling

At the core of the Orion language are network creation operators.

6.4.1 Network Definition

Given a set of *linking paths* as input, the ***link*** operator extracts a network (edge table) according to the process described in Section 5. All input linking paths must have the same starting and ending fields. If multiple linking paths are provided, the *link* operator will construct a single edge table that is the union of the per-path edge tables. The resulting table includes a *path_id* column indicating which linking path generated a link. If an optional aggregation parameter is provided, the operator will generate a final edge table by applying an aggregate function over a group-by of the *source* and *target* columns; the aggregate values then become the edge weights.

The *link* operator also accepts optional projection and filtering parameters. Projection parameters consist of node or linking table columns to include across joins (as if included in a SQL *SELECT* clause). Filter parameters are name-value pairs of table names and predicate functions. Both single-table predicates (for filtering either node or linking tables) and two-table predicates (for join predicates on linked tables) are accepted. The link operator also accepts boolean parameters for suppressing self-links in unipartite graphs and for ensuring that only distinct edges are considered in linking tables. The latter enforces pre-aggregation of linking tables.

6.4.2 Sub-Network Extraction

Once a network has been created, Orion provides operators for extracting sub-graphs. For example, an analyst may want to compare time-slices of an evolving network or various ego-centric networks extracted from a larger social graph. The ***filter*** operator creates a filtered edge table based on a set of edge and node predicates. Edges are removed if a node predicate returns false for any incident node.

Given a set of "focus" nodes, the ***subgraph*** operator returns a subgraph containing all edges within a specified minimum distance. Orion measures graph distance by counting hops or summing edge weights. In the future, we plan to also support the degree-of-interest extraction method introduced by van Ham and Perer [31].

```
Workflow w = new Workflow();
w.add(Tasks.read("ohc")       // load data file
  .file("ohc.csv")
  .type("csv"));
w.add(Tasks.promote("users")  // promote username field
  .from("ohc")
  .select("username"));
w.add(Tasks.promote("forums") // promote forum field
  .from("ohc")
  .select("forum"));
w.add(Tasks.link("cp") // create forum × forum network
  .path("forums.id", "ohc.forum",
        "ohc.username", "forums.id")
  .distinct(true));
w.add(Tasks.stat("cp") // calculate edge weight deviance
  .field("dev")
  .stat("edgeWeightDeviance"));
```

Figure 5: An example Orion workflow definition for the online health case study in §7.1. Here the workflow is shown as Java code; workflows can also be persisted using a simple XML format.

Orion also provides ***iterators*** that enable repeated invocation of an operator over a sequence of parameter settings. Iterators are useful for performing batch operations, such as repeated filtering or subgraph extraction. Iterators can implement *split* operations to segment networks according to categorical or numerical fields. For numerical data columns, analysts can choose to split a network using a sliding window (e.g., to create separate time slices of a network) or an anchored window (e.g., to show the evolution of a network over time) while specifying the bounds of interest. Iterators also enable batch statistics calculation (see §6.5).

6.4.3 Network Aggregation

At times an analyst will be more interested in the aggregate properties of a graph than in leaf-node details; given a social network, she may wish to view the aggregate connections among genders or cities. The ***rollup*** operator aggregates edges according to specified properties of the nodes (c.f., [33, 32]). The rollup operator generates an aggregate edge table and node tables for the node attributes.

6.5 Network Analysis

Orion additionally provides network analysis algorithms. The ***stats*** operator computes one or more statistics of a network and stores the resulting values in the corresponding node or edge tables. The statistics operator is modular, allowing user-defined functions to be added to the workflow language. Currently these functions must be written in the Java programming language and conform to a provided interface definition. Supported statistics include *in-degree*, *out-degree*, *betweenness centrality*, *eigenvector centrality*, *clustering coefficient*, *edge weight asymmetry*, *edge weight deviance*, *community identification*, and *linkage-based sorting*.

While many of these metrics are common to social network analysis, a few deserve special mention. The edge weight asymmetry and deviance measures are inspired by van Ham et al.'s Honeycomb [32] system. The former is simply the logged ratio of edge weights between corresponding anti-parallel edges in a directed graph. The latter calculates the amount an edge weight deviates from the expected value if one assumes a uniform random distribution of total weight across the cells of the adjacency matrix. Deviance can help identify edges with unexpectedly strong or weak strengths, particularly in dense aggregated networks.

