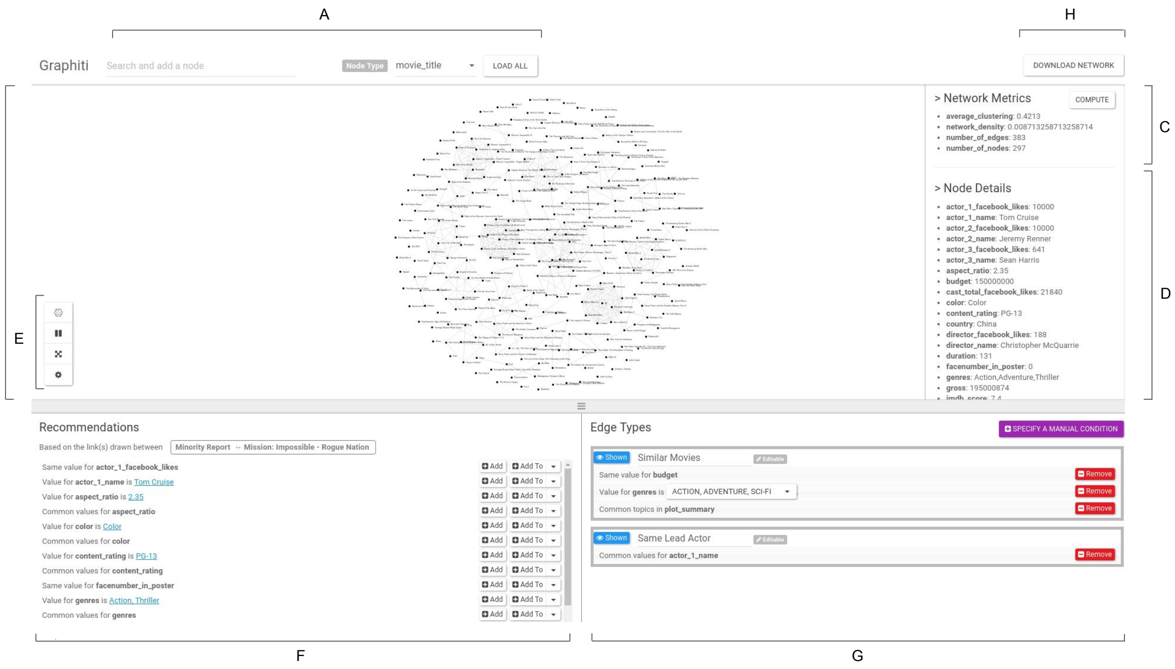
Graphiti: Interactive Specification of Attribute-based Edges for Network Modeling and Visualization

Arjun Srinivasan, Hyunwoo Park, Alex Endert, and Rahul C. Basole



> Network Metrics

- **average_clustering**: 0.4213
- **network_density**: 0.008713258713258714
- **number_of_edges**: 383
- **number_of_nodes**: 297

> Node Details

- **actor_1_facebook_likes**: 10000
- **actor_1_name**: Tom Cruise
- **actor_2_facebook_likes**: 10000
- **actor_2_name**: Jeremy Renner
- **actor_3_facebook_likes**: 641
- **actor_3_name**: Sean Harris
- **aspect_ratio**: 2.35
- **budget**: 150000000
- **cast_total_facebook_likes**: 21840
- **color**: Color
- **content_rating**: PG-13
- **country**: China
- **director_facebook_likes**: 188
- **director_name**: Christopher McQuarrie
- **duration**: 131
- **facenumber_in_poster**: 0
- **genres**: Action,Adventure,Thriller
- **gross**: 195000874
- **imdb_score**: 7.4

Recommendations

Based on the link(s) drawn between Minority Report ↔ Mission: Impossible - Rogue Nation

Same value for **actor_1_facebook_likes**
Value for **actor_1_name** is Tom Cruise
Value for **aspect_ratio** is 2.35
Common values for **aspect_ratio**
Value for **color** is Color
Common values for **color**
Value for **content_rating** is PG-13
Common values for **content_rating**
Same value for **facenumber_in_poster**
Value for **genres** is Action, Thriller
Common values for **genres**

Edge Types

Similar Movies
Same value for **budget**
Value for **genres** is ACTION, ADVENTURE, SCI-FI
Common topics in **plot_summary**

Same Lead Actor
Common values for **actor_1_name**

Fig. 1. Graphiti's Interface: (A) Data Input Row (B) Visualization Canvas (C) Network Metrics Container (D) Node Details Container (E) Quick Access Controls (F) Recommendations Panel (G) Edge Type Panel (H) Download Network Button

Abstract— Network visualizations, often in the form of node-link diagrams, are an effective means to understand relationships between entities, discover entities with interesting characteristics, and to identify clusters. While several existing tools allow users to visualize pre-defined networks, creating these networks from raw data remains a challenging task, often requiring users to program custom scripts or write complex SQL commands. Some existing tools also allow users to both visualize and model networks. Interaction techniques adopted by these tools often assume users know the exact conditions for defining edges in the resulting networks. This assumption may not always hold true, however. In cases where users do not know much about attributes in the dataset or when there are several attributes to choose from, users may not know which attributes they could use to formulate linking conditions. We propose an alternate interaction technique to model networks that allows users to demonstrate to the system a subset of nodes and links they wish to see in the resulting network. The system, in response, recommends conditions that can be used to model networks based on the specified nodes and links. In this paper, we show how such a demonstration-based interaction technique can be used to model networks by employing it in a prototype tool, Graphiti. Through multiple usage scenarios, we show how Graphiti not only allows users to model networks from a tabular dataset but also facilitates updating a pre-defined network with additional edge types.

Index Terms—Network modeling; visual analytics; user interaction;

1 Introduction

Network visualizations, often in the form of node-link diagrams, are an effective means to understand relational patterns between entities, discover entities with prominent structural characteristics, and identify clusters of interest [19]. Given their flexible applicability and general popularity, network visualizations are used in diverse areas, ranging from biological food webs and social networks to computer systems and business ecosystems [27].

There are a plethora of existing tools that allow users to visualize and explore networks and compute relevant metrics[1]. These tools assume that users already have data in a proper network form, perhaps as node and edge lists or in a pre-defined graph markup language. However, this may not always be the case. Most datasets, regardless of how they are collected (e.g., web scraping, user surveys) are not inherently

- Arjun Srinivasan, Alex Endert, and Rahul C. Basole are with Georgia Institute of Technology. E-mail: {arjun010, endert, basole}@gatech.edu.
- Hyunwoo Park is with The Ohio State University. E-mail: park.2706@osu.edu.

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org.
Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx/

[1]http://www.kdnuggets.com/2015/06/top-30-social-network-analysis-visualization-tools.html

a network but are typically in the form of a list of data points with attributes associated with each point. Such data is commonly stored in a tabular format as spreadsheets or relational databases. The process of creating networks from such types of raw data is commonly referred to as "network modeling" [29, 18]. The primary task in modeling a network typically involves specifying two primitives: (1) entities that should represent *nodes* and (2) conditions that determine *edges* between these nodes. Depending on the structure of the dataset and technical expertise of the user, specifying these two primitives can be both challenging and time consuming.

For a given dataset with multiple attributes, the network modeling complexity can be amplified as users could construct a wide range of different networks depending on their driving research questions or hypotheses. A common approach for modeling networks involves either writing custom scripts (e.g., Python or SQL), exporting the resulting tables, and modifying them for subsequent use in existing network visualization tools. This process, even for small datasets, demands significant technical expertise. Moreover, once visualized, users may discover that they need to form different edges to answer specific questions. In such cases, users need to go back and repeat the network modeling and analysis process, frequently switching between different programming environments and visualization systems. This process is tedious and may even discourage users from exploring additional network models. Supporting and accelerating this process is thus not only desirable but also essential to knowledge discovery.

To meet these challenges, a number of tools have offered a graphical user interface that enabled users to both model and visualize networks (e.g., [48, 29, 18]). These tools leverage direct manipulation based interaction or use drag-and-drop designs to allow users to create their network model by specifying attributes they want to use to connect the nodes. In Ploceus [29], for instance, given a tabular data source, the system allows users to add attributes as nodes and connect them if they are co-occurring values in a row of the input table. This notion of connecting attributes works well when users are clear about the linking conditions and the desired edges are based on co-occurring values. However, when users do not know enough about a dataset, or, when there are dozens or even thousands of attributes to choose from, attribute linking conditions may become difficult to specify. Further, while existing tools allow connecting multiple attributes, specifying edge types that are composed of two or more linking conditions to model a multiple edge type (multilayer) network remains challenging.

