Origraph: Interactive Network Wrangling

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

University of Utah

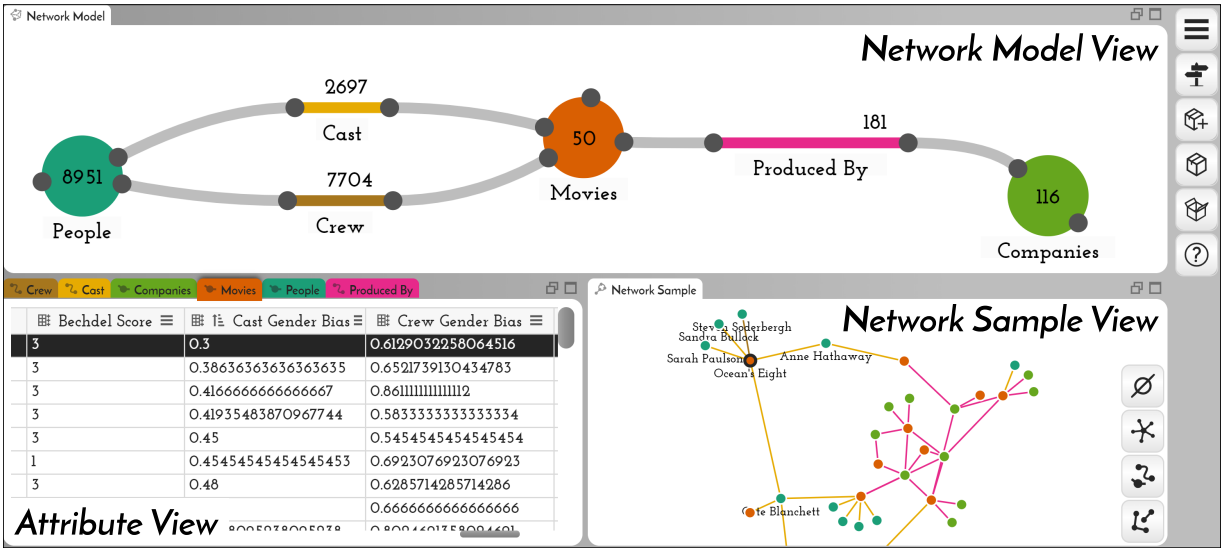

Figure 1: Overview of the Origraph UI. The **network model view** shows relationships between node and edge classes and is the primary interface for operations related to connectivity. The **attribute view** shows node and edge attributes in a table and is the primary interface for attribute-related operations. The **network sample view** visualizes a preview of the current state of the network.

Abstract

Networks are a natural way of thinking about many datasets. The data on which a network is based, however, is rarely collected in a form that suits the analysis process, making it necessary to create and reshape networks. Data wrangling is widely acknowledged to be a critical part of the data analysis pipeline, yet interactive network wrangling has received little attention in the visualization research community. In this paper, we discuss a set of operations that are important for wrangling network datasets and introduce a visual data wrangling tool, Origraph, that enables analysts to apply these operations to their datasets. Key operations include creating a network from source data such as tables, reshaping a network by introducing new node or edge classes, filtering nodes or edges, and deriving new node or edge attributes. Our tool, Origraph, enables analysts to execute these operations with little to no programming, and to immediately visualize the results. Origraph provides views to investigate the network model, a sample of the network, and node and edge attributes. In addition, we introduce interfaces designed to aid analysts in specifying arguments for sensible network wrangling operations. We demonstrate the usefulness of Origraph in two Use Cases: first, we investigate gender bias in the film industry, and then the influence of money on the political support for the war in Yemen.

Keywords: Graph visualization; Data abstraction; Data wrangling

Index Terms: Human-centered computing [Information visualization]: ;— [Human-centered computing]: Visualization systems and tools—; Information systems [Graph-based database models]: —

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

1 Introduction

Data wrangling—which includes cleaning data, merging datasets, and transforming representations—is known to be a tedious and time-consuming part of data analysis [25]. Historically, wrangling was done with scripting languages such as Python, Perl, and R, or manipulation in spreadsheet tools, requiring significant computational skills. More recently, a new generation of interactive data wrangling tools instead uses visualization, interactive specification of rules, and machine learning to improve the efficiency and scale of data manipulation tasks while also providing accessibility to a broader set of analysts [26, 61, 64].

These powerful, interactive data wrangling tools, however, ignore a data type that is increasingly important [6]: networks. Whereas some datasets inherently represent a network that exists in the physical world, such as the connections between neurons in a brain or roads between cities, many other datasets also benefit from a network representation during analysis. The influence of social connections on obesity rates [10], the spread of information via a digital media platform [3], or the evolution of sticky feet of geckos [18] are but a few examples.

In such cases, an analyst has one, or possibly many, mental models of the data as a network. However, source data rarely conforms to the way an analyst thinks about it. To model data as a network, analysts must wrangle the dataset, often starting with tabular or key-value data. Transforming data itself can lead to new hypotheses, and thus a new network representation of the data. Also, new tasks often necessitate new data abstractions [40]. It stands to reason that the ability to rapidly and easily transform network data can foster creative visualization solutions and simplify both exploration and

communication of the key aspects of a dataset.

Existing network wrangling tools, most notably Ploceus and Orion [21, 33], focus on creating an initial network model, but no tools yet exist to iteratively and interactively reshape the network model itself with operations such as converting between nodes and edges [41]. Other operations that leverage edges, such as connectivity-based filtering [8], can currently be performed only with code. Consequently, the steep learning curve of programming languages creates an unnecessary barrier that prevents many analysts from wrangling their own networks.

In this paper, we introduce Origraph, our primary contribution (see Figure 1). Origraph is a visual, interactive network wrangling tool that implements an expanded set of wrangling operations that allow analysts to model and reshape networks from input data in various forms. The goal of Origraph is to allow analysts to translate their data into the network representation that is most suited to answer their analysis questions and refine or revamp that representation as analysis questions change over time.

We also contribute a discussion of network wrangling operations, propose several new operations, and then classify them into a preliminary taxonomy. The elicitation of these operations is grounded in a literature analysis, an analysis of user needs in prior projects, eliciting missing capabilities in existing tools, and our own experience in wrangling network data. Origraph implements all operations we identified. Operations that are unique to Origraph are concerned with introducing new nodes, edges, or attributes based on leveraging network structures and multivariate attributes simultaneously. For example, Origraph supports an operation to introduce edges between two nodes if these nodes are connected by a path with specific properties.

Origraph includes visualizations of the data that support analysts in their reshaping and analysis process. Dedicated views communicate the state of the network, the attributes of the nodes and edges, and a sample of the network as currently modeled. Specialized views and algorithms support analysts in making network wrangling decisions. Origraph is web-based and open-source. A prototype is available at `https://origraph.github.io/`.

We designed Origraph with a diverse audience in mind: from data journalists who analyze bot networks on social media, to social scientists who investigate the spread of specific terms in political circles, to biologists who study protein interaction. We do not expect users of Origraph to be able to program. Some advanced functionality is made available for more skilled analysts, such that they can write expressions slightly above the level of formulas in spreadsheet software to perform sophisticated filtering and aggregation. At the same time we believe that Origraph can also speed up the wrangling process of skilled programmers.

We validate Origraph in two complex network modeling use cases. First, we reshape a movie dataset so that we can investigate gender biases in recent popular movies. In the second use case, we integrate data from various sources and build a network that allows us to investigate how money from donors could influence votes in the US Senate on issues related to the war in Yemen.

2 Related Work

Origraph draws on related work in graph editing, tabular data wrangling, graph databases, and network modeling. Graph editing refers to tools aimed at visualizing and editing existing networks. Tabular data wrangling tools work with tabular data and excel at data transformation and querying. Graph databases are optimized for storing, indexing, and querying large networks. Network modeling involves extracting a network from tabular data. We discuss relevant prior work in the respective subsections below. Various network visualization systems support basic operations, such as aggregating nodes into supernodes, or filtering nodes and edges. We limit our discussion here to tools and techniques that support more sophisticated wrangling operations and refer to review articles [35, 43] for a survey of more general network visualization methods.

2.1 Graph Editing

A wide array of tools are designed to allow users to visualize and edit networks and their associated attributes. Tools such as Cytoscape [55] and Gephi [5] focus on the topology of the graph and offer several editing features. Similarly, Graphviz [12] is a collection of graph drawing tools, including layout programs and customizable graph editors.

Tools in this space allow users to modify a network, mainly by creating or deleting nodes and edges. This level of editing is useful for tasks such as finding and correcting mistakes in the data or inputting new data. However, these edits are not based on rules; rather they are limited to the instance level and hence do not generalize to the entire network. Filters are a common exception to this restriction: most network visualization tools support filtering nodes or items.