Community identification is performed via a greedy hierarchical clustering optimizing Newman's modularity metric [25]. Linkage-based sorting provides an integer sort order of nodes that attempts

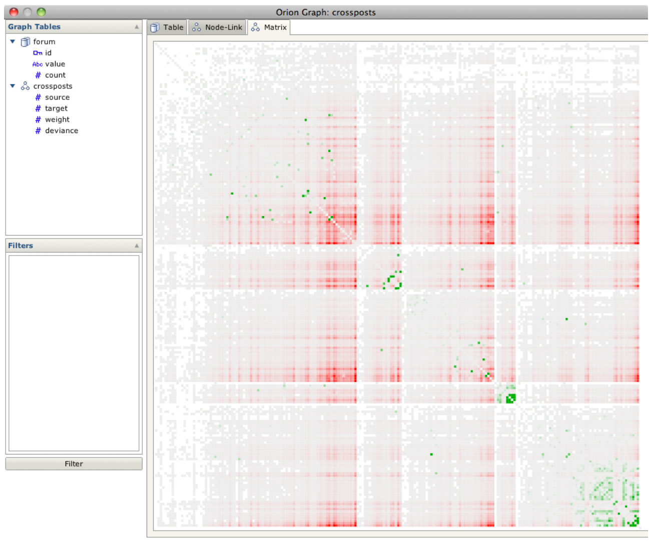


Figure 6: Matrix view of connections between online health forums. Edges are weighted by the number of distinct cross-posters. The cells are then colored according to edge weight deviance.

to minimize the distance among connected nodes. We approximate this objective by seriating the nodes using the cluster tree constructed by the community identification algorithm (c.f., [34]). The resulting ordering is particularly useful for visualization purposes, such as permuting the rows and columns of an adjacency matrix diagram to reveal clusters (c.f., [13]).

The ***layout*** operator is similar to the *stats* operator, put instead computes spatial coordinates for subsequent layout in a visualization. This operator currently supports force-directed layout only.

6.6 Summary

In summary, the transformations supported by Orion are realized in a declarative workflow language. Saved Orion sessions are simply XML-serialized versions of this workflow, and so can easily be edited or reviewed directly in a text editor. While the Orion interface enables rapid specification of these workflows, we have also found that programmatic use of the workflow language (as in Figure 5) has greatly aided data analysis in our research groups.

7 Case Studies

We now present a collection of case studies illustrating how Orion has been applied to conduct network analyses in multiple domains.

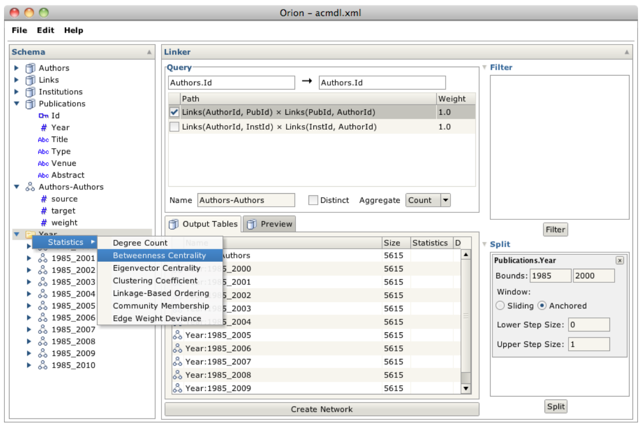
7.1 Online Health Communities

The scenario in §4.1 introduced an analysis of online health communities. In addition to comparing the social networks of individual forums, our collaborating analyst is also interested in exploring the connections between communities. Might cross-posting behavior provide insights into the comorbidity of medical conditions?

To assess such questions, the analyst generates a network in which the nodes correspond to discussion forums and edge weights indicate the number of distinct users who have posted in both forums. To construct this network with Orion, the analyst promotes both the *username* and *forum* fields to node tables. The analyst then requests a network with *forum* nodes as both the source and target. Orion suggests the desired result: linking forums by shared users.

The analyst then runs the edge weight deviance statistic to calculate the degree to which edge weights vary from the expected value (assuming a uniform random distribution). The resulting matrix diagram is shown in Figure 6, with cells colored by deviance.