We propose an alternate approach to modeling networks that allows users to directly communicate (demonstrate) to the system a subset of nodes and links they wish to see in the resulting network. The system, in response, recommends linking conditions that result in a network with the demonstrated nodes and links. With this approach, the task of identifying potential linking conditions is delegated to the system. Users only need to specify their interest in terms of which nodes they wish to see connected and then pick and choose from the suggested conditions to specify edges and construct networks. The notion of letting users demonstrate the desired result that the system can interpret and help create is commonly referred to as *demonstration-based interaction*. This style of interaction has been used in a diverse range of applications (e.g., [22, 42, 23, 37]) and is particularly promising for network modeling.

In this paper, we show how demonstration-based interaction can be applied to the task of network modeling. We show how users can interactively model networks by directly adding or removing links in a node-link visualization. We implement our proposed interaction technique in a prototype tool called *Graphiti*. Each time a link is demonstrated between two nodes, Graphiti presents a ranked list of potential conditions that the user can choose from to define an edge type. New edges are immediately added to the existing node-link visualization allowing rapid exploration of alternate network models. We illustrate the applicability of Graphiti using two typical starting points: (1) modeling networks using a single-table multivariate data and (2) modifying pre-defined networks.

Our contributions are twofold. First, we add to our understanding of demonstration-based interaction by focusing on network/graph visualization. Second, we provide an interactive system that employs the proposed demonstration-based interaction for network modeling. We illustrate our proposed technique and system through two usage scenarios.

2 Related Work

2.1 Network Visualization and Analysis Tools

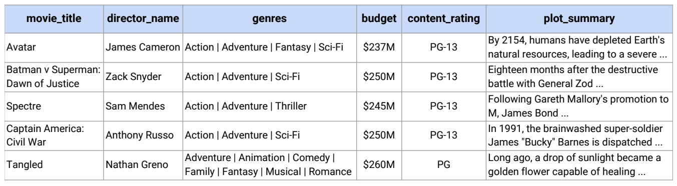
A plethora of systems facilitate network visualization and analysis [19], many of which are available as open source, free, or commercially available software [8, 5, 12, 7, 41], toolkits [32, 39], or research prototypes [3, 40, 24, 26, 44]. All these systems assume that a well-defined network is given as input to the tool. They provide users with a variety of features and visualizations to interactively explore networks. In contrast, our work is focused on the network modeling tasks, which typically come prior to using one of these visualization tools. A complete review of network visualization systems is out of scope of this paper, but can be found in survey reports such as [19, 46, 9].

2.2 Demonstration-based Interaction

The demonstration-based interaction technique has been applied to a wide range of applications. One of the most common applications of the technique in human-computer interaction is programming by demonstration (PbD). Allen Cypher, in his 1993 article [14], states "*The motivation behind Programming by Demonstration is simple and compelling: if a user knows how to perform a task on the computer, that should be sufficient to create a program to perform the task. ... [T]he user should be able to instruct the computer to "Watch what I do", and the computer should create the program that corresponds to the user's actions.*". The motivation behind our work is similar — we want to allow users to "demonstrate" networks they wish to create and enable systems to use the demonstrations and help users model the desired networks.

Other applications of demonstration-based interaction include data cleaning (e.g., [31, 43, 28]), geometrical design (e.g., [22, 21]), 3D drawing (e.g., [20, 42]), data wrangling [23], and more recently visualization construction [37], among others. Among these, the two most relevant applications to our work are Data Wrangler [23] and VisExemplar [37]. Data Wrangler [23] allows users to demonstrate desired changes to a tabular dataset by making direct edits on the table elements (e.g., select and delete empty rows or select a substring to create a new column). In response, the system suggests potential transforms that may be applied to generalize the demonstrated change and update the data table. These suggested transforms are in the form of natural language descriptions that users can refine via interactive parameters. VisExemplar [37] lets users create and switch between visualizations (e.g., bar chart, scatterplot), or map data attributes to graphical encodings (e.g., color, size) by allowing users to demonstrate the desired actions by directly manipulating the visual representation (e.g., moving points or changing color of points). In response, the system estimates user intentions and recommends potential visual transformations (e.g., drawing a scatterplot or mapping color to a data attribute).

We employ a similar interaction technique as VisExemplar in our work and allow users to demonstrate their desired networks by directly manipulating a node-link diagram. In response to users' demonstrations, we identify and recommend potential attribute-based conditions that can be used to model networks based on the demonstrated sub-network. We build upon the ideas presented by Wrangler [23] by presenting recommendations using interactive natural language descriptions. Another recent tool relevant to our work is VISAGE [34]. VISAGE leverages the notion of query by example [49] that allows users to specify a pattern of a sub-network of their interest in terms of node types or specific nodes. The system queries a graph database and returns all sub-networks that match the provided pattern. The primary difference between this approach and ours is that we focus on identifying and presenting conditions so users can model new networks based on incremental demonstrations. VISAGE, on the other hand, focuses on querying graph databases and returning cases that specifically match a given pattern.



movie_title	director_name	genres	budget	content_rating	plot_summary
Avatar	James Cameron	Action | Adventure | Fantasy | Sci-Fi	$237M	PG-13	By 2154, humans have depleted Earth's natural resources, leading to a severe ...
Batman v Superman: Dawn of Justice	Zack Snyder	Action | Adventure | Sci-Fi	$250M	PG-13	Eighteen months after the destructive battle with General Zod ...
Spectre	Sam Mendes	Action | Adventure | Thriller	$245M	PG-13	Following Gareth Mallory's promotion to M, James Bond ...
Captain America: Civil War	Anthony Russo	Action | Adventure | Sci-Fi	$250M	PG-13	In 1991, the brainwashed super-soldier James "Bucky" Barnes is dispatched ...
Tangled	Nathan Greno	Adventure | Animation | Comedy | Family | Fantasy | Musical | Romance	$260M	PG	Long ago, a drop of sunlight became a golden flower capable of healing ...

Fig. 2. Table showing a portion of the IMDB movies dataset.

2.3 Network Modeling Tools

A number of systems have explored the idea of helping users create networks from tabular data. Weaver [48] presented CineGraph showcasing the *attribute relationship graphs* approach and distinguished between *attributed graphs* (where an object is connected to its attributes) and attribute relationship graphs (where attributes are connected based on occurrence). TouchGraph Navigator [2] and Centrifuge [1] are examples of two commercial systems that provide graphical user interfaces to create attribute relationship graphs from data tables. PivotGraph [47] and Honeycomb [45] also support some level of network modeling and aggregate networks by "rolling up" edges based on node attributes. Gilbert and Auber [16] infer hierarchies between table columns and present an interactive visualization of these hierarchies to allow users to select a desired network model that links entities according to shared column values. The Tulip framework by Auber et al. [5, 6] also enables analysis and visualization of relational data. Tulip provides a suite of functions that help developers import data in various formats, generate multiple data models, and create interactive visualizations to explore the data models. Ploceus [29] by Liu et al. allows users to leverage direct manipulation based interaction to model attribute relationship graphs from relational databases. The system provides a suite of filtering, aggregation, and subdivision operations that are defined using relational algebra. Orion [18] also uses relational data tables as its fundamental model and represent networks as edge tables. Orion lets users specify which node types they want to connect in their networks, and recommends potential linking graphs.

Similar to the above listed systems, our work also focuses on the general challenge of helping users model networks. However, our work fundamentally differs from the existing systems in terms of the interaction technique used. We focus on exploring how demonstration-based interaction can be applied to the task of modeling networks. Further, while existing tools largely focus on modeling networks from tabular data, we showcase how our proposed technique can be used to not only model networks from tabular data, but also to add new edge types to pre-defined networks.

3 Challenges in Modeling and Specifying Edges in Networks

Modeling networks is a challenging task and requires users to specify node types and relationships (edges) between these node types. Node types and edges are typically specified based on a single attribute or a set of attributes available in the data [48]. Consider a tabular dataset of movies, a snapshot of which is shown in Figure 2. The table includes information such as director name, budget, summary of the movie's plot, genres the movie belongs to, content rating, among others. This dataset could be modeled as a network in multiple ways, depending on the criteria chosen for nodes and edges.

For instance, one type of network could be a movie network, where movies (nodes) are connected if they were directed by the same director (edges). Another type of network could be an actor co-occurrence network, where actors (nodes) are connected if they appeared in the same movie (edges). This class of networks, where nodes are of a single type (typically defined using a single attribute) are commonly referred to as *unipartite networks*.

Another class of networks that can be modeled may be one where both actors and movies are nodes and they are connected if an actor is a part of a movie (or a movie has an actor). An alternative network could be one with movies, actors, and directors as nodes. Movies are connected to actors and directors if they contain an actor or are directed by a specific director. Such types of networks that have multiple node types with edges connecting different types of nodes are commonly referred to as *k-partite networks*.