Generally, graph editing tools assume a well-defined network as input. These tools are primarily designed to represent network models as they exist, and do not have features aimed at creating or deriving new models. The Tulip framework [1] touches on network modeling by enabling users to import data and generating multiple data models for users to visualize and explore. However, users do not have control over how these data models are generated, nor can they modify them.

2.2 Wrangling Tabular Data

Data wrangling applications for tabular data include tools such as Google Refine [23], Data Wrangler [26] and its commercial successor Trifacta Wrangler [61], and Microsoft Excel, which focus on data transformation and cleanup. These systems enable analysts to reformat input data to best suit their analysis tasks, but they are not designed to support network data.

Some data wrangling tools use network visualization in the process of wrangling data. D-Dupe [9] uses a network perspective to help resolve duplicate entities while cleaning a dataset. Schema Mapper [54] uses network visualizations to explore how one hierarchical dataset maps to another. GraphCuisine uses a visual interface to generate random networks [2]. Although these approaches utilize network visualization in the wrangling process, and support wrangling tasks on non-tabular data, they do not wrangle networks themselves.

NodeXL [56], an extension to Excel, allows users to import, visualize, and transform network data. However, NodeXL provides only a minimal set of network transformation features. Moreover, Liu *et al.* [33] point out that conducting extensive network reshaping with NodeXL would require users to be Excel experts.

2.3 Scripting for Graph Wrangling

Another domain of related work is network-specific technologies ranging from libraries such as NetworkX [17] and mully [19] to graph databases such as neo4j [24], OrientDB [46], and GraphDB [45]. Similar to Origraph, these approaches allow users to create and transform network models as needed. However, creating or changing a network model in such systems must be done through scripting, requiring users to be proficient in the context-specific language. In contrast, Origraph does not require programming for most operations and allows users to create and transform network models with an interactive interface.

2.4 Network Modeling

Network modeling, i.e., the concept of creating networks from tabular data, has been explored by several tools. The need for these tools arises from the frequent storage of network data as tables, containing a list of items and their associated attributes. These

tools commonly also provide visualization to better understand and explore the resulting network structure.

Commercial systems such as TouchGraph Navigator [60] and Centrifuge [58] support network modeling by allowing users to create attribute relationship graphs from tabular data. Attribute relationship graphs, which were introduced by Weaver [66], refer to graphs where attributes are connected based on co-occurrence. A related approach, used by both PivotGraph [65] and HoneyComb [63], generates new network models by aggregating nodes that have a certain attribute.

The two systems most related to Origraph are Orion [21] and Ploceus [33]. Both systems model attribute relationship graphs based on tables from relational databases. Ploceus's approach is based on relational algebra whereas Orion uses relational tables as its base and represents networks as edge tables. Although Orion and Ploceus are well suited for creating networks from tabular data, they do not support reshaping existing networks. The ability to iterate on chosen data abstractions is restricted to creating a new network model from the raw data. Each distinct network model needs to be specified from the ground up; users cannot create alternative network models with an existing model as a starting point. Both Orion and Ploceus do support node attributes, but unlike Origraph, do not support edge attributes beyond simple edge weights. We discuss differences and similarities between Origraph and these two tools in more detail in Section 10.

Another related modeling tool is Graphiti [57], which uses demonstration-based interaction to create edges in networks. Graphiti infers possible data abstractions from instance-level operations. Although Graphiti supports modifying existing networks by adding new edge types, it does not support more extensive reshaping operations such as changing nodes to edges or deriving new classes from attributes of connected nodes or edges.

Ultimately, Origraph builds on existing work by extending the concept of network modeling and providing powerful network reshaping features. In this vein, Origraph offers support for edge attributes, more expressive network modeling features such as faceting and deriving reduced attributes, and annotation features. Our work differs from other systems by giving users the ability to leverage existing network models to create more expressive abstractions for node and edges that best reflect their semantic understanding of their data, and by enabling rich edge semantics that other systems cannot represent.

3 Terminology

Here we define terminology that we use to describe the operations, as well as the functionality of Origraph. In this work, we focus on modeling **networks** using **nodes** and **edges**. Both nodes and edges can have **attributes** associated with them. A **class** defines a common set of attributes for either nodes or edges: a **node class** defines a class of nodes, and an **edge class** defines a class of edges. We treat nodes and edges as fluid concepts that can change, and use the term **class** to generically refer to node and edge classes and **items** to refer to generic instances of nodes or edges. **Supernodes** are nodes representing a set of other nodes.

We use the term **network model** for the set of classes and their relationships. The network model describes relationships between types of nodes and edges on an abstract level and is independent from concrete instances of the classes, similar to how a database schema is independent from the specific data in the database. Concrete nodes and links make up the **network topology**. It is noteworthy that networks with trivial models (e.g., a single node class and a single edge class) can be arbitrarily large and complex.

Edges can be directed or undirected. Node and edge attributes can be numerical, ordinal, nominal, sets, or labels/identifiers. Attributes may also contain complex data structures such as nested hierarchies, lists, and objects.

4 Eliciting Operations

We developed our set of proposed operations through a reflective analysis process, in which there is a direct mapping from user tasks to operations. Connecting nodes, for example, is both a user task and a network wrangling operation. We use the term operation, as we frame operations from the perspective of what a network wrangling system should support. In our elicitation, we reflected upon published task lists for visual graph analysis [29, 43, 50]; related systems that support limited aspects of graph wrangling [8, 20, 32] our own experiences working with domain experts on graph analysis [8,27,31,36,42–44,47–49] and those of colleagues [13,41,51,65, for example]; and an informal task analysis that we conducted with new collaborators [3, 4, 28, 37, 38]. Through an iterative, dialogical approach among the co-authors that involved discussions, written summaries, and design sketches, we considered the myriad of literature and experiential sources in determining the range of operations for wrangling and reshaping networks that are, or could be, useful for analysts. The final list of operations represents our proposed vision of what should constitute a feature-rich system for fully supporting a flexible approach to using networks.

These operations could support a new approach to answering complex question like those we pose in Section 9, one that makes use of networks as a data representation that can be derived from various input data sources and flexibly reshaped as an analysis progresses and new questions come up.

As interactive network wrangling is a relatively new research area, the ambition of our work is to expand the vocabulary of operations, and to classify these operations in our preliminary taxonomy. Important future work will be to identify a formal taxonomy and to study which operations are useful for specific tasks and user group combinations.

5 Operations

We classify the operations we identified into three categories, which constitute our taxonomy: **modeling operations** that modify the network model and network topology by introducing new node and/or edge classes in a network; **item operations** that modify the network topology by removing or introducing items (instances of links or nodes); and **attribute operations** that manipulate the attributes of, or add new attributes to, existing classes.

We illustrate these operations with a simple movie network consisting of movies, actors, and roles with attributes such as genre, company, and gender. We introduce a more complex wrangling workflow with a real movie dataset in Section 9.1. We provide a comparison of which operations are supported in other systems, and which are unique to Origraph in Supplementary Table 1.

Although we do not claim that this list is exhaustive, we demonstrate that the combination of these operations extends the space of transformations currently possible with Orion and Ploceus [21, 33].

5.1 Modeling Operations

Modeling operations affect the network model by introducing or removing node or edge classes. Modeling operations also affect the network topology because they implicitly create new node and edge instances.

Connecting or disconnecting items is the most fundamental modeling operation. Connections can be established by leveraging the primary key / foreign key approach well known in the database literature, where each item of a tabular dataset has a unique ID, and another column has a foreign key pointing to the primary key of the same or another table. This is also a common way to store graphs in non-volatile memory: many network file formats store lists of nodes and lists of links between these nodes. We also consider cases where an item in a table stores a list of connection targets using foreign keys. More generally, connections can be derived from arbitrary attributes, for example by (partially) matching strings,

or by evaluating arbitrary functions on attributes. In the movie dataset, for example, we could introduce edges between movies that have a significant number of female actors to form a clique of gender-balanced movies.

Promoting attributes allows users to promote the unique values of an attribute column to a new class (Figure 2). In the movie example, we could take a column containing film genres and promote these to a separate "genres" class, while at the same time introducing edges between the new genre nodes and the movie nodes.

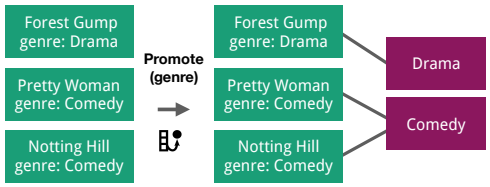

Figure 2: Promoting the genre attribute of the three movies. A new genre node class with two instances contains two nodes with the unique genres. The new nodes are connected to their source nodes.

Faceting slices a class based on the value of an attribute and creates new classes for each slice, as illustrated in Figure 3. An example is to facet the movie class on the genre attribute, which generates a new class for each unique genre containing only the movies of that genre.