By inspecting both this matrix view and the sorted edge table, the analyst has flagged a number of unexpected connections. Some



Authors.Id → Authors.Id

	Path	Weight
✓	Links(AuthorId, PubId) × Links(PubId, AuthorId)	1.0
	Links(AuthorId, InstId) × Links(InstId, AuthorId)	1.0

Name: Authors-Authors Distinct: ☐ Aggregate: Count

Name	Size	Statistics	D
uthors	5615		
5_2000	5615		
5_2001	5615		
5_2002	5615		
5_2003	5615		
5_2004	5615		
Year:1985_2005	5615		
Year:1985_2006	5615		
Year:1985_2007	5615		
Year:1985_2008	5615		
Year:1985_2009	5615		

Publications.Year
Bounds: 1985 2000
Window: Sliding [✓ Anchored]
Lower Step Size: 0
Upper Step Size: 1

Figure 7: Using Orion to subdivide an ACM co-authorship network by publication date. The Split region on the lower right provides controls for defining a filtering window; the Preview pane in the center lists all resulting tables. When the networks are created, the Schema Viewer on the left groups the results to support batch statistics calculation.

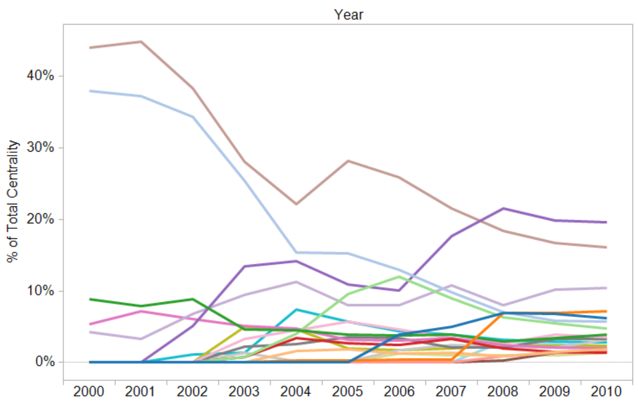


Figure 8: Time-sliced betweenness centrality scores for researchers in the ACM digital library. Centrality scores are normalized per year. The data was generated in Orion and then exported to Tableau.

connections appear to indicate possible data errors; for example, the hearing loss forum has unexpectedly strong connections to many other forums. Other strong connections indicate interesting co-occurrences (e.g., cold/flu and stress, ear/nose/throat problems and heart disease) or common misdiagnoses (e.g., lupus and lyme disease). Orion has enabled her to make these observations in a matter of minutes. The analyst is now following up on these results, for example by correlating them with external comorbidity data.

7.2 Academic Production and Collaboration

We are also using Orion to explore academic production and collaboration networks; for example, §5.1 discusses data extracted from the ACM Digital Library. To inspect the career progress of computer scientists, we use Orion to construct a social network based on co-publication. Using Orion's subgraph extraction facilities (Figure 7), we define social networks over increasing periods of time (e.g., first all publications up to 2000, then 2001, etc). We then batch compute betweenness centrality scores for each extracted network. As Orion enables easy data export, we subsequently loaded the data into Tableau for further analysis, leading to the plot in Figure 8.

Orion's flexibility also enables assessment of other models. For example, we have constructed the network of all researchers who have published in the same venue (by promoting and linking on the publication venue) within the same year (by specifying a join predicate enforcing matching years).

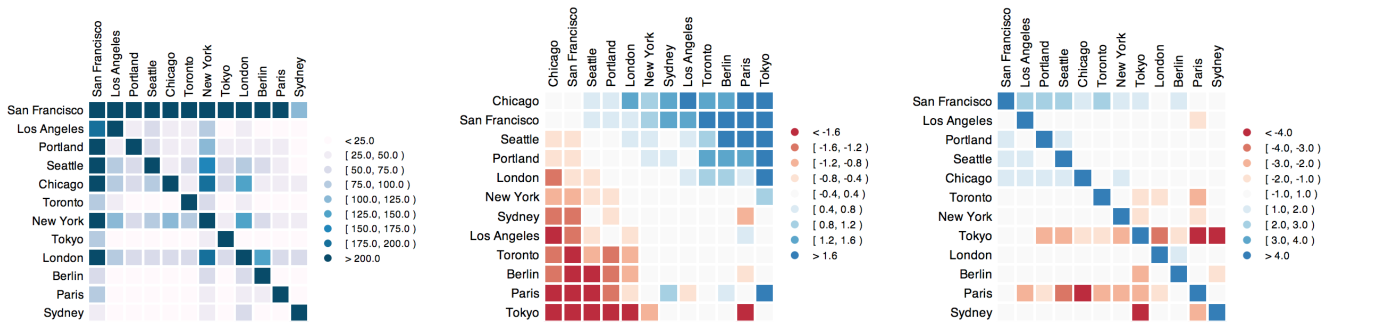


Figure 9: Matrix diagrams resulting from an Orion analysis of GitHub, a hosting service for open-source software. (a) Raw counts of follower links between cities, sorted by geographic proximity (column nodes "follow" row nodes on GitHub). (b) Follower links colored and sorted by asymmetry. (c) Followed links colored by deviance from expected value, sorted geographically.