A third class of networks could be between nodes of same or different types but may have multiple edge types defined using a combination of attributes. For example, in a network where movies are nodes, two movies can be connected if they share a common actor (first edge type), or if they have a common director (second edge type). Individual edge types can be composed of multiple attributes. So, two movies could be connected if they have the same actor and director (one edge type), or they belong to the same genres (another edge type). This class of networks, where there are edges of different types, is called a *multilayer network* [15].

Most examples presented above are of attribute relationship graphs [48] where edges are defined by co-occurring attribute values. This notion of co-occurrence can be extended to include edges based on ranges of numerical values (e.g., connect movies if their budgets are similar), topics extracted from textual descriptions (e.g., connect two movies if they have common topics in their plots such as a common location), among others. The sheer types of networks and the plethora of ways in which edges can be specified makes modeling networks a challenging task. Even with tools available to help users model networks (e.g., [29, 18]), to create these networks, users are still required to know not only the resulting network they want, but also know the dataset well enough to specify the exact attributes and conditions to specify the nodes and various types of edges between them. Our proposed technique seeks to reduce this effort. We leverage demonstration-based interaction to allow users to directly specify their desired networks and offload the responsibility of finding all possible linking conditions to the system. As a proof-of-concept, we implemented a prototype, Graphiti, that focuses on helping users model unipartite, multilayer networks.

The examples presented above are not exhaustive and are not meant to describe the entire space of network types. Our goal here is to highlight and introduce the problem space of network modeling. For the remainder of this paper, we will use the term ***conditions*** or ***linking conditions*** to refer to attribute based conditions used to determine if two nodes must be connected in a network. For instance, in the case of the movies dataset, in a network where two movies are connected if they have the same director, the condition is "have the *same value* for the attribute *director*". One or more conditions can be used to define an ***edge type***. An example of an edge type in a network where movies are connected to each other can be "same-director-and-lead". This edge type is defined by the conditions "have *same value* for the attribute *director*" and "have *same value* for the attribute *lead actor*".

4 Proposed Technique

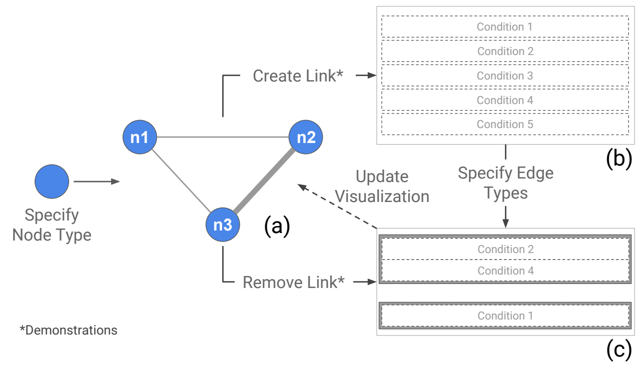


Fig. 3. Proposed technique. (a) Visualization (b) Recommended linking conditions (c) User-specified edge types.

Figure 3 provides a conceptual depiction of the overall workflow of our proposed demonstration-based technique. The interactions in the proposed technique are currently restricted to node-link visualizations. We now briefly discuss the main steps of the proposed technique and how they can be achieved in our prototype system.

Specifying nodes. Users begin by specifying the attribute (e.g., *actor_name*, *movie_title*) that represents node types in their desired networks. We refer to this attribute as the *node type attribute*. Once the attribute is specified, nodes with identical labels are automatically grouped as a single node. This grouping action is commonly referred to as *pivoting* [47] and is described further in a later section. In our prototype, users can specify the node type attribute using a dropdown listing all potential attributes that can be used as nodes in the resulting network. More details about the dropdown and interface are provided in later sections.

Creating links. Once the node type attribute is specified, users can add nodes they wish to use as part of their demonstration to the visualization. To express they want to see two specific nodes connected in the resulting network, users can demonstrate a link between the two nodes in the visualization itself. When a link is drawn between two nodes, the system returns a list of all potential linking conditions (Figure 3b). In our prototype, links can be drawn between two nodes by first clicking on one of the two nodes, and then clicking on the second node. Examples of this interaction and more details about the representation of links (e.g., stroke style, thickness) are discussed in later sections.

Removing links. If users identify a link connecting nodes they do not want to see connected in their resulting network, they can choose to remove the link directly from the visualization. Removing a link from the visualization deletes the edge between two nodes which, in turn, removes the associated edge types. Updated edge types are immediately reflected across the visualization allowing users to rapidly see the effects of their actions. In our prototype, to select a link for removal, users can take one of two approaches. First, users can right-click on a link and select remove link from a context menu that follows the right-click. However, this interaction of directly selecting a link may not be feasible when there are many links and links overlap with each other. In such cases, users can choose to double-click on one of the two nodes that form the link. This highlights all adjacent nodes and connecting links. Users can then right-click on one of the highlighted links and remove it from the canvas. Users can preview the edge types associated with a link before removing a link by simply hovering on the link.

Specifying Edge Types. Users can specify one or more edge types by simply choosing from the suggested list of conditions (Figure 3c). New edge types are immediately added to the visualization, facilitating rapid and iterative exploration of new networks [4]. Linking conditions can be customized and transferred between edge types. Examples showing specification of edge types and the interactions required to do so are discussed in later sections.

5 Recommending Linking Conditions

The core feature of the proposed technique lies in the idea of dynamically generating and suggesting linking conditions based on the users' demonstrations of their desired networks. In our prototype tool, Graphiti, each time a link is demonstrated, the system identifies potential linking conditions using a set of heuristics and recommends them to the user. The computations performed and heuristics applied to generate and suggest the conditions are discussed below.

5.1 Attribute Types and Aggregated Node Tables

Identifying Data Types. When a dataset is first loaded into Graphiti, the system scans the dataset to classify each attribute as one of four data types— *numerical*, *categorical*, *list*, or *textual*. As suggested by their names, numerical and categorical attributes define attributes that are composed of numbers (e.g., *budget*, *duration*) and strings that specify categories (e.g., *movie_title*, *content_rating*) respectively. The list data type defines attributes that are composed of lists of categories (e.g., *genre* in Figure 2). Pipe (|) and semi-colon (;) separated values are considered as list attributes. Attributes that contain text blobs or descriptions are labeled as textual attributes. More details about how classifying attributes into types is used when suggesting linking conditions is discussed in section 5.2.

Computing Aggregated Node Tables. Since Graphiti currently focuses on unipartite networks, it only supports using categorical attributes as node type attributes. As stated earlier, nodes with identical labels are automatically aggregated into a single node. To store the list of aggregated nodes, an aggregated node table is created each time the node type attribute is specified or updated. The values in the aggregated table are computed based on data types of the individual attributes, and are used for all calculations when generating recommendations for linking conditions (discussed in the next section). For list attributes, this table stores a list of all the elements of the individual lists. For categorical attributes, the table stores a single value or a list of values depending on the number of unique categories for an attribute. For numerical attributes aggregated from multiple data points, the aggregated node table stores the sum and average of all values associated with a node. As aggregated values for textual attributes, the system internally extracts topics using Latent Dirichlet allocation (LDA) [11] and stores them as a list. For example, for the movie 'Titanic', the system identifies topics such as 'artist', 'love', 'wet', 'ship'. The system currently extracts and stores a maximum of ten topics per textual attribute. Extracting information like topics from default values enables recommending advanced linking conditions such as co-occurrence of common topics within textual fields.

5.2 Identifying Potential Linking Conditions

To generate the list of all possible linking conditions, the system uses a pre-defined set of heuristics. At a high level, the heuristics compute attribute-based similarities between nodes that have been connected via demonstrations and are defined based on the four supported data types. Algorithm 1 summarizes the overall logic for identifying and recommending linking conditions. Below, we describe how linking conditions are computed and recommended based on the individual data types.

Categorical attributes. Conditions recommended based on categorical attributes are geared to help users model edges that connect nodes if they share specific categories or values (e.g., connect movies if the director is 'Christopher Nolan'), or connect nodes if they share any common value for a particular attribute (e.g., connect movies if they have the same director).

When a link is demonstrated between two nodes that have categorical attributes, a check is made to see if the two nodes have the same values for any categorical attributes. If they have matching values, two conditions are generated: (1) a condition that can be applied to create a link between any two nodes that share the same value for the attribute as the two linked nodes, and (2) a more generalized condition that can be applied to create a link between any two nodes that have the same value for the matched categorical attribute. The condition generation logic for categorical attributes is highlighted in lines 6-10 in Algorithm 1.

For example, if a link was drawn between 'Frozen' and 'Harry Potter and the Chamber of Secrets' as part of a demonstration in a movie network, the conditions recommended based on the categorical attribute *content_rating* would be to connect *movie_title*s if: (1) *content_rating* is PG, and (2) *content_rating* is same.

