Integrating Querying and Browsing in Partial Graph Visualizations

Adam Perer and Frank van Ham

IBM Research

Abstract

Partial graph visualizations, sometimes also referred to as online graph visualizations, are visual representations of graphs that show only connections around a particular point of interest. They are often a useful, less complex alternative to visual representations of graphs as a whole or can serve as visualizations when the whole data is not accessible at once. Their biggest downside is that users quickly lose orientation because they never get a sense of the structure of the whole graph. In this paper, we introduce techniques that improve user exploration of partial graphs by integrating both browsing and querying information finding paradigms. From a given partial graph, users can specify queries relevant to user tasks and the results will be displayed as visual aids that helps users navigate representations of graphs with a limited look-ahead. Much like signposts in the physical world, these graphcues highlight the shortest path to relevant nodes from the user's current location in the graph. We demonstrate these concepts on a social network of approximately 500,000 people and 30 million relationships.

Categories and Subject Descriptors**:** H.5.2 [User Interfaces]: User Interfaces – Information Visualization

1 Introduction

In a world where a substantial part of the data flood comes in the form of abstract graphs (or networks) of ever increasing size, many data-centric professions are struggling to keep up. For instance, banks that investigate fraudulent transactions must now deal with millions of transactions between individual accounts and a typical software project contains thousands of dependencies between modules. Visual representations are essential to gaining insight into these interconnected structures. However, for network data, typical visualizations like node-link diagrams often lack such scalability. Most node-link diagrams become a chaotic web of overlapping nodes and tangled edges when displaying even just a few hundred nodes. Furthermore, many analysts may not be interested in global patterns but instead are trying to learn something more about the structures around particular data points. Typically, financial fraud analysts may not be trying to understand the overall patterns of account activity, but have to focus on untangling the web of transactions around a particular account that has been flagged as suspect. Programmers do not have to understand all dependencies in each of their projects, but may simply wish to know what the impact is of a proposed change on a single module.

Struggling with the limitations of node-link diagrams and demands of analysts, there have been several attempts to deviate from the "Overview first, zoom and filter, details on demand" visualization strategy as coined by [SHN96] when it comes to navigating graphs. Proposed research [ECH97, HP09, LPP06] and some commercial products have long proposed non-overview browsing strategies. For instance, the "Search, Show Context, Expand-on-Demand" strategy [HP09] involves users picking a particular data point as their focus and then having the system deliver an optimally relevant context based upon a custom user-interest function. Users can then navigate the visualization by expanding context in their desired directions.

However, the main drawback of such approaches is that users do not have a complete overview. Analysts can only see a partial view of the graph surrounding a single data point at any given time. Furthermore, the layout of the same structure might be rendered differently depending on the initial data point chosen by the user. This weakens one's ability to become familiar with the global position of data points and to find other data points from one's current position.

This paper makes two novel contributions: Firstly, we advocate the tight integration of two different search paradigms in a single, coherent partial view network visualization. Previous partial view graph visualizations have mostly relied on the *browsing* paradigm, where users are expected to guide themselves through the structure, based on the currently visible subset of the structure. Instead, we propose an integrated navigational model consisting of alternating cycles of textual querying followed by contextual browsing: Users view a part of the network, can then run a textual query over the entire nodeset and can subsequently see how the resultset relates to the currently visible part. Although overview based graph visualizations typically allow textual querying, doing this for partial view graph visualizations is much harder, since they, by definition, lack an overview of the entire graph. We are as yet unaware of any work that attempts to integrate the results from textual querying in partial view network visualizations.

Secondly, we designed a visual encoding that takes into account both size of the query results and their structural relationship to the currently visible graph. In our proposed encoding, which we call *graphcues*, we highlight direction, distance and potential relevance in each visual cue. These

cues can then be used by the user to decide on a direction for exploration beyond the partial view.

In the next section we briefly discuss research in information finding. In Section 3 we describe the concepts behind graphcues, detailing their computation and visual representation. Section 4 shows how graphcues can be applied in practice, using a large social network dataset as an example. In Section 5, we give a short overview of other related work in the area. Finally, Section 6 discusses limitations and suggests further work, and we conclude in Section 7.