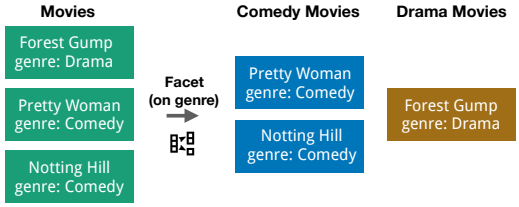

Figure 3: Faceting a movies node class based on genres. The facet operation results in two new node classes, one for "Comedy Movies" and the other for "Drama Movies".

Converting between nodes and edges transforms connected nodes into edges, or *vice versa*, retaining the connectivity and semantics of the network, as illustrated in Figure 4. For example, given actors connected to movies, we convert the movie nodes to edges, which results in a collaboration network between actors, where edges connect actors who have acted together in a movie.

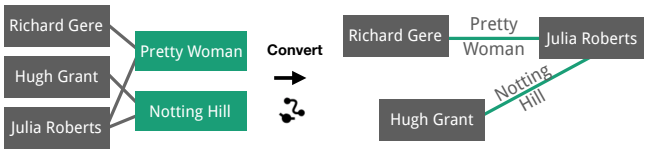

Figure 4: Converting nodes to edges. Here, movie nodes are converted to edges, resulting in a network linking co-actors.

Edge projection introduces an edge based on a path in the network model between nodes, as illustrated in Figure 5. The path can be specified with a set of rules leveraging the classes and the attributes of the network. For example, in an actor–role–movie–production-company network, edge projection can be used to connect an actor with the production-company. Another example is to project edges from an actor–movie–actor relationship to an actor–actor relationship, limiting edges to financially successful collaborations by considering only edges that transit through movies that had a box office return above a specified number.

Creating supernodes aggregates the information of multiple nodes into a single supernode that represents all the constituting nodes. For example, we could create a supernode that represents all comedy movies (Figure 6). There are different ways to realize a "create supernode" operation: one can retain all aggregated nodes and edges, or replace the aggregated nodes with the supernode. In Origraph, we retain all nodes and edges, which can be filtered out in a subsequent step.

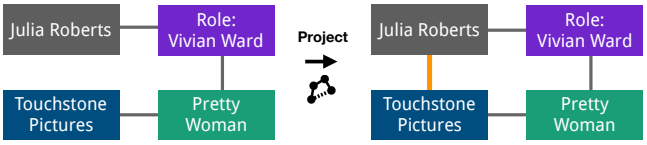

Figure 5: Projecting an edge. Paths connecting an actor to a role to a movie to a production company are projected to an actor–production-company edge (orange).

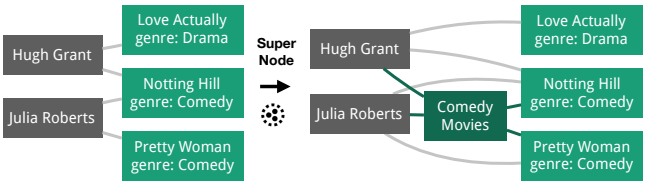

Figure 6: Creating a supernode. Notting Hill and Pretty Woman are combined into the supernode Comedy Movies. This supernode inherits all edges and also has edges to the aggregated movies.

Rolling up edges combines parallel edges, or multiple edges that connect the same two nodes, into a single edge of a new edge type. For example, in a co-appearance network of actors, some actors might have appeared together in multiple movies, represented by multiple edges. The rollup operation results in a single new edge connecting these actors.

5.2 Item Operations

Item operations change the number of items in existing classes. They may leverage the network model, but they do not modify it. Item operations affect the topology of the network, as they manipulate which nodes and edges exist.

Filtering by attributes removes nodes or edges based on the values of an attribute. For example, we could filter by removing all movie nodes that grossed less than $10 million, or remove all actor-movie edges where the role was not a speaking role.

Connectivity-based filtering describes filter operations on items (nodes or edges) that leverage the connectivity of a network. In the movie network, where actors are connected to movies they acted in, an example of connective filtering is to remove all actors who have never acted in a movie that grossed more than $100 million. Connectivity-based filtering can also leverage complex, multi-hop operations [8].

5.3 Attribute Operations

Attribute operations modify class attributes or create new ones, but they do not impact the network model or the network topology. They are, however, an important prior or subsequent step in many modeling operations.

Deriving in-class attributes leverages existing attributes in a node or edge class to derive new ones. For example, for actor nodes with birth and death year, we could derive the attribute "age", representing either the actor's current age, if they are alive, or their age at death.

Connectivity-based attribute derivation is concerned with deriving attributes for a node or edge class based on attributes in a possibly indirectly connected class (Figure 7). As an example, in an actor-movie network, we can compute a new attribute "gender bias" on the movie class by iterating through all the actors connected to each movie and dividing the number of men by the total number of actors in that movie. Connectivity-based attribute derivation can be very useful as a follow-up step to many modeling operations. For

example, when creating supernodes, it is often relevant to also aggregate some of the attributes of the original nodes into the supernode. Combined with modeling operations such as promoting attributes, edge projection, creating supernodes, or rolling up edges, this is the network version of the split-apply-combine (or map-reduce) strategy common in tabular data analysis [67]. As with tabular data, potentially relevant apply functions are immensely diverse.

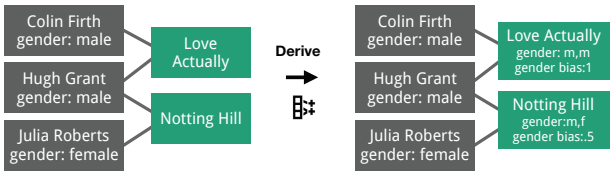

Figure 7: Deriving attributes across connected classes. Nodes of the movie class derive attributes from the adjacent actor class. In this case, the gender of the actors is used to calculate a gender bias score.

Changing edge directionality introduces directionality to, or removes it from, previously undirected edges, or changes the direction of edges. Like all other operations, changing edge direction is rule based. For example, in a network of movies and actors, edges between movies and actors can be made directional to represent an "acted in" relationship.

5.4 Housekeeping Operations

In addition to these data wrangling operations, a system implementing these operations will also need to enable a series of basic operations, such as importing data from various file formats, extracting nested file structures, exporting data for consumption by graph visualization tools, renaming classes and attributes, removing or hiding classes or attributes, etc. Since these are not specific to network wrangling, we do not discuss them in detail.

5.5 Discussion

Several of these operations are available in prior systems, yet three operations are unique to Origraph. We discuss the merits of these novel operations here. **Converting between nodes and edges** is a critical operation when working with multi-typed networks. Consider the case of movies and actors. In the source network, only relationships between movies and actors are captured. If analysts desire to investigate collaboration patterns, they have to convert the movie nodes to edges, capturing collaboration between actors. **Connectivity-based attribute derivation**, in contrast, is critical for analyzing network effects in datasets, i.e., effects that propagate along connections, a question that is fundamental to, e.g., studying social networks [10]. Deriving attributes based on distant connections makes it possible to reveal these network effects at the nodes they influence. For example, we can identify gender bias of movies by deriving attributes from the actors, or, even further removed, create a gender bias score for production companies based on attributes of nodes (actors) that are twice removed. **Connectivity-based filtering**, in turn, is a logical extension of attribute derivation: it allows analysts to leverage network effects at arbitrary distances to simplify the network to suit their analysis needs.

Some of the described operations can be achieved by sequentially executing other operations. For example, **connectivity-based filtering** can be achieved by first **deriving** attributes based on connectivity and then **filtering** based on attributes. Similarly, converting nodes to edges could be achieved by first **projecting edges** and then deleting the intermediate nodes and edges. We have chosen to include these operations nevertheless, because we believe that they are closer to an analyst's mental model and require less indirection.

6 Origraph Design

We implemented the network wrangling operations we identified in Origraph, complementing them with visualizations supporting the operations, showing the state of the network in various views. A key design goal of Origraph is to immediately visualize the effect of an operation on the network model, the underlying topology, and the attributes, so that analysts can easily understand their actions.

The interface contains three main views, shown in Figure 1: a network model view, an attribute view, and a network sample view, in addition to various operation-specific views. The network model view is one of several ways users can modify the state of the network by generating new node and edge classes, and connect existing classes. The attribute view shows a table for each node and edge class, where attributes are columns and rows are instances. Table headers enable users to filter, sort, control instance labeling, and generate new node classes based on attributes. The network sample view displays a sample of the network with a concrete instance of each class in the network model. Views can be flexibly placed and scaled.

Origraph aims to make interactions for operations as close to their visual representation as possible. For example, operations such as **connecting** node classes are supported through direct manipulation of the network model; similarly, operations that utilize attributes are accessible from the attribute headers.