List and Textual attributes. As stated earlier, textual attributes are internally represented in the aggregated nodes table as lists of extracted topics from text fields. Hence, a common set of rules apply when generating recommendations for linking conditions based on list and textual attributes. Conditions based on these data types are geared to help users model edges that connect nodes if they have common values for list attributes or topics in textual attributes (e.g., connect movies if they share one or more similar topics or entities in their description field), or connect nodes if they have specific common values for list attributes or specific common topics for textual attributes (e.g., connect movies if they belong to the [action,adventure] genres).

When a link is demonstrated between two nodes that have list or textual attributes, a check is made to see if the two nodes have common values within lists or topics in any of these attributes. If they have common values, two conditions are generated: (1) a condition that can be applied to create a link between any two nodes that share the same common values or topics for the attribute as the two linked nodes, and (2) a more generalized condition that can be applied to create a link between any two nodes that have common values for the specific list or textual attribute. The condition generation logic for list and textual attributes is highlighted in lines 11-20 in Algorithm 1.

For example, if a user connected 'Gods of Egypt' and 'Exodus: Gods and Kings', the conditions generated based on extracting topics from the *plot_summary* attribute (textual) and the *genres* attribute (list) would be connect *movie_title*s if: (1) *plot_summary* has common topics, (2) *plot_summary* has topics [egypt], (3) *genres* have common values, and (4) *genres* have values [action, adventure].

Numerical attributes. Conditions recommended based on numerical attributes are geared to help users model edges that connect nodes if they have the same value (e.g., connect movies if they have the same number of actors), or connect nodes if they have significantly close values (e.g., connect movies if their durations are within 5 minutes of each other), or have values that fall under the same bin (e.g., connect movies if they are both low-budget movies). Two values are considered significantly close if the normalized difference δ between them is less than or equal to a user-specified threshold (which, in our usage scenarios is set to *0.3*)

The normalized difference δ between two data points d_i and d_j for the numerical attribute $a_k \in \{a_1, \ldots, a_m\}$ (m= total number of attributes in the dataset) is defined according to Equation 1.

$$\delta(i,j,k) = \frac{|d_{i,k} - d_{j,k}|}{\max(a_k) - \min(a_k)} \tag{1}$$

where $d_{i,k}$ is the value of attribute a_k for data point d_i and $max(a_k)$ and $min(a_k)$ represent the maximum and minimum values respectively of attribute a_k within the dataset. For binning based recommendations, for each numerical attribute, we compute quartiles and use them to identify and group nodes in one of three bins: *low-range* ($< Q_1$), *mid-range* ($Q_1 \leq val \leq Q_3$), and *high-range* ($> Q_3$).

When a link is demonstrated between two nodes that have numerical attributes, a check is made to see if the two nodes have the same, significantly close, or values in the same bin for any numerical attribute. If the nodes have the same value, a condition is generated that can be applied to create a link between any two nodes that have a matching value for the attribute. If two nodes have significantly close values, a condition is generated that can be applied to create a link between any two nodes that have values within (or more than) a certain range of each other. Similarly, if two nodes have values that belong to the same bin (low-range, mid-range, high-range), a condition is generated that can be applied to create a link between all nodes belonging to the same numerical bin. The condition generation logic for numerical attributes is highlighted in lines 21-37 in Algorithm 1. The *[relational_operators]* (line 28, Algorithm 1) are essentially equals (=), less than equals (≤), and greater than equals (≥).

For example, if the movies 'Toy Story 3' and 'Cars 2' were connected via a demonstration, the conditions generated based on the numerical attributes *budget* and *duration* are to connect *movie_title*s if: (1) *budget* is same, (2) *budget* is in the same range, (3) *budget* is in *mid-range*, (4) Difference between *duration* [=, ≤, ≥] 3, (5) *duration* is in the same range, and (6) *duration* is in *mid-range*.

Note that while the examples talk about single link demonstrations, multiple links can be created between more than two nodes as a part of a single demonstration. In such cases, the same rules are applied to generate conditions but every time a link is drawn, each condition is checked to verify if it satisfies all the demonstrated links. If not, the condition is removed from the list of suggestions and the list is iteratively refined. If, for a demonstration, none of the node attributes satisfy the heuristics mentioned above, the system would offer no recommendations.

Algorithm 1: generateLinkingConditions(node1,node2)

```
  Data: Demonstrated link [passed as a node tuple (node1,node2)]
  Result: Suggested Conditions
1 conditions = []
2 for attribute in datasetAttributes do
3     if attribute == nodeTypeAttribute then
4         continue
5     else
6         if attribute.type is Categorical then
7             val1, val2 = values for node1, node2
8             if val1 == val2 then
9                 conditions.add(link nodes based on common value for
                    attribute)
10                conditions.add(link nodes if attribute has value val1)
11        else if attribute.type is List or Textual then
12            lst1, lst2 = values or topics for node1, node2
13            vals = lst1.intersection(lst2)
14            if vals.length > 0 then
15                if attribute.type is List then
16                    conditions.add(link nodes based on common values for
                        attribute)
17                    conditions.add(link nodes if attribute has values [vals])
18                else if attribute.type is Textual then
19                    conditions.add(link nodes based on common topics in
                        attribute)
20                    conditions.add(link nodes if attribute has topics [vals])
21        else if attribute.type is Numerical then
22            num1, num2 = values for node1, node2
23            normDiff =
                getNormalizedDifference(num1,num2,attribute)
24            if normDiff <= threshold then
25                if normDiff == 0 then
26                    conditions.add(link nodes if they have same value for
                        attribute)
27                else
28                    conditions.add(link nodes if difference in values for
                        attribute is [relational_operators] |num1-num2|)
29            if (num1 < Q_1) and (num2 < Q_1) then
30                conditions.add(link nodes if attribute has values in the
                    same range group)
31                conditions.add(link nodes if attribute has low-range
                    values)
32            else if (Q_1 <= num1 <= Q_3) and (Q_1 <= num2 <= Q_3)
                then
33                conditions.add(link nodes if attribute has values in the
                    same range group)
34                conditions.add(link nodes if attribute has mid-range
                    values)
35            else if (Q_3 < num1) and (Q_3 < num2) then
36                conditions.add(link nodes if attribute has values in the
                    same range group)
37                conditions.add(link nodes if attribute has high-range
                    values)
38 return conditions
```

5.3 Presenting and Ranking Linking Conditions

Similar to Wrangler's [23] presentation of suggested transforms, to aid comprehension of recommended linking conditions, we generate short, interactive natural language descriptions for each linking condition. Figure 4a shows an example of some descriptions. Attribute names are shown in bold and values specific to the presented demonstration are highlighted using blue, underlined text. Users can browse through the list of recommendations and promote conditions they are interested in to edge types that will be used in their resulting networks.

As they are generated, each linking condition is assigned a match score in the range [0-1] where 1 indicates a perfect match for an attribute value. When recommended, conditions are ranked based on their match score and recency of the user's interaction with a condition's attribute type. Conditions having the same score are sorted alphabetically based on the labels of attributes the conditions represent.

6 GRAPHITI

6.1 User Interface

Figure 1 highlights Graphiti's user interface. The data input row (Figure 1A) on top lets users select the node type attribute, add specific nodes, or add all nodes in the dataset to the visualization canvas. For larger datasets, Graphiti randomly samples a subset (500 nodes) of the input data to add to the canvas. This is done to avoid rendering issues associated with large-sized networks.

The visualization canvas (Figure 1B) displays a node-link diagram rendered using D3's force-directed layout [13]. The canvas also acts as a sandbox-like environment for users to provide demonstrations of networks they wish to create. The analytics panel (Figure 1C) computes and displays network metrics such as density, and average clustering coefficient. The details panel (Figure 1D) presents details of individual nodes when nodes are hovered upon. Quick access icons (Figure 1E) allow users to remove unconnected nodes (⊛), pause/simulate the force layout (❚❚), re-center the visualization (⊠), and adjust link opacity (⚙).

The recommendations panel (Figure 1F) is dynamically populated when a user demonstrates a link on the visualization canvas. This panel displays the list of potential linking conditions that can be used to include the demonstrated links as a part of a modeled network.

Users can pick and choose conditions from the recommendations panel and add them to the edge type panel (Figure 1G). This panel allows users to specify multiple edge types based on recommended or manually specified linking conditions (discussed in section 6.3).

6.2 Specifying Edge Types

Graphiti lets users specify edge types by allowing them to pick and choose from recommended conditions. Edge types are defined in the edge type panel (Figure 1G). Each edge type is composed of one or more linking conditions. Conditions within an edge type are logically separated by an AND operator. The edge types themselves are logically separated by an OR operator. For example, Figure 4b shows a snapshot of the edge type panel with two edge types. The first edge type ("Similar Movies") is defined by the conditions specifying that movies should be connected if they have the same *budget*, have the *genres* [Action,Adventure,Sci-Fi], and have common topics in their *plot_summary*. The second edge type ("Same Lead Actor") is defined using a single condition: connect movies if they have the same value for *actor_1_name*.