The ACM publication data contributes to a larger analysis initiative with social scientists at the first author's university. The scientists are studying academic collaboration and have collected multiple data sets indicating links among university faculty. In addition to publication databases, the data include department and PhD committee memberships and co-PI relations on grants. These heterogeneous edge sets can be combined and weighted in any number of ways to form a collapsed network. We are using Orion's edge aggregation features to create and compare network models built from heterogeneous linking data.

7.3 Software Development on GitHub

Finally, we have used Orion in collaboration with computer scientists studying global development patterns in open-source software. The data under investigation comes from GitHub, a web service that hosts open-source projects. Using the GitHub web API, the researchers have collected over 1,000,000 commits and 500,000 explicit "follower" connections among roughly 50,000 users. In addition, each user's location has been geocoded according to a self-reported location string and then mapped to near-by major metropolitan areas (see Heller et al. [12] for more details).

Using Orion, we can quickly generate and analyze networks extracted from this data. For example, we have constructed social networks based on commit history: a link is included between two users A and B if B makes a commit to the same repository immediately after A. We can specify this network in Orion by linking users via a table of commits. We have as input two tables: one for users and another for commits. The commit table includes the date, project name, and the user (as a foreign key). First we promote the project (repository) name to its own table, then link users according to a shared repository. We limit links to temporally adjacent commits by first applying a rank operation based on the commit date, and then adding a join predicate that ensures that only adjacent ranks are included in the resulting network.

We can also construct networks of "who follows whom" by linking users using a table of extracted follower relations. By requesting the user location attribute as a node type (rather than the user id), we construct an aggregated graph among major cities, with edge weights indicating the number of connections between users in those cities. We can then apply edge weight asymmetry and deviance statistics to examine differences among various locales. Figure 9 shows selected matrix views from this analysis (originally published in [12]). For example, in Fig. 9c we see that Paris and Tokyo each have many fewer incoming "followed" links than would be expected if links were assigned randomly, and that San Francisco consistently receives a surplus of "followed" links. While these particular images have been stylized for publication using Protovis [4], the underlying analysis can be performed completely within Orion.

8 Conclusion and Future Work

This paper introduces Orion, a system for interactive modeling, transformation, and visualization of network data. By providing a unified model, workflow language, and graphical user interface for iterative network manipulation, the construction and comparison of networks empower analysts to be more exploratory and flexible in their analysis. Through case studies involving online health communities, academic collaboration networks, and global software development, we demonstrate how Orion supports the visual analysis of multidimensional heterogenous networks.

While our case studies illustrate how Orion can be applied to real-world analysis tasks, each study was conducted in the context of a collaboration between the analysts and ourselves. A necessary next step is to evaluate how analysts use Orion without external assistance. User studies with representative tasks and participants would certainly help surface usability issues and inform iterative design. However, we believe the most important test will come from analysts independently applying Orion in their own work.

As analysts gain the flexibility to create new models and transformations of network data with Orion, a critical need arises for better methods to preview and compare the constructed networks. While Orion provides capabilities to support these tasks, we believe that providing even more sophisticated visual and statistical techniques to summarize the similarities, differences, trends, and outliers of the resulting networks is an area ripe for future research. Additionally, while Orion provides great power for analysts to model networks that match their hypotheses, the vast number of possibilities to construct a network may seem daunting. An interactive visual representation of Orion's linking graph may assist users in understanding and specifying network models. Future work might also provide users with smarter automatic suggestions to help uncover networks models with interesting and meaningful patterns.

Acknowledgements

The authors thank Diana MacLean, Sanjay Kairam, Brandon Heller, Eli Marschner, and MedHelp for sharing data, and thank the anonymous reviewers for their suggestions. The first author was supported by NSF Grant CCF-0964173 and The Boeing Company.

References

[1] E. Adar. GUESS: a language and interface for graph exploration. In *ACM CHI*, pages 791–800, 2006.

[2] C. Aggarwal and H. Wang, editors. *Managing and Mining Graph Data*. Springer, 2010.