6.1 Network Model View

The network model view serves to represent all classes and their relationships, and to initiate modeling operations. The network model is represented as a node-link diagram, as shown in Figure 8. Node classes are represented as circles and edge classes as lines. Generic classes (before they are assigned to be nodes or edges) are represented as diamonds. We chose these encodings because they are consistent with the mental model most analysts have about nodes and edges. Recall that edges can have arbitrarily many and complex attributes, just as nodes, and that they can exist independent of nodes, which implies that we need to interact with edges in the same ways as with nodes. Our final edge encoding enforces that a segment of the edge is always horizontal and allows a connection to a node on each side (notice that a segment of the cast edge is horizontal in Figure 8). The horizontal segment ensures readable labels (no tilted text) and a simplified way of interacting with edges (no rotation). A downside of these restrictions is that effective layouts can be challenging with off-the-shelf algorithms, which is why we rely mostly on manual layouts. We also experimented with other glyphs for edges that are more flexible in terms of where we can draw connection lines. We, however, found it more important to use the edge metaphor than to enable better automatic layouts.

The number of items in each class is displayed with labels. We also experimented with graphical encodings for the number of items, but found that significantly different class sizes are common, resulting in unbalanced visuals and difficulties in interacting with the classes. Each class is assigned a unique color and a class label. Both are used consistently across all views for elements associated with that class. Although qualitative color scales are limited in the number of distinguishable colors, we observed that most cases do

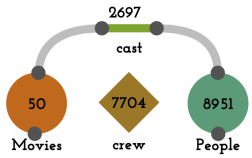

Figure 8: A simple network model for the movies dataset. Node classes (movies, people) are represented as circles, edge classes as lines (cast), and generic classes as diamonds (crew). The cast edge class connects movies and people.

not exceed six or seven classes. For more than eight classes, we use gray as the color and fall back to labels and interactive highlighting.

Nodes and edges have handles that can be used to initiate **connection** and **edge projection** operations by dragging the handle of one class to the other. **Conversion** and **direction changes** are available in place in the network model view.

6.2 Attribute View

The **attribute view** uses a tabular layout with multiple tabs to represent classes, attributes, and items. Each class is displayed in a separate table/tab, with a column for each class attribute and a row for each item. In contrast to other tools [21,33], we chose to show not only the attributes and data types, but also the whole table, because we believe that it is important to be able to quickly assess the full dataset when making wrangling decisions. Our attribute view shows the data in cells as numbers or strings, but we consider visual encodings for future versions [16]. The attribute table can also handle nested data structures, such as arrays and objects, which we encode using curly or square brackets for objects and arrays, respectively, enclosing a number indicating the size of the data structure.

Column headers serve as the interface for initiating operations that are based on attributes, including the modeling operations **promotion** and **faceting**, the item operations **direct** and **connectivity-based** filtering, as well as the attribute operations for **in-class** and **across-class** attribute derivation.

6.3 Network Sample View

The network sample view renders a force-directed node-link diagram containing a sample of the network in its current state. The nodes are colored according to their class; node labels are shown on hover, and are rendered persistently for selected nodes and their neighbors. The attribute view and network sample view are linked through highlighting. Which attribute to use as a label can be changed in the attribute view.

By default, we sample from node and edge classes while ensuring that the sampled items are connected. We achieve this by using the depth-first sampling approach described by Hu and Lau [22], but slightly modify it to balance sampling across classes. Alternatively, users can seed nodes or edges of interest using controls in the network sample view, or add specific items from an attribute table. To see the relationships of a specific node, its neighbors can be added on demand.

Our strategy of working with network samples and selected items is based on the assumption that most networks wrangled with Origraph exceed what can sensibly be drawn using a simple node link layout. Sampling is appropriate when the goal of the analyst is to quickly judge the effect of wrangling operations, whereas the approach of selectively querying and expanding a subgraph allows analysts to address focused network tasks, where the details of specific node and edge neighborhoods and their attributes matter [44].

6.4 Operation Views

In addition to the views that represent the state of the network and the underlying data, Origraph provides several views designed to aid analysts in executing operations.

Connection Support Interface

The **connect** operation matches items in two classes. Origraph supports node-edge or node-node connections. Node-edge connections connect the nodes to only one "side" of an edge and are commonly followed by a similar operation on the other side. For example, when treating movie roles as edges, we connect that edge first to actors and then to movies using two connect operations. Node-node operations implicitly introduce a new edge class.

Connecting two classes can be difficult, especially if the attribute tables are large, and inconsistent column labels are used. Origraph

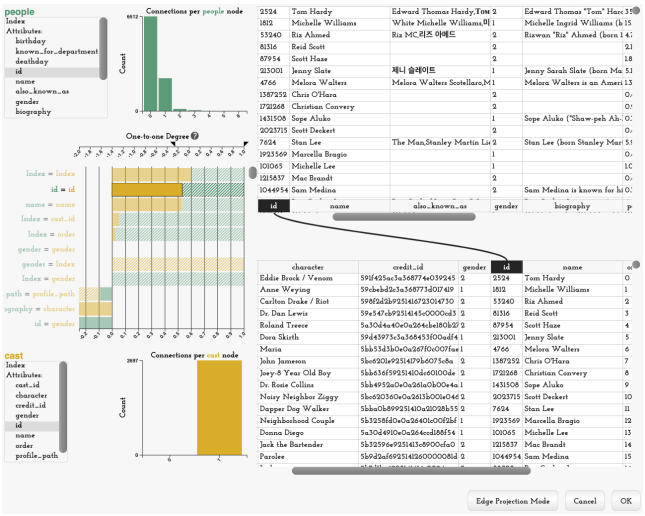

Figure 9: The connection support interface aids in finding the right connections between attributes of two different classes. The raw data of each class is shown on the right. The left side shows a score for different attribute combinations. Here the column named 'id' is selected in both classes (people and cast). The histograms at the top and the bottom show the degree distribution for the selected attribute.

supports this process through a visual interface, shown in Figure 9. The interface calculates all pairwise matches between the source and target classes for all attribute combinations and the index of the items. Based on that information, we calculate a heuristic score for every attribute combination between the items of two classes (*src* and *trg*) as follows:

$$score_{n,m} = \sum_{class \in \{src,trg\}} \frac{\sum_{item \in class} \begin{cases} \frac{1}{deg_{n,m}(item)}, & \text{if } deg_{n,m}(item) > 0 \\ -1, & \text{otherwise} \end{cases}}{|class|}$$

Here, n is the attribute in the *src* class, m in the *trg* class, and $deg_{n,m}(item)$ is the degree of the item when connecting between the classes based on these attributes. The heuristic is designed to give a greater weight to 1-1 matches and slightly discounts matches to nodes with higher degrees. Discounting nodes with higher degrees avoids issues with cases where a small set of categorical labels that are identical between the two classes produces many matches. The heuristic penalizes missing connections and normalizes the final scores to the class size. The heuristic results in a score of 2 if each item in the source class matches exactly one item in the target class and there are no mismatches, and -2 if no matches are found.

The contribution of each class to this heuristic is shown in the interface as a stacked bar chart. Solid bars encode the final score, and shaded bars encode one-sided scores where the overall score is smaller than the score of one class. Additionally, histograms show the degree distribution of the items. Figure 9 shows connections for people to a list of cast members. The people class contains many individuals who are crew but not cast members, but each cast member has a match in the people table. The people degree distribution shows that most people acted in no movies (the crew), followed by many in a single movie, and much fewer in multiple movies. The heuristic score shows that the highest score matches the indices of the two classes, which is not correct. The score between the two id attributes is slightly lower. The relationship between these attributes is also shown as a link between the tables.

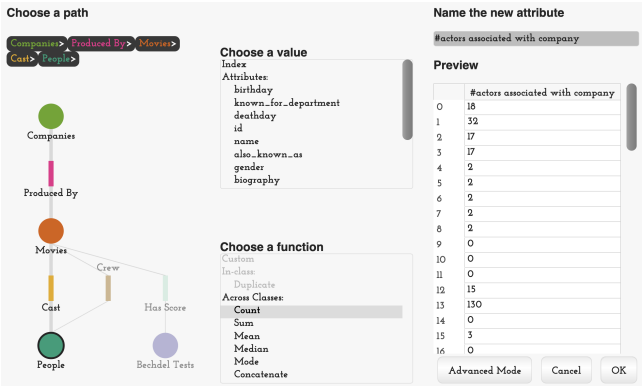

Figure 10: Path selection (left) and apply function (right) interface showing a **derive connected attribute** operation counting the number of actors associated with a production company. A preview of the results is shown on the right.

Path-Based Operations

Several operations, including **edge projection**, **deriving attributes based on connectivity**, and **connectivity-based filtering**, require analysts to specify conditional paths in a network. For example, to connect an actor with a direct edge to a production company, as illustrated in Figure 5, analysts have to specify the path from actor to role to movie to production company.

To simplify selecting these paths, Origraph includes an interface that displays all paths across node and edge classes starting from a selected node, as shown in Figure 10. Analysts can select specific paths (including loops), which are shown in a breadcrumb interface at the top. These paths are then used as input to the aforementioned operations.