To add a condition from the recommendations panel as a new edge type, users can click the first add (⊞) button to the right of each condition (Figure 4a). Alternatively, users can select the edge type they want to append a condition to using a dropdown list of edge types provided by the second add condition button (Figure 4a). Users can transfer conditions between edge types or promote a nested condition within an edge type to a new edge type via drag-and-drop operations. The thickness of a link in the visualization canvas indicates the number of edge types it is composed of. This is highlighted in the representative Figure 3, the link between nodes n2 and n3 is thicker to indicate it encompasses two edge types while other links are of a single type. This is also shown in Figure 8a.

Conditions with demonstration specific values become interactive when they are added to the edge type panel. For instance, Figure 4c shows how conditions in Figure 4a become interactive when promoted to the edge type specification panel. Currently supported interactive widgets include dropdowns for operators (e.g., equals, less than equals) and categorical or list values, and sliders for numerical values. Changes made to conditions using these interactive widgets are immediately reflected on the visualization canvas. By facilitating this dynamic query style interaction [4] with the visualization canvas, Graphiti allows users to rapidly and iteratively explore new networks without requiring them to switch between dialog boxes or tabs as in the case of some existing tools (e.g., [18, 29]).

Networks may look drastically different based on which types of edges are shown [33, 35]. Graphiti allows users to enable/disable edge types to see their effect on the modeled network. To do so, users can toggle the enable/disable (👁/⊘) button associated with each edge type (Figure 4b). Graphiti also allows users to assign labels to edge types in the specification panel itself. By default, each new edge type is assigned the label "New Edge Type $< count >$" where *count* represents the number of edge types created in a session.

6.3 Other Features

Expanding nodes. To facilitate smoother exploration, once edge types are specified, Graphiti allows users to expand nodes on the canvas. Expanding a node adds nodes linked to the particular node based on the specified edge types. To expand a node, users can right-click on a node and select "Expand" from a context menu. Expanding to see connected nodes is particularly helpful when users are performing bottom-up exploration. That is, in scenarios where a user begins with a small number of nodes and constructs links based on connections between only those nodes, the ability to add directly connected nodes helps the user explore the network one node's neighborhood at a time.

Manual Condition Specification. Graphiti's primary feature is its recommendation of linking conditions for demonstration-based interaction. However, to offer a more complete user experience and workflow, Graphiti also allows manual specification of linking conditions using widgets like dropdowns and sliders. Manual conditions can be specified by clicking the "Specify a Manual Condition" button in the edge type panel (Figure 1G). Figure 4d shows an example of manual condition specification. As attributes and parameters for the condition are selected, the system keeps presenting the remaining parameters to specify a condition. Manually specified conditions get added as a new edge type at the bottom of the edge type panel by default.

Having the option of manual condition specification along with demonstration-based interaction can be particularly helpful in cases where users have some pre-conceived hypothesis of the network they wish to construct in terms of attributes. In such cases, users may begin with the manual specification approach and then switch to using a demonstration-based approach as they explore the network visualization and find entity pairs they wish to see connected/disconnected. Alternatively, users could begin with demonstration-based interaction and as they get a better sense of attributes and values they are interested in, they can switch to the manual specification to add new edge types.

7 USAGE SCENARIOS

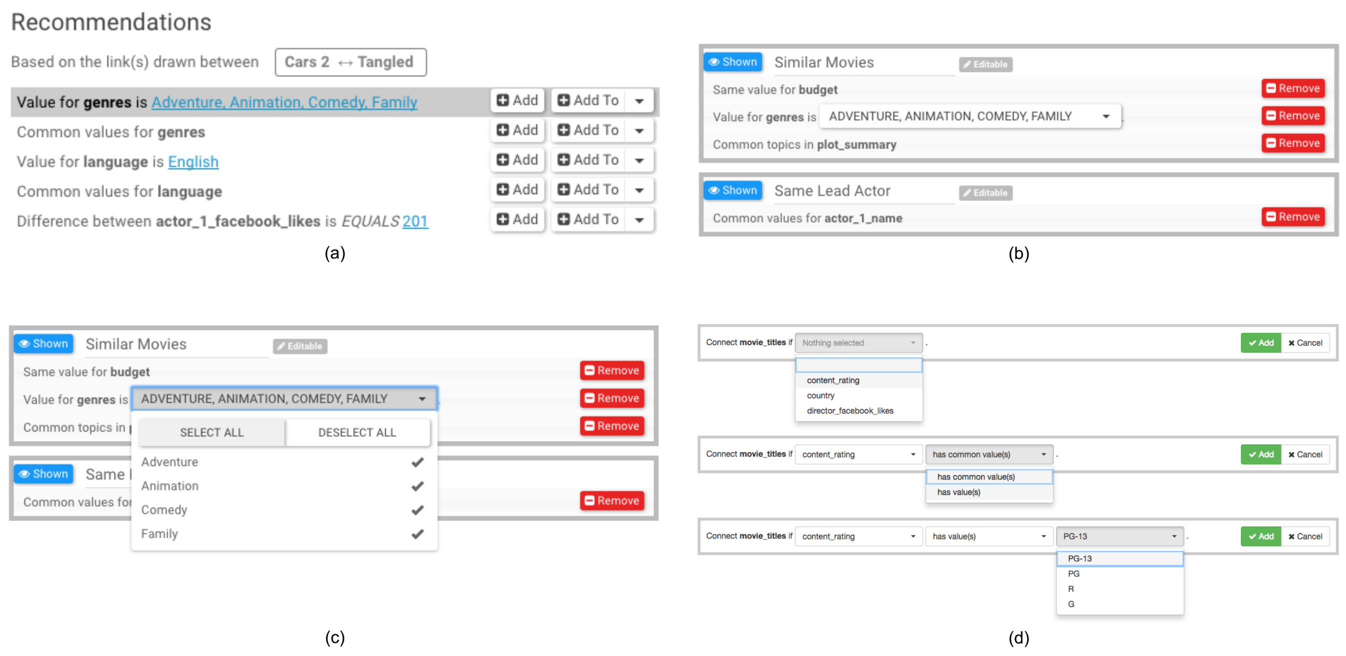
7.1 Modeling Networks from Tabular Data

Consider the following prototypical usage scenario of Graphiti. Sarah is a film enthusiast and has downloaded a csv file of the aforementioned movies dataset (see Figure 2). Sarah imports the file into Graphiti to explore the data and model potential networks.

Having recently watched "Batman vs Superman: Dawn of Justice" and "Captain America: Civil War", Sarah begins her analysis by exploring potential connections between these two movies that represent parallel comic universes – DC and Marvel, respectively.

She begins by selecting *movie_title* from the node type attribute specification dropdown (Figure 1A), indicating that she wants to use movies as nodes in the network. She adds the two movies to the visualization canvas by typing the titles into the input textbox. Sarah then connects the two movies by clicking on the first node, in this case "Batman vs Superman: Dawn of Justice' and then "Captain America: Civil War", drawing a temporary link (shown using a dotted line) between the two nodes (Figure 5a).

In response, the system recommends a list of conditions that she can use to connect the two movies. While scanning through the list of recommendations, three conditions capture Sarah's interest: same value for *budget*, common *genres* of [action, adventure, sci-fi], and common topics in *plot_summary*. She uses these three conditions to


Recommendations

Based on the link(s) drawn between Cars 2 ↔ Tangled

Value for **genres** is Adventure, Animation, Comedy, Family

Common values for **genres**

Value for **language** is English

Common values for **language**

Difference between **actor_1_facebook_likes** is *EQUALS* 201

Similar Movies: Same value for **budget**; Value for **genres** is ADVENTURE, ANIMATION, COMEDY, FAMILY; Common topics in **plot_summary**

Same Lead Actor: Common values for **actor_1_name**

Connect **movie_titles** if content_rating has value(s) PG-13

Fig. 4. Interactive conditions and edge type specification widgets in Graphiti. (a) Example of recommended linking conditions (b) Example of two user-specified edge types (c) Interactive condition widgets (d) Manual condition specification example.

define a new edge type (Figure 5b). This changes the line connecting the two movies to a solid gray line indicating that the link is no longer temporary and is drawn based on a specified edge type (Figure 5b).

Next, to see other movies that are connected based on the same three conditions, Sarah adds all movies in her dataset to the canvas using the "Load all" button (Figure 1A). This updates the canvas with a node-link visualization showing all movies in the dataset as nodes. Given the edge condition she created, two nodes are connected if they belong to the genres of [action, adventure, sci-fi], were produced with the same budget, and have common topics in their plot summary.