[3] V. Batagelj and A. Mrvar. Pajek – program for large analysis. *Connections*, page 47, 1998.

[4] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE Trans Vis & Comp Graphics*, 15(6):1121–1128, 2009.

[5] M. Freire, C. Plaisant, B. Shneiderman, and J. Golbeck. ManyNets: an interface for multiple network analysis and visualization. In *ACM CHI*, pages 213–222, 2010.

[6] G. W. Furnas. Generalized fisheye views. In *ACM CHI*, pages 16–23, 1986.

[7] Gephi. `http://gephi.org/`.

[8] M. Ghoniem, J.-D. Fekete, and P. Castagliola. On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis. *Information Visualization*, 4:114–135, July 2005.

[9] H. He and A. K. Singh. Graphs-at-a-time: Query language and access methods for graph databases. In *ACM SIGMOD*, pages 405–417, 2008.

[10] J. Heer and M. Agrawala. Software design patterns for information visualization. *IEEE Trans Vis & Comp Graphics*, 12(5):853–860, 2006.

[11] J. Heer and M. Bostock. Declarative language design for interactive visualization. *IEEE Trans Vis & Comp Graphics*, 16(6):1149–1156, 2010.

[12] B. Heller, E. Marschner, E. Rosenfeld, and J. Heer. Visualizing collaboration and influence in the open-source software community. In *Mining Software Repositories*, pages 223–226, 2011.

[13] N. Henry and J.-D. Fekete. Visually exploring large social networks. In *INTERACT*, pages 604–610, 2007.

[14] N. Henry, J.-D. Fekete, and M. J. McGuffin. NodeTrix: a hybrid visualization of social networks. *IEEE Trans Vis & Comp Graphics*, 13:1302–1309, November 2007.

[15] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Trans Vis & Comp Graphics*, 6:24–43, January 2000.

[16] D. Huynh and S. Mazzocchi. Google Refine. `http://code.google.com/p/google-refine/`.

[17] igraph. `http://igraph.sourceforge.net/`.

[18] Jung. `http://jung.sourceforge.net/`.

[19] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *ACM CHI*, pages 3363–3372, 2011.

[20] H. Kang, L. Getoor, B. Shneiderman, M. Bilgic, and L. Licamele. Interactive entity resolution in relational data: A visual analytic tool and its evaluation. *IEEE Trans Vis & Comp Graphics*, 14(5):999–1014, 2008.

[21] H. Kang, C. Plaisant, B. Lee, and B. B. Bederson. NetLens: iterative exploration of content-actor network data. *Information Visualization*, 6(1):18–31, March 2007.

[22] B. Lee, M. Czerwinski, G. Robertson, and B. B. Bederson. Understanding research trends in conferences using PaperLens. In *ACM CHI Extended Abstracts*, pages 1969–1972, 2005.

[23] J. Leskovec. SNAP. `http://snap.stanford.edu/`.

[24] Network Workbench. `http://nwb.slis.indiana.edu/`.

[25] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, 69(6), 2004.

[26] A. Perer and B. Shneiderman. Systematic yet flexible discovery: guiding domain experts through exploratory data analysis. In *Intelligent User Interfaces (IUI)*, pages 109–118, 2008.

[27] B. Shneiderman and A. Aris. Network visualization by semantic substrates. *IEEE Trans Vis & Comp Graphics*, 12:733–740, 2006.

[28] M. A. Smith, B. Shneiderman, N. Milic-Frayling, E. Mendes Rodrigues, V. Barash, C. Dunne, T. Capone, A. Perer, and E. Gleave. Analyzing (social media) networks with NodeXL. In *Communities and Technologies (C&T)*, pages 255–264, 2009.

[29] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Trans Vis & Comp Graphics*, 8(1):52–65, January 2002.

[30] UCINET. `http://www.analytictech.com/ucinet/`.

[31] F. van Ham and A. Perer. "Search, Show Context, Expand on Demand": Supporting large graph exploration with degree-of-interest. *IEEE Trans Vis & Comp Graphics*, 15:953–960, November 2009.

[32] F. van Ham, H.-J. Schulz, and J. M. Dimicco. Honeycomb: Visual analysis of large scale social networks. In *INTERACT*, pages 429–442, 2009.

[33] M. Wattenberg. Visual exploration of multivariate graphs. In *ACM CHI*, pages 811–819, 2006.

[34] L. Wilkinson. *The Grammar of Graphics*. Springer-Verlag, Secaucus, NJ, 2005.