Specifying Functions

The two operations concerned with attribute derivation (**in-class** and **connectivity-based** attribute derivation); and the filter operations (**direct** and **connectivity-based** filtering) are based on evaluating functions over one or multiple values. Here, the tension is most acute between supporting operations that do not strictly require programming, yet could benefit significantly from its expressiveness. To address this balance, each of these operations has two modes. One is a standard, non-programming mode that provides sets of common functions such as count, sum, or median for attribute derivation; or simple less-than, equality, or greater-than comparisons for filtering (see Figure 10).

An advanced mode drops analysts into a pre-populated code template that implements the standard function they had previously chosen. They can then adapt this function as needed. For example, Figure 11 shows the computation of a gender bias ratio, which was minimally adapted from auto-generated code for computing the median.

7 Implementation

Origraph is a client-only web application, published under a permissive open-source license. The source code is available at `https://github.com/origraph/origraph.github.io`.

Origraph is designed to help analysts create a set of rules for reshaping a graph in a lightweight, flexible interface. In its current form, it scales to medium-sized networks (up to tens of thousands of nodes) that can fit in memory. However, Origraph's interface is built on top of an independent graph processing library `origraph.js`, available at `https://github.com/origraph/origraph.js`. Ultimately, our goal is to support interactive operations with an in-memory sample of a large graph, and then export a script capable of applying users' rules to much larger datasets, similar to the strategy pioneered by existing tabular data wrangling tools [26, 61, 64].

The underlying library interacts with raw data using two different layers of abstraction: a **table network** layer, underneath an **interpreted network model** layer. Similar to the schema of a relational database, the table network is a lazily evaluated specification of raw and derived tables, and any connections between them. The interpreted network layer maps tables in the table network to classes in the network model: both node and edge classes must map to a specific target table. This way, edges in the graph are always guaranteed to support features, such as edge attributes, that existing network modeling tools [21, 33] cannot represent. Furthermore, edge classes reference up to two paths through the table network, from the target edge table to its connected node tables. Distinct paths between tables imply that edge classes can be fully or partially disconnected, and these states are reflected in Origraph's interface. Keeping these two layers separate allows for free-form reinterpretation of the underlying tables as node or edge classes.

8 Input and Output

As for any data wrangling tool, input and output are important considerations. Our design goal for input is to ingest data formats from "in-the-wild" data sources and to avoid the need for pre-processing. To this end, Origraph supports tabular data that can contain explicit node and link lists; alternatively links can be inferred based on attributes.

Another input data type is hierarchical, specifically JSON, which is a common format for API responses from numerous online services. The hierarchy commonly contains multiple levels of items that can be individually represented as nodes and edges. A movie object, for example, can contain an array of all cast members, themselves represented as complex objects. Origraph implements special **unroll** and **expand** operations to convert these nested structures into separate classes.

Origraph currently exports d3.js-style JSON, zipped CSV files, and GEXF for analyzing the wrangled graphs in off-the-shelf graph visualization tools such as Gephi or Cytoscape. The prototype comes with multiple datasets that can be loaded in raw or pre-assembled format.

9 Use Cases

Here we demonstrate how Origraph can be used to wrangle two network datasets. The data for both cases was retrieved from several different APIs, and no data wrangling was performed outside of Origraph. The scripts used to access the API endpoints and the resulting data can be found at `https://github.com/origraph/data`; each dataset is also provided as a sample in Origraph. Figures showing the full state of the interface at each step (M1–M18; Y1–Y15) are available in the supplemental material.

9.1 Gender Bias in Movies

The network used in this example is a network of movies, actors, crew members, and production companies. The data was retrieved from "The Movie Database" [11], a community-built movie and TV database. The analysis question we want to investigate concerns issues related to gender bias in the movie industry.

We select the 50 most popular movies (according to TMDB's internal ranking of popularity), retrieve data related to these movies, and store the data in three JSON files. The movies file contains information on the 50 most popular movies, including attributes such as movie id, title and year of release, budget, genre, spoken languages, popularity, run time, revenue, and nested objects describing the production companies involved. The people file contains information on all people in these movies and contains a unique id for each person in addition to attributes such as gender, popularity, place of birth, and day of death. The credits file contains two nested objects, one

for cast and one for crew. These objects contain attributes such as roles, departments, and the ids for movies and people. In total, the datasets contains 2689 cast items, 7704 crew items, 116 production companies, 181 relationships between production companies and movies, and 8951 people.

We also retrieved a dataset containing Bechdel ratings for 7871 movies [59], including the 50 that we retrieved from TMDB. The Bechdel test assigns a rating from 0-3 to each movie based on three criteria: (1) the movie must have at least two women in it, who (2) talk to each other, about (3) something besides a man. A movie that fails all three criteria is assigned a score of 0; a movie that passes the test is assigned a score of 3. Supplementing the TMDB data with the Bechdel rating allows us to answer some interesting gender-related questions about the movies.

We start the process of modeling a network by importing the raw data described above. Our first objective is to combine the Bechdel and TMDB data about movies. To do this, we employ **convert**, **derive attribute**, **connect**, and **derive connected attribute** operations to arrive at a combined movies class with attributes from both sources (see Figures M1–M6). We then unroll nested cast and crew objects as distinct classes (Figure M7), **convert** people to nodes, cast and crew to edges (Figure M8), and then **connect** both cast and crew to people and movies (Figure M9). At this stage, we have modeled a basic network; movies and people are connected through cast and crew edges, and each movie has an associated Bechdel score. This dataset might suffice to answer basic questions concerning which people participated in movies with high or low Bechdel scores; however, an analyst might also be interested in gender bias at the level of production companies.

To treat production companies as distinct entities, we scroll through the movie attributes to "production_companies". The values in this column are arrays of objects, so we unroll them into their own class (Figure M10). The unroll operation also connects the new items to the movie nodes they originated from. We rename the newly created class "produced by" to better reflect the meaning of this class in our model.

We leverage the **promote** operation on the company "name" attribute to create a new class with unique company names. We then rename the newly created class "Companies" (Figures M11–M12). However, companies are now connected to movies only indirectly via "produced by" nodes. Semantically, it would be more meaningful to connect companies directly to movies via an edge, which we can achieve with the **convert** operation, applied to the "produced by" node class (Figure M13). Since Origraph supports rich edge attributes, these edges preserve all the attributes of the original data.

With this iteration of the network model built, we are ready to reshape the network to support analysis questions regarding gender bias for each movie. To do this, we compute a new attribute in the movie table. Similar to the earlier task of copying Bechdel scores, this operation requires information from connected classes, so we use the **connectivity-based attribute derivation** operation. In order to calculate the gender bias for actors in a movie, we must access the gender of all the people who are connected to the movie via a cast edge. In this example, we are interested in the ratio of men to women, so we select gender from the list of attributes, choose the mean computation that approximates what we want to compute (Figure M14), and then select the advanced mode, which allows us to modify the mean code to compute the ratio of men to women (Figures 11 and M15). A preview table on the right shows sample values to ensure we are computing the attribute correctly.

Once the gender bias has been computed, we sort on this attribute in the table, which reveals that of the movies in this dataset, "Ocean's Eight" is the one with the highest ratio of women to men (Figure M16). Interestingly, the movie with the highest gender bias, "BlacKKKlansman", passes the Bechdel test, with a score of three, which suggests that one metric (or both) possibly oversimplifies the

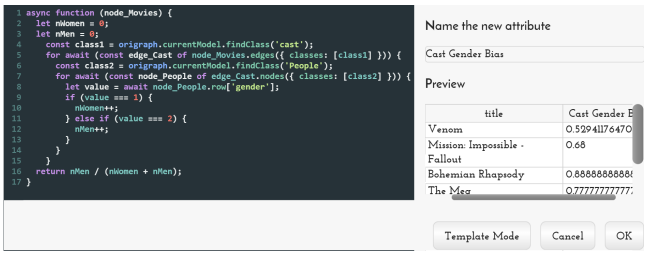

Figure 11: Interface for deriving a custom attribute using code. In this case, the user adapts an auto-generated template for computing the mean value, changing eight lines of code: the counter initialization lines (2,3); the *if* conditionals and increments (9-13); and the return statement (16).

concept of gender bias. To explore the relationship between each metric, we can export the movies class as a CSV file, and visualize Cast Gender Bias, Crew Gender Bias, and Bechdel scores in a scatterplot matrix (Figure M17).

A follow-up question is to find out which actors tend to be cast in movies with a lower gender bias. We can explore the local connections around movies of interest using Origraph's sample interface (Figure 1), or export finished network dataset to a more dedicated network visualization tool such as Gephi (Figure M18).