2 Information finding in Graphs

Users typically adopt either a *querying* or *browsing* paradigm to locate information they are interested in. The first, searching, typically involves the user specifying a number of keywords that match the information he or she is interested in. These keywords are then used by a search engine to produce one or more potential matches, which the user can subsequently inspect in more detail. When browsing, on the other hand, a user inspects an item because it is related to the item she is currently viewing. Information browsing typically involves navigating from one information item to the next, for example by clicking a hyperlink. Each paradigm has its own distinct advantages: *querying* is convenient because it is fast, specific and predictable, provided good keywords are available. Browsing is slow but useful when exact keywords are lacking or hard to formulate, or when the context of an information item is important. In practice, potential solutions to complex searches often consist of multiple sub goals, each of which may require a different search strategy [MS88].

2.1 Querying

Querying in graphs usually takes the form of direct queries on node attributes, or structural queries using a graph query language like SPARQL. With both of these types of queries we can find (sets of) nodes with specific properties, and gauge the distribution of other properties over this set.
When visualizing the resultset and the connections between them, we can understand how the different nodes in this resultset interrelate.
Querying is undoubtedly powerful and fast, but we often miss the opportunity to find nodes or connections that we weren't looking for initially. If one particular node is of interest to the user, chances are that neighboring nodes warrant some attention, even if they did not exactly match the filters set up by the query.

2.2 Browsing

In contrast, browsing does allow us to hop from one interesting item to another potentially interesting item. However, the size of modern datasets makes browsing challenging because it is often impossible to render every item explicitly beforehand. The question then becomes how to guide users to items in that dataset that might be interesting, even if they are not currently in the active view. In [FUR97], Furnas describes two concepts that underlie navigation in abstract data structures: effective view traversability and effective view navigability. Effective view *traversability* means that for each data item in the structure the number of neighboring items must be low and the number of steps between any two information items must be small (both compared to the total number of items). Partial graph views generally meet the second requirement, as the diameter of the graphs under consideration is small compared to their size. The first requirement can be enforced by using a degree of interest function to limit the number of connecting items where needed [HP09].

Effective view *navigability* means that the user must be able to reliably find short paths between data items. Furnas formulates this in terms of outlink info, the information that tells a user which items are reachable from a specific data item. For effective navigability, outlink info must be complete and consistent, but to be usable it must be small. For graphs both of these constraints contradict, making navigability problematic in practice.

Alternatively, outlink info can also be formulated as residue [FUR97] or information scent [PIR97]. Information scent is a concept that tells the user how useful it might be to follow a particular link to an information item. Information scent has been used mostly in the context of world wide web navigation, and scent based models have been used to analyze and predict web site browsing patterns [CPP00].

Instead of attempting to store general outlink info for every node in a potentially large graph, our proposal is to make the displayed outlink info *contextual* to the search task the user is currently facing. The search task of the user can be specified using the query paradigm. One the one hand this reduces the outlink info that needs to be displayed for a single node to a manageable size, but on the other hand requires us to compute outlink info on the fly for a specific query.

3 Graphcues

We render this compacted outlink info with a glyph-like representation called a *graphcue*. A graphcue is in many ways analogous to a physical signpost on a crossroad, indicating the direction and, optionally, distance to a particular destination of interest. Signposts on road networks typically indicate direction to a known geographic location or landmark of importance, allowing users to triangulate their approximate position. If a user knows what the relative position of their desired destination is with respect to such landmarks, they may use these directions to navigate.

However, applying landmark-based navigation to abstract graphs requires that users already have an idea of what nodes might be considered landmarks and, more importantly, how they connect. In practice, this last point is a significant cognitive challenge as we cannot compute a static layout for these massive graphs.

Instead of landmarks, we propose a solution that bases these signposts on users' current information needs. In the real world, many signposts can be considered contextual: signposts to public restrooms are usually found in busy pedestrian locations with much foot traffic and signposts indicating currency exchange shops are typically found near national borders or important tourist sites. In a sense, such signposts are dependent on the (expected) navigational