Observing that there are several nodes that have no connections, Sarah clicks the quick access icon (⊛) to remove all nodes that have no connections and gets to a network shown in Figure 5c. Sarah explores the network and identifies many subsets of movies that meet her chosen edge criteria: Kung Fu Panda 3—The Mummy: Tomb of the Dragon Emperor, Harry Potter and the Order of the Phoenix—Frozen—Harry Potter and the Goblet of Fire, among others.

As she explores the network, Sarah comes across three movies starring actor Tom Cruise— "Minority Report", "Mission Impossible–Rogue Nation", and "Mission Impossible III". Sarah notices that out of these three, only two ("Mission Impossible–Rogue Nation" and "Mission Impossible III") are connected. This intrigues her to see how the network may change if "Minority Report" was connected to the two Mission Impossible movies. To explore this, Sarah draws a link between "Minority Report" and "Mission Impossible– Rogue Nation". This leads to the system recommending a new set of linking conditions (Figure 5d). Sarah scrolls through the list of recommendations and adds the condition same values for *actor_1_name* as a new edge type. This updates the visualization to a network where movies are now connected based on her earlier specified edge type (with three conditions), the new edge type (same actor condition), or both. While scanning the resulting network, Sarah notices a thicker link (reflecting both edge types) connecting only two movies (Harry Potter and the Goblet of Fire, and Harry Potter and the Order of the Phoenix).

Curious to see the trend across the entire dataset, Sarah again adds all movies in her dataset to the canvas resulting in a network shown in Figure 1. Scanning this network Sarah realizes to her surprise that the only pair of movies in the dataset that meet both edge types she specified are, in fact, the two Harry Potter movies she identified earlier. To analyze the modeled network using different layouts and metrics provided by advanced network analysis tools (e.g., Gephi [7]), Sarah labels the two edge types as "Similar Movies" and "Same Lead Actor" by simply typing them into the edge type label box (Figure 1G). Sarah exports the network as a set of node and edge csv files for external use using the "Download network" button (Figure 1H).

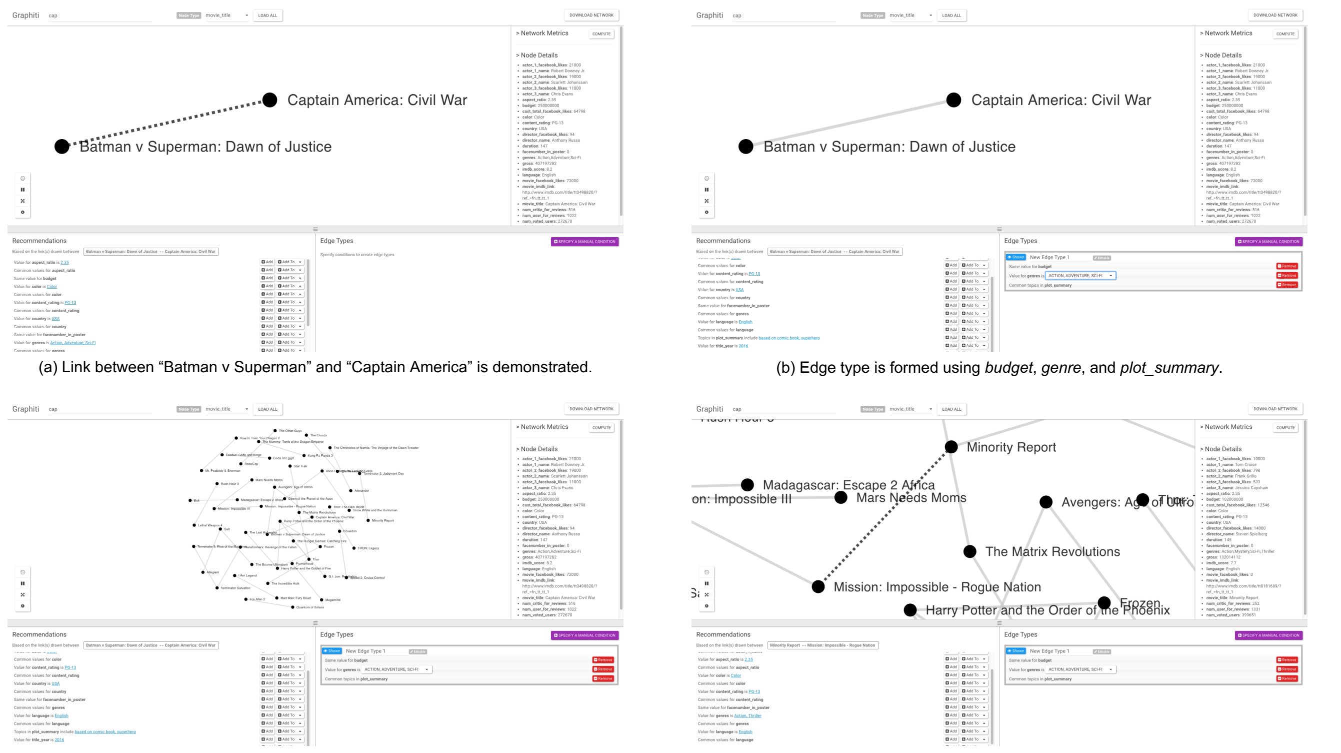
7.2 Adding New Edge Types to Pre-defined Networks

A key advantage of the proposed demonstration-based technique is that it can also be used to modify networks that have already been used in other visualization tools, and have a pre-defined network data structure. The ability to add new attribute-based edges to networks that already have edges defined based on external data or knowledge (e.g., alliance, friendship) can help users explore the network from a different perspective and gain new insights not previously possible with pre-specified edge types. To illustrate this use case, we present a second usage scenario.

Consider Alberto, an Italian data analyst, analyzing a pre-defined alliance network of European and Middle Eastern energy companies. He imports the network as a set of node and edge files. There are multiple attributes associated with each company (node) including *country*, *allSegments (segment affiliation)*, *lat/long (geographic information)*. A snapshot of the imported node file is shown in Figure 6. The edge file consists of a list of node pairs, representing alliances (link) between two companies (nodes).

By default, Graphiti uses a force-directed layout to render the nework as a node-link visualization. Upon exploring the current network, Alberto's interest is peaked when he notices that two well-known Italian energy companies, "SEL SpA" and "Iride SpA", are not connected to each other (i.e., they have not formed an alliance). Curious by this finding, Alberto wonders what inter- and intra-country alliance patterns might exist and how the overall alliance network structure of this energy ecosystem would transform if companies from the same country where grouped together.

Alberto draws a link between the two aforementioned Italian companies. Similar to the scenario with a tabular dataset, this leads to a set of recommended linking conditions (Figure 7a). To see the general trend of intra- and cross- country alliance patterns, Alberto adds the condition: common values for *country* as a new edge type. This immediately updates the visualization to show three types of links: *pre-defined*, *user-specified*, and *hybrid* (Figure 8).


Graphiti
Captain America: Civil War
Batman v Superman: Dawn of Justice
Network Metrics
Node Details
Recommendations
Edge Types
Minority Report
Madagascar: Escape 2 Africa
Mars Needs Moms
Avengers: Age of Ultron
The Matrix Revolutions
Mission: Impossible - Rogue Nation
Harry Potter and the Order of the Phoenix
Frozen

(a) Link between “Batman v Superman” and “Captain America” is demonstrated.

(b) Edge type is formed using *budget*, *genre*, and *plot_summary*.

(c) Only the sub-network containing the specified edge type is shown.

(d) Link between “Minority Report” and “Mission Impossible: Rogue Nation” is demonstrated.

Fig. 5. Screenshots from a usage scenario showing how Graphiti can be used to model networks from a tabular dataset.

Pre-defined links that were loaded directly from the input-edge file are colored blue. User-specified links, which represent links that were formed based on edge types specified during the session, are colored gray. Hybrid links that contain both edge types (i.e., pre-specified and user-specified) are portrayed using thicker lines (to indicate multiple edge types) and are colored maroon.

name	primarySegment	allSegments	country	SIC	latitude	longitude
Vyborg Shipyard JSC	Manufacturing	Manufacturing	Russian Fed	3731	60.7092159	28.7440507
SEL SpA	Transportation, Communications, Electric, Gas, And Sanitary Services	Transportation, Communications, Electric, Gas, And Sanitary Services	Italy	4911	46.4981125	11.3547801
Aksoy Holding AS	Mining	Wholesale Trade; Construction; Mining; Finance, Insurance, And Real Estate;	Turkey	1311	41.0096334	28.9651646
Petrojet	Services	Services	Egypt	8711	30.0488185	31.2436663

Fig. 6. SDC energy nodes table snapshot

Based on the updated visualization showing both pre-defined and the newly specified edge type (Figure 7b), Alberto makes several observations about the network that he could not have made so easily by only considering the links in the input network file. First, he notices several clusters of nodes indicating the different countries in the dataset. The largest clusters (countries with the most companies) represent Russia, Italy, and Turkey.