9.2 Money and Political Support for the War in Yemen

Our second use case focuses on a network of current US senators, voting behavior on the recent bill regarding US support of the Yemen war, donors of these senators, and their social media statements related to this issue. This topic has received considerable attention in the media [15, 34] and serves as an interesting use case to demonstrate the ability of Origraph to connect data from disparate sources to tell a compelling story. The bill in question (session 2, roll call 250) was to determine whether the US should remove military support from the war in Yemen. The final vote count was in support of removing support. Yet it is still interesting to model a network that can address questions related to the votes of senators and their connections to donors. One such question is whether senators who voted against the Yemen bill were financially supported by a specific subset of donors with an interest in the conflict. A related question is whether senators were more or less vocal in their support or opposition, which we can estimate based on press releases and tweets.

We obtained the data from the ProPublica Congress API [52] and through the Twitter API [62]. Again, we did not perform any preprocessing on the datasets retrieved from these APIs. The raw data files imported in this example include: information for all current senators including gender and political party, the way each senator voted for the bill on Yemen, all press releases made by members of the senate in relation to the Yemen bill, all FEC reports about donations made in 2018 that either support or oppose a member of the senate, and all tweets made by senators from Nov. 28 to Dec. 4, 2018.

Once the raw files have been loaded (Supplementary Figure Y1), we **convert** senators, votes, campaign contributions, press releases, and tweets to nodes (Figure Y2). We **connect** senators to their twitter accounts by expanding nested information (Figures Y3–Y5). Because we ultimately care about connecting senators to their tweets, we can abstract away the twitter account class by **converting** it to an edge. We now have a model of senator nodes, connected directly to their tweets (Figure Y6). The next step involves **connecting** senators to their votes, their press releases, and any donations made toward or against their campaign. This step can be done directly with the connect interface, as press releases, votes, and donations all have an attribute that directly references the senator ids (Figure Y7). We are particularly interested in distinguishing

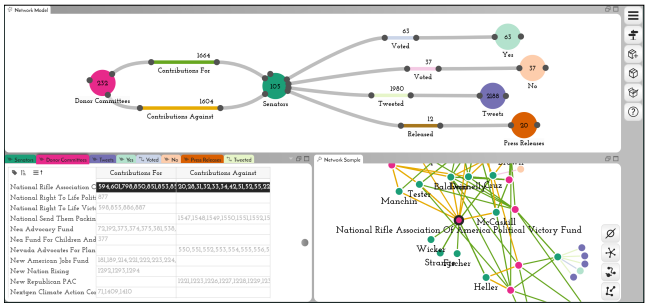

Figure 12: A political donor, vote, tweet, and press release network. Senators are connected to committees that donated to either support or oppose them. They also have edges connecting them to how they voted on the Yemen bill, their tweets, and press releases.

between senators who voted for or against the bill, so we **facet** the votes node into separate "Yes" and "No" classes before connecting them (Figure Y8).

The last step in modeling this network involves extracting the individual donation committees. Since several donations are made by the same committee, we first want to extract all unique committees from the donations class. We leverage the **promote** operation on the "committee name" attribute, which generates a new "Donor Committees" class, with edges connecting each committee to their donation in the "Donations" class (Figure Y9).

Because we are interested in which committees support or oppose certain senators, we **convert** the donations node into an edge (Figure Y10), and then **facet** the edge into contributions that support and those that oppose a senator (Figure Y11).

With this network modeled, as shown in Figure 12, we can turn to the initial questions regarding relationships between donors and vote outcomes on the Yemen bill. Our current network connects donor committees to Yes and No votes through specific senators. An analyst may be less interested in specific senators than relationships between funding sources and votes; consequently, the intermediate senators are a hindrance in this state of the network. To remedy this, we can **project new edges** that connect donor committees directly to vote classes (Figure Y12), resulting in a new edge class that is a reasonable proxy for donor interest in the conflict. Exporting the new projected edges, with donor and vote nodes, into Gephi allows us to explore the donor interest network (Figure Y13). We discover that, even though many republicans voted Yes on this bill, the NRA supported only Republican senators who voted No, and opposed senators who voted Yes. Although this pattern in connectivity does not directly prove NRA support for the Yemen conflict, it could suggest a point that an analyst or journalist might want to investigate further.

Having incorporated Twitter and press release data, we may also be interested in whether specific senators were vocal about the conflict. We can derive attributes on senators to ascertain whether their tweets contain relevant "Khashoggi," "Saudi," or "Yemen" strings, as well as whether or not a senator issued a press release about their vote (Figure Y14). These new attributes can be interpreted as set relationships—yet another data model—that might be better explored in a tool such as UpSet [30]. Exporting the derived data to UpSet reveals that, with the exception of only three senators, senators who voted No were relatively silent about their votes (Figure Y15).

10 Discussion

Data abstractions often change in response to evolving exploration and analysis tasks. The shape of the data is a proxy for an analyst's real-world problem, and must be continuously assessed and refined [14]. We argue that transforming a dataset into the form best suited to answer analysis questions is essential, yet the tools currently available for wrangling multivariate network datasets are not powerful enough. The alternative of using scripting or database query languages is time consuming, requires skill sets many analysts do not have, and does not allow for rapid iterative exploration of the changes made.

To address this problem, we introduce novel operations and visual analysis interfaces to support sophisticated interactive network wrangling. By giving users the freedom to explore alternative data abstractions, tools like Origraph have the potential to become powerful thinking tools that could begin to address the open challenge of how to support visualization practitioners in exploring more diverse data abstractions [7, 40]. Although the design space of data wrangling tools is still in its infancy, it is worth considering its breadth: identifying data wrangling as a distinct category from data analysis [25] may need to be extended further to distinguish among data analysis, classical "wrangling", and abstraction transformation. The standard notion of data wrangling [25] is often viewed through a lens of data integration, diagnosing, and cleaning data problems. In contrast, abstraction transformation is motivated by users' mental models and hence requires a distinct set of methods that support different operations. Nevertheless, each category benefits from tight integration with the others. For example, in Origraph, we integrate data analysis through visualization with wrangling, which is also imperative for checking whether wrangling operations have their intended effects.

10.1 Comparing Origraph to a Computational Workflow

All wrangling operations that can be executed with Origraph can also be implemented using different programming languages. However, using a programming language has multiple disadvantages: first, the skill level required to implement certain operations limits the number of individuals who can execute them significantly, making it prohibitive for data-literate non-programmers (e.g., individuals who use Excel, Tableau, or Statistics software such as SPSS) to do network wrangling.

All modeling operations in Origraph can be executed without programming, which is noteworthy because operations that change the database schema are relatively complicated in scripting languages such as SQL. Only non-standard attribute operations require programming in Origraph: the "apply" functions following the split-apply-combine paradigm and advanced conditionals for filtering items. Origraph provides multiple defaults for these functions, such as sums, means, and concatenation. However, the design space of useful functions is considerable; hence Origraph allows analysts to write custom expressions. We argue, however, that expressions operating on lists of attributes are much easier to write compared to, for example, operations that change the network model. Nevertheless, we plan on simplifying our scripting interface, for example, by making it more like Excel, with basic formulas, easy references to data sources, and accessible documentation.

10.2 Comparison to Other Tools

Origraph shares parts of the vision and functionality of Orion [21] and Ploceus [33]. Origraph, however, is unique and novel with regard to three aspects: (1) it supports important operations that are missing in Orion and Ploceus, (2) it treats edges as first-class objects that can be associated with complex attributes, and (3) it contains sophisticated, algorithm-supported visual interfaces that make executing complex operations easier. Orion and Ploceus also do not support nested data structures (lists and hierarchies) as attributes of the tables.

With regards to operations, Origraph supports all operations listed in Section 5. Ploceus and Orion do not support **converting between nodes and edges**, **creating supernodes**, **connectivity-based filtering**, **changing edge direction**, and **connectivity-based attribute derivation**. Although Origraph and Ploceus sup-

port **projecting edges**, they do so only for immediate neighbors, whereas Origraph can project edges based on arbitrary complex paths. Notably, several of the operations that are unique to Origraph are about leveraging network structures, e.g., aggregating attributes at a certain distance from source nodes. Supplementary Table 1 contains a comparison of these operations. Finally, in contrast to Orion and Ploecus, the code of Origraph is open source, and the tool is available for anyone to upload datasets.

10.3 Evaluation

Criteria for evaluating data wrangling systems are not well established. Similar to visualization authoring toolkits, the primary goal of a data wrangling system is to *create*, rather than *analyze*—consequently, a wide variety of concerns merit evaluation, such as expressiveness, creativity support, flexibility, guidance, efficiency, usability, learnability, and integration [53]. In this work, we have focused primarily on demonstrating expressiveness through use cases, and to some extent, the system's design demonstrates creativity support, flexibility, and integration. We consciously chose not to run a quantitative study comparing Origraph with other systems or methods. A comparison to existing systems, such as Ploceus and Orion, is not possible, since these tools are not available to the public. A comparison to a programming-based approach hinges on the scripting skills of the subjects; yet our target users are those who have no or minimal programming skills. Future work is needed to evaluate concerns that we have not considered in depth, such as usability. We also believe that a discussion around how to best evaluate data wrangling systems is needed.

10.4 Limitations

Visualization. The current focus of Origraph is on making sophisticated data wrangling operations as easy as possible to execute. Origraph supports analysis through visualizations, but visualizations of the network and the attributes can be improved in many ways. We are planning to integrate Origraph with a general purpose multivariate network visualization system that is based on existing tools developed at our lab [27, 44, 47].

Focus on Abstraction Design. Origraph targets the design of data abstractions, rather than the discovery, capture, curation, or creation of data [39]. For example, the support for cleanup and dealing with missing data is limited. We believe that such operations are better addressed in separate tools before importing data into Origraph.

Scalability. Although Origraph scales to thousands of nodes and edges, it does not scale to arbitrarily large networks. We plan on improving scalability using an approach common to visual data wrangling tools: loading a sampled dataset. Operations can then applied and refined on the sample using visual inspection. After the transformation is completed, a script is generated that can be used to process a larger dataset offline, potentially leveraging cloud infrastructure.

Connectivity Heuristic. In our experiments with various datasets, we found that our heuristic always gives high scores to the right combination of attributes, but also observed that index-to-index scores are commonly highly ranked. If, for example, two datasets have the same number of rows but have nothing else in common, our heuristic would give a perfect score to the index-to-index connection. Since some datasets rely on the index of items to establish connections between datasets, we cannot generally exclude it from our computation. However, we are considering treating connectivity combinations involving an index separately from attribute to attribute connections. We observed that our heuristic can be slow with large classes. It has a runtime complexity of $\mathcal{O}(n*m*i*j)$ where n and m are the number of unique items in the classes (which are frequently very large), and i and j are the number of attributes plus the index in the classes (which are typically small). Sampling could reduce the runtime complexity to $\mathcal{O}(k*m*i*j)$, where k is a user-chosen value that is much smaller than n and should result in a significant speed-up with a limited loss of accuracy.

11 Conclusions and Future Work

By implementing the set of operations that we have identified, Origraph is a first steps toward allowing data-literate non-programmers to wrangle networks to answer important questions in their area of inquiry and to produce visualizations they can use to communicate their findings. Origraph is designed to ingest raw data, e.g., in the form of JSON retrieved from an API, and then enable analysts to wrangle the data into a network that corresponds to their mental model. Build-in visualization capabilities enable analysts to quickly iterate between analysis and modeling. Once a network is in the desired form, it can be either investigated within Origraph or exported for analysis in more sophisticated multivariate network visualization tools. Through two use cases, we have demonstrated Origraph's expressive potential.

We hope to explore and augment Origraph through ongoing deployment, testing, and redesign. These efforts will include more exhaustive evaluations with users testing and improving the tool's usability; exploring the usefulness of each operation and what operations may be useful beyond the ones that we have identified; and identifying gaps in its expressiveness. Additionally, we plan to integrate more representative sampling and improve scalability; to give users access to more sophisticated algorithms, such as seriated instance ordering, or clustering-based group assignments; to add provenance features to make it possible to edit, audit, share, document, and replicate users' transformation processes; and to integrate more closely with data cleaning and multivariate graph visualization to more fully support end-to-end analysis workflows.

12 Acknowledgments

This work was funded by the National Science Foundation (OAC 1835904, IIS 1350896, IIS 1751238) and by the Utah Genome Project.

References

[1] D. Auber, D. Archambault, R. Bourqui, M. Delest, J. Dubois, A. Lambert, P. Mary, M. Mathiaut, G. Mélançon, B. Pinaud, B. Renoust, and J. Vallet. TULIP 5. In R. Alhajj and J. Rokne, eds., *Encyclopedia of Social Network Analysis and Mining*, pp. 1–28. Springer new-York, Aug. 2017. doi: 10.1007/978-1-4614-7163-9_315-1

[2] B. Bach, A. Spritzer, E. Lutton, and J.-D. Fekete. Interactive Random Graph Generation with Evolutionary Algorithms. In *Proceedings of the 20th International Conference on Graph Drawing*, GD'12, pp. 541–552. Springer-Verlag, Berlin, Heidelberg, 2013. doi: 10.1007/978-3-642-36763-2_48

[3] C. A. Bail. Combining natural language processing and network analysis to examine how advocacy organizations stimulate conversation on social media. *Proceedings of the National Academy of Sciences*, 113(42):11823–11828, Oct. 2016. doi: 10.1073/pnas.1607151113

[4] C. A. Bail. Cultural carrying capacity: Organ donation advocacy, discursive framing, and social media engagement. *Social Science & Medicine*, 165:280–288, Sept. 2016. doi: 10.1016/j.socscimed.2016.01.049

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

[6] M. Behrisch, D. Streeb, F. Stoffel, D. Seebacher, B. Matejek, S. H. Weber, S. Mittelstaedt, H. Pfister, and D. Keim. Commercial Visual Analytics Systems-Advances in the Big Data Analytics Field. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–24, July 2018. doi: 10.1109/TVCG.2018.2859973

[7] 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

[8] A. Bigelow and M. Monroe. Jacob's Ladder: The User Implications of Leveraging Graph Pivots. *IEEE Transactions on Visualization and Computer Graphics (PacificVis 2019), to appear.*, p. Forthcoming, June 2019.

[9] M. Bilgic, L. Licamele, L. Getoor, and B. Shneiderman. D-Dupe: An Interactive Tool for Entity Resolution in Social Networks. In *2006 IEEE Symposium On Visual Analytics Science And Technology*, pp. 43–50, Oct. 2006. doi: 10.1109/VAST.2006.261429

[10] N. A. Christakis and J. H. Fowler. The Spread of Obesity in a Large Social Network over 32 Years. *New England Journal of Medicine*, 357(4):370–379, July 2007. doi: 10.1056/NEJMsa066082

[11] T. M. D. Community. The Movie Database (TMDb). https://www.themoviedb.org/documentation/api, 2018.

[12] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull. Graphviz: Open Source Graph Drawing Tools. In *Graph Drawing*, Lecture Notes in Computer Science, pp. 483–484. Springer, Berlin, Heidelberg, Sept. 2001. doi: 10.1007/3-540-45848-4_57

[13] N. Elmqvist, T.-N. Do, H. Goodell, N. Henry, and J. Fekete. ZAME: Interactive Large-Scale Graph Visualization. In *Visualization Symposium, 2008. PacificVIS '08. IEEE Pacific*, pp. 215–222, 2008. doi: 10.1109/PACIFICVIS.2008.4475479

[14] D. Fisher and M. Meyer. *Making Data Visual*. O'Reilly Media, Jan. 2018.

[15] B. Freeman. Meet the Senators Who Took Saudi Money — The American Conservative. *The American Conservative*, Dec. 2018.

[16] K. Furmanova, S. Gratzl, H. Stitz, T. Zichner, M. Jaresova, A. Lex, and M. Streit. Taggle: Combining Overview and Details in Tabular Data Visualizations. *arXiv:1712.05944 [cs]*, Dec. 2017.

[17] 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.

[18] T. J. Hagey, J. C. Uyeda, K. E. Crandell, J. A. Cheney, K. Autumn, and L. J. Harmon. Tempo and mode of performance evolution across multiple independent origins of adhesive toe pads in lizards. *Evolution*, 71(10):2344–2358, Oct. 2017. doi: 10.1111/evo.13318

[19] Z. Hammoud and F. Kramer. Mully: An R Package to Create, Modify and Visualize Multilayered Graphs. *Genes*, 9(11), Oct. 2018. doi: 10.3390/genes9110519

[20] J. Heer and A. Perer. Orion: A System for Modeling, Transformation and Visualization of Multidimensional Heterogeneous Networks. In *IEEE Visual Analytics Science \& Technology (VAST)*, p. 10, 2011.

[21] 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

[22] P. Hu and W. C. Lau. A Survey and Taxonomy of Graph Sampling. *arXiv:1308.5865 [cs, math, stat]*, Aug. 2013.

[23] D. Huynh and S. Mazzocchi. Google Refine, 2011.

[24] N. Inc. Neo4j, 2010.