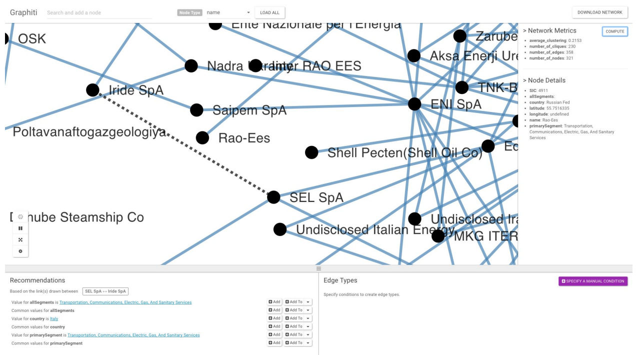
Upon further exploration, Alberto observes that while there are many intra-country alliances between Russian and Italian companies, Turkish companies tend to form much fewer alliances with other Turkish companies. Examining the density of blue links, Alberto also notices that Russian and Italian companies tend to form a high number of cross-country alliances. Looking for similar patterns between countries, Alberto considers other clusters and continues with his analysis.

8 Discussion

Incorporating demonstration-based interaction into network visualization tools. Graphiti is only one example showcasing how demonstration-based interaction can be incorporated into network visualization tools. With the usage scenario presented in section 7, we show how the proposed interaction can be used in the context of a pre-defined network. Thus far, we have shown how some demonstration-based interactions may be incorporated within node-link diagrams. However, the proposed technique is more general and can be applied to other network representations. For example, given a matrix representation of a network where cells are filled if there is a link between two nodes (rows and columns of the matrix), users may click an empty cell to add a new link and the system can generate recommendations similar to Graphiti. Given this potential scalability of the idea and the presented proof-of-concept for node-link visualizations, an area for further exploration lies in facilitating demonstration-based interactions in existing and emerging network visualization tools. This will allow users to modify their networks dynamically during exploration and potentially enable more insights during the analysis process.

Recommending more than just linking conditions. In the presented work, we leverage demonstration-based interactions such as creating and removing links to recommend potential linking conditions that help users specify edge types and model networks. However, the proposed technique can be expanded to offer more types of suggestions to help users perform tasks such as data cleaning and clustering using a network metaphor. For example, an application of the proposed technique could be in cleaning data by removing duplicate entities from a dataset. Consider a tool like D-Dupe [10] that helps in resolving duplicate entities. The tool operates primarily via menu-driven interfaces, requiring users to explicitly parametrize the matching algorithms. An alternative to this could be to allow users to directly connect entities that users think should be merged. In response, the system can scan results of multiple algorithms to identify the parameters and perform the matching.

Striking a balance between parametrization and suggestions. The proposed interaction aims at striking a balance between the user and the system in terms of operations performed. For example, in Graphiti, users provide a demonstration, and the system recommends linking conditions based on those demonstrations. This is followed by the user specifying edge types, and the system immediately incorporating them to the visualization, and so on. Another approach could be one where Graphiti directly recommends networks (instead of potential conditions) based on the user's demonstrations. Striking


Graphiti

Recommendations

Based on the link(s) drawn between SEL SpA ↔ Iride SpA

Value for **allSegments** is Transportation, Communications, Electric, Gas, And Sanitary Services

Common values for **allSegments**

Value for **country** is Italy

Common values for **country**

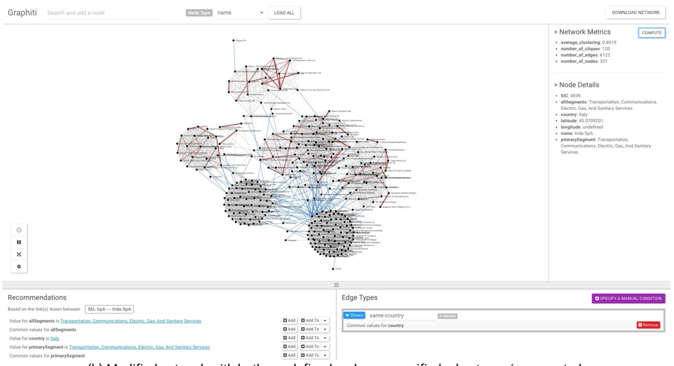
Value for **primarySegment** is Transportation, Communications, Electric, Gas, And Sanitary Services

Common values for **primarySegment**

Edge Types

Specify conditions to create edge types.

(a) Link between “SEL SpA” and “Iride SpA” is demonstrated.


Graphiti

> Network Metrics

- average_clustering: 0.8919
- number_of_cliques: 120
- number_of_edges: 6122
- number_of_nodes: 321

Recommendations

Based on the link(s) drawn between SEL SpA ↔ Iride SpA

Value for **allSegments** is Transportation, Communications, Electric, Gas, And Sanitary Services

Common values for **allSegments**

Value for **country** is Italy

Common values for **country**

Value for **primarySegment** is Transportation, Communications, Electric, Gas, And Sanitary Services

Common values for **primarySegment**

Edge Types

same-country

Common values for **country**

(b) Modified network with both pre-defined and user-specified edge types is generated.

Fig. 7. Screenshots from a usage scenario showing how Graphiti can be used to add new edge types to a predefined network.

the balance between steps taken by the user and the system, and understanding pros and cons of different approaches remains an open area for future exploration. Further, exploring alternative ways of presenting recommendations and enabling the right level of abstraction in recommendations are open challenges.

9 Limitations and Future Work

In working with sample tabular and network datasets from varying domains we have already identified situations that highlight potential limitations of the tool and areas for future work. For example, Graphiti currently only considers four types of attributes: numerical, list, categorical, and textual when recommending conditions. While these cover a large portion of possible attribute types, this limits the system in terms of its recommendation and aggregation capabilities for other attribute types such as geographical values (e.g., latitude/longitude, city) and temporal data (e.g., dates, timestamps). Temporal and geographical attributes are currently treated as categorical attributes. By also considering these additional attribute types, Graphiti can support other aggregation operations like proximity grouping [29].

As a proof-of-concept of the proposed technique, we have focused on using it to model undirected, unipartite, and multilayer networks. While there are several use cases that are supported by these types of networks (e.g., projected networks [25], similarity networks [17]), lack of support for other classes of networks (e.g., multipartite, temporal) restricts the types of questions that can be asked. For example, with the IMDB dataset, users may want to see a network where both directors and movies are nodes and a director is connected to all movies he has directed. Also, extending the system capabilities to support relational databases and multiple data sources will enhance the potential list of practical applications.

In terms of the recommendations, we currently show all linking conditions generated based on the logic presented in Algorithm 1. However, additional logic could be used to limit the conditions to more useful or interesting attributes. One way of doing this could be to compute network metrics such as density and number of clusters for networks that would be constructed if a recommendation was accepted. These metrics could then be used to score the “usefulness” or “interestingness” of networks (e.g., networks with more clusters are more interesting). These scores, in turn, could be used to rank and filter the recommendations themselves.

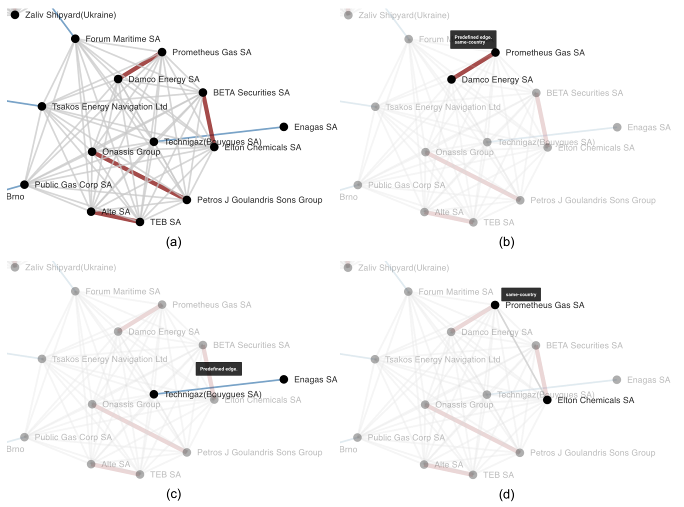


Fig. 8. Link types in Graphiti. (a) Screenshot showing all link types (b) Hybrid link highlighted (c) Pre-defined link highlighted (d) User-specified link highlighted.