[25] S. Kandel, J. Heer, C. Plaisant, J. Kennedy, F. van Ham, N. H. Riche, C. Weaver, B. Lee, D. Brodbeck, and P. Buono. Research directions in data wrangling: Visualizations and transformations for usable and credible data. *Information Visualization*, 10(4):271–288, Oct. 2011. doi: 10.1177/1473871611415994

[26] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, p. 3363, 2011. doi: 10.1145/1978942.1979444

[27] E. Kerzner, A. Lex, C. Sigulinsky, T. Urness, B. Jones, R. Marc, and M. Meyer. Graffinity: Visualizing Connectivity in Large Graphs. *Comput. Graph. Forum*, 36(3):251–260, June 2017. doi: 10.1111/cgf.13184

[28] J. S. Lauritzen, C. L. Sigulinsky, J. R. Anderson, M. Kalloniatis, N. T. Nelson, D. P. Emrich, C. Rapp, N. McCarthy, E. Kerzner, M. Meyer, B. W. Jones, and R. E. Marc. Rod-cone crossover connectome of mammalian bipolar cells. *Journal of Comparative Neurology*, 527(1):87–116, Jan. 2019. doi: 10.1002/cne.24084

[29] B. Lee, C. Plaisant, C. S. Parr, J.-D. Fekete, and N. Henry. Task Taxonomy for Graph Visualization. In *Proceedings of the AVI Workshop on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization (BELIV '06)*, pp. 1–5, 2006. doi: 10.1145/1168149.1168168

[30] A. Lex, N. Gehlenborg, H. Strobelt, R. Vuillemot, and H. Pfister. UpSet: Visualization of Intersecting Sets. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1983–1992, Dec. 2014. doi: 10.1109/TVCG.2014.2346248

[31] A. Lex, C. Partl, D. Kalkofen, M. Streit, S. Gratzl, A. M. Wassermann, D. Schmalstieg, and H. Pfister. Entourage: Visualizing relationships between biological pathways using contextual subsets. *IEEE transactions on visualization and computer graphics*, 19(12):2536–2545, Dec. 2013. doi: 10.1109/TVCG.2013.154

[32] Z. Liu, S. B. Navathe, and J. T. Stasko. Network-based visual analysis of tabular data. In *2011 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 41–50, Oct. 2011. doi: 10.1109/VAST.2011.6102440

[33] Z. Liu, S. B. Navathe, and J. T. Stasko. Ploceus: Modeling, visualizing, and analyzing tabular data as networks. *Information Visualization*, 13(1):59–89, Jan. 2014. doi: 10.1177/1473871613488591

[34] C. Maza. Republican senators who tried to kill Yemen war resolution were paid by Saudi lobbyists. *Newsweek*, Nov. 2018.

[35] F. McGee, M. Ghoniem, G. Melançon, B. Otjacques, and B. Pinaud. The State of the Art in Multilayer Network Visualization. *Computer Graphics Forum*, 0(0), 2019. doi: 10.1111/cgf.13610

[36] M. Meyer, B. Wong, M. Styczynski, T. Munzner, and H. Pfister. Pathline: A Tool For Comparative Functional Genomics. *Computer Graphics Forum*, 29(3):1043–1052, June 2010. doi: 10.1111/j.1467-8659.2009.01710.x

[37] Miriah Meyer, Alexander Lex, Jeff Baumes, Curtis Lisle, Luke Harmon, Bryan Jones, Christopher Bail, and Shaunna Morrison. Collaborative Research: Framework: Software: HDR: Reproducible Visual Analysis of Multivariate Networks with MultiNet. 2018.

[38] S. M. Morrison, C. Liu, A. Eleish, A. Prabhu, C. Li, J. Ralph, R. T. Downs, J. J. Golden, P. Fox, D. R. Hummer, M. B. Meyer, and R. M. Hazen. Network analysis of mineralogical systems. *American Mineralogist*, 102(8):1588–1596, 2017. doi: 10.2138/am-2017-6104CCBYNCND

[39] M. Muller, I. Lange, D. Wang, D. Piorkowski, J. Tsay, Q. V. Liao, C. Dugan, and T. Erickson. How Data Science Workers Work with Data: Discovery, Capture, Curation, Design, Creation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pp. 126:1–126:15. ACM, 2019. doi: 10.1145/3290605.3300356

[40] T. Munzner. A Nested Model for Visualization Design and Validation. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):921–928, Nov. 2009. doi: 10.1109/TVCG.2009.111

[41] 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, 2009 Nov-Dec. doi: 10.1109/TVCG.2009.116

[42] C. Nobre, N. Gehlenborg, H. Coon, and A. Lex. Lineage: Visualizing Multivariate Clinical Data in Genealogy Graphs. *IEEE Transactions on Visualization and Computer Graphics*, 25(3):1543–1558, 2019. doi: 10.1109/TVCG.2018.2811488

[43] C. Nobre, M. Meyer, M. Streit, and A. Lex. The State of the Art in Visualizing Multivariate Networks. *Computer Graphics Forum (EuroVis 2019, to appear)*, 2019.

[44] C. Nobre, M. Streit, and A. Lex. Juniper: A Tree+Table Approach to Multivariate Graph Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):544–554, Jan. 2019. doi: 10.1109/TVCG.2018.2865149

[45] Ontotext. GraphDB, 2000.

[46] OrientDB. OrientDB, 2010.

[47] C. Partl, S. Gratzl, M. Streit, A. M. Wassermann, H. Pfister, D. Schmalstieg, and A. Lex. Pathfinder: Visual Analysis of Paths in Graphs. *Computer Graphics Forum*, 35(3):71–80, June 2016. doi: 10.1111/cgf.12883

[48] C. Partl, A. Lex, M. Streit, D. Kalkofen, K. Kashofer, and D. Schmalstieg. enRoute: Dynamic path extraction from biological pathway maps for in-depth experimental data analysis. In *2012 IEEE Symposium on*

Biological Data Visualization (BioVis), pp. 107–114, Oct. 2012. doi: 10.1109/BioVis.2012.6378600

[49] C. Partl, A. Lex, M. Streit, D. Kalkofen, K. Kashofer, and D. Schmalstieg. enRoute: Dynamic path extraction from biological pathway maps for exploring heterogeneous experimental datasets. *BMC Bioinformatics*, 14(19):S3, Nov. 2013. doi: 10.1186/1471-2105-14-S19-S3

[50] A. J. Pretorius, H. C. Purchase, and J. T. Stasko. Tasks for Multivariate Network Analysis. In A. Kerren, H. C. Purchase, and M. O. Ward, eds., *Multivariate Network Visualization*, number 8380 in Lecture Notes in Computer Science, pp. 77–95. Springer International Publishing, 2014.

[51] A. J. Pretorius and J. J. van Wijk. Visual Inspection of Multivariate Graphs. *Computer Graphics Forum (EuroVis '08)*, 27(3):967–974, 2008. doi: 10.1111/j.1467-8659.2008.01231.x

[52] ProPublica. The ProPublica Congress API. https://www.propublica.org/datastore/api/propublica-congress-api, 2018.

[53] D. Ren, B. Lee, M. Brehmer, and N. H. Riche. Reflecting on the Evaluation of Visualization Authoring Systems : Position Paper. In *2018 IEEE Evaluation and Beyond - Methodological Approaches for Visualization (BELIV)*, pp. 86–92, Oct. 2018. doi: 10.1109/BELIV.2018.8634297

[54] G. G. Robertson, M. P. Czerwinski, and J. E. Churchill. Visualization of Mappings Between Schemas. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '05, pp. 431–439. ACM, New York, NY, USA, 2005. doi: 10.1145/1054972.1055032

[55] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks. *Genome Research*, 13(11):2498–2504, Jan. 2003. doi: 10.1101/gr.1239303

[56] 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 *Proceedings of the Fourth International Conference on Communities and Technologies*, pp. 255–264. ACM, June 2009. doi: 10.1145/1556460.1556497

[57] 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

[58] C. Systems. Centrifuge, 2018.

[59] B. Test. The Bechdel Test Movie List. https://bechdeltest.com/, 2018.

[60] TouchGraph. Graph Visualization and Social Network Analysis Software — Navigator - TouchGraph.com. http://www.touchgraph.com/navigator, 2018.

[61] Trifacta. Trifacta Wrangler, 2012.

[62] Twitter, Inc. Twitter Developer Platform. https://developer.twitter.com, 2018.

[63] F. van Ham, H.-J. Schulz, and J. M. Dimicco. Honeycomb: Visual Analysis of Large Scale Social Networks. In T. Gross, J. Gulliksen, P. Kotzé, L. Oestreicher, P. Palanque, R. O. Prates, and M. Winckler, eds., *Human-Computer Interaction – INTERACT 2009*, Lecture Notes in Computer Science, pp. 429–442. Springer Berlin Heidelberg, 2009.

[64] R. Verborgh and M. D. Wilde. *Using OpenRefine*. Packt Publishing Ltd, Sept. 2013.

[65] M. Wattenberg. Visual Exploration of Multivariate Graphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pp. 811–819. ACM, New York, NY, USA, 2006. doi: 10.1145/1124772.1124891

[66] C. Weaver. Multidimensional data dissection using attribute relationship graphs. In *2010 IEEE Symposium on Visual Analytics Science and Technology*, pp. 75–82. IEEE, Salt Lake City, UT, USA, Oct. 2010. doi: 10.1109/VAST.2010.5652520

[67] H. Wickham. The Split-Apply-Combine Strategy for Data Analysis. *Journal of Statistical Software*, 40(1):1–29, Apr. 2011. doi: 10.18637/jss.v040.i01