In terms of the visualization, Graphiti presently only uses a node-link diagram as compared to existing tools (e.g., [29, 18]) that also use alternative representations such as matrices [18] and connected lists [29]. Including these alternative visualizations can also help circumvent the rendering challenges associated with node-link diagrams. Since networks resulting from certain attribute combinations can be very dense, making it easier to remove links and interacting with dense subgraphs using alternative interaction techniques (e.g., [30]) is another area for improvement. Graphiti currently does not support the notion of edge weights based on attribute values. Adding the option to assign weights to edges based on attributes could help users in exploring more aspects of relationships between entities in a dataset. While Graphiti supports specifying multiple edge types (multilayer networks), the graphical representation of links does not directly show the multiple edge types. Facilitating enhanced analysis of multilayer networks (e.g., using substrate and catalyst networks [35]) is another area for improvement.

Lastly, as highlighted in the usage scenarios, to test the proposed technique and Graphiti’s features we explored a variety of datasets from multiple domains. However, evaluating Graphiti in terms of both usability and user experience metrics [36, 38], and conducting studies with experts to model networks in practical scenarios for specific domains such as healthcare (e.g., creating similarity networks between patients) or finance (e.g., creating networks between venture capital companies) would further help validate the strengths and weaknesses of the proposed technique and prototype.

10 Conclusion

Modeling networks can be a difficult and time-consuming task. People must understand their data, and formalize their specification for how to create edges in order to visually explore the network. We present a demonstration-based interaction technique for network modelling. Our technique enables users to demonstrate links between two nodes, from which the system computes and recommends potential linking conditions that can be combined to form edges. We implement our technique into a visual analytic prototype, Graphiti, to demonstrate its effectiveness. Finally, we present two usage scenarios which illustrate the intended usage and functionality.

Acknowledgments

This research was funded in part by the Tennenbaum Institute and a Georgia Tech SPAG Award.

References

[1] Centrifuge systems. http://centrifugesystems.com/, 2017.

[2] Touchgraph navigator: graph visualization and social network analysis software. http://www.touchgraph.com/navigator, 2017.

[3] J. Abello and J. Korn. Mgv: A system for visualizing massive multi-digraphs. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):21–38, 2002.

[4] C. Ahlberg and B. Shneiderman. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 313–317. ACM, 1994.

[5] D. Auber. Tulipa huge graph visualization framework. *Graph drawing software*, pages 105–126, 2004.

[6] D. Auber, D. Archambault, R. Bourqui, M. Delest, J. Dubois, B. Pinaud, A. Lambert, P. Mary, M. Mathiaut, and G. Melancon. Tulip III. In *Encyclopedia of Social Network Analysis and Mining*, pages 2216–2240. Springer, 2014.

[7] M. Bastian, S. Heymann, M. Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *ICWSM*, 8:361–362, 2009.

[8] V. Batagelj and A. Mrvar. Pajek-program for large network analysis. *Connections*, 21(2):47–57, 1998.

[9] F. Beck, M. Burch, S. Diehl, and D. Weiskopf. A taxonomy and survey of dynamic graph visualization. In *Computer Graphics Forum*. Wiley Online Library, 2016.

[10] M. Bilgic, L. Licamele, L. Getoor, and B. Shneiderman. D-dupe: An interactive tool for entity resolution in social networks. In *IEEE Symposium On Visual Analytics Science And Technology*, pages 43–50. IEEE, 2006.

[11] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[12] S. P. Borgatti, M. G. Everett, and L. C. Freeman. Ucinet for windows: Software for social network analysis. 2002.

[13] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

[14] A. Cypher and D. C. Halbert. *Watch what I do: programming by demonstration*. MIT press, 1993.

[15] M. De Domenico, A. Solé-Ribalta, E. Cozzo, M. Kivelä, Y. Moreno, M. A. Porter, S. Gómez, and A. Arenas. Mathematical formulation of multilayer networks. *Physical Review X*, 3(4):041022, 2013.

[16] F. Gilbert and D. Auber. From databases to graph visualization. In *Information Visualisation (IV), 2010 14th International Conference*, pages 128–133. IEEE, 2010.

[17] R. A. Hanneman and M. Riddle. Introduction to social network methods, 2005.

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

[19] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on visualization and computer graphics*, 6(1):24–43, 2000.

[20] T. Igarashi and J. F. Hughes. A suggestive interface for 3d drawing. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 173–181. ACM, 2001.

[21] T. Igarashi, S. Kawachiya, H. Tanaka, and S. Matsuoka. Pegasus: a drawing system for rapid geometric design. In *CHI 98 conference summary on Human factors in computing systems*, pages 24–25. ACM, 1998.

[22] T. Igarashi, S. Matsuoka, S. Kawachiya, and H. Tanaka. Interactive beautification: a technique for rapid geometric design. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 105–114. ACM, 1997.

[23] 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*, pages 3363–3372. ACM, 2011.

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

[25] M. Latapy, C. Magnien, and N. Del Vecchio. Basic notions for the analysis of large two-mode networks. *Social networks*, 30(1):31–48, 2008.

[26] B. Lee, C. S. Parr, C. Plaisant, B. B. Bederson, V. D. Veksler, W. D. Gray, and C. Kotfila. Treeplus: Interactive exploration of networks with enhanced tree layouts. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1414–1426, 2006.

[27] M. Lima et al. Visual complexity. *source: http://www. visualcomplexity. com*, 2007.

[28] J. Lin, J. Wong, J. Nichols, A. Cypher, and T. A. Lau. End-user programming of mashups with vegemite. In *Proceedings of the 14th international conference on Intelligent user interfaces*, pages 97–106. ACM, 2009.

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

[30] M. J. McGuffin and I. Jurisica. Interaction techniques for selecting and manipulating subgraphs in network visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):937–944, 2009.

[31] R. C. Miller and B. A. Myers. Interactive simultaneous editing of multiple text regions. In *USENIX Annual Technical Conference, General Track*, pages 161–174, 2001.

[32] J. OMadadhain, D. Fisher, S. White, and Y. Boey. The jung (java universal network/graph) framework. *University of California, Irvine, California*, 2003.

[33] A. Perer and B. Shneiderman. Balancing systematic and flexible exploration of social networks. *IEEE Transactions on Visualization and Computer Graphics*, 12(5), 2006.

[34] R. Pienta, A. Tamersoy, A. Endert, S. Navathe, H. Tong, and D. H. Chau. Visage: Interactive visual graph querying. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pages 272–279. ACM, 2016.

[35] B. Renoust, G. Melançon, and T. Munzner. Detangler: Visual analytics for multiplex networks. In *Computer Graphics Forum*, volume 34, pages 321–330. Wiley Online Library, 2015.

[36] B. Saket, A. Endert, and J. Stasko. Beyond usability and performance: A review of user experience-focused evaluations in visualization. *Beyond Time And Errors: Novel Evaluation Methods For Visualization*, 2016.

[37] B. Saket, H. Kim, E. T. Brown, and A. Endert. Visualization by demonstration: An interaction paradigm for visual data exploration. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):331–340, 2017.

[38] B. Saket, C. Scheidegger, and S. Kobourov. Comparing node-link and node-link-group visualizations from an enjoyment perspective. In *Computer Graphics Forum*, volume 35, pages 41–50, 2016.

[39] D. A. Schult and P. Swart. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conferences (SciPy 2008)*, volume 2008, pages 11–16, 2008.

[40] B. Shneiderman and A. Aris. Network visualization by semantic substrates. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):733–740, 2006.

[41] M. E. Smoot, K. Ono, J. Ruscheinski, P.-L. Wang, and T. Ideker. Cytoscape 2.8: new features for data integration and network visualization. *Bioinformatics*, 27(3):431–432, 2010.

[42] S. Tsang, R. Balakrishnan, K. Singh, and A. Ranjan. A suggestive interface for image guided 3d sketching. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 591–598. ACM, 2004.

[43] R. Tuchinda, P. Szekely, and C. A. Knoblock. Building mashups by example. In *Proceedings of the 13th international conference on Intelligent user interfaces*, pages 139–148. ACM, 2008.

[44] F. van Ham and A. Perer. Search, show context, expand on demand: Supporting large graph exploration with degree-of-interest. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):953–960, 2009.

[45] F. Van Ham, H.-J. Schulz, and J. M. Dimicco. Honeycomb: Visual analysis of large scale social networks. In *IFIP Conference on Human-Computer Interaction*, pages 429–442. Springer, 2009.

[46] T. Von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J.-D. Fekete, and D. W. Fellner. Visual analysis of large graphs: state-of-the-art and future research challenges. In *Computer graphics forum*, volume 30, pages 1719–1749. Wiley Online Library, 2011.

[47] M. Wattenberg. Visual exploration of multivariate graphs. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 811–819. ACM, 2006.

[48] C. Weaver. Multidimensional data dissection using attribute relationship graphs. In *Visual Analytics Science and Technology (VAST), 2010 IEEE Symposium on*, pages 75–82. IEEE, 2010.

[49] M. M. Zloof. Query-by-example: A data base language. *IBM systems Journal*, 16(4):324–343, 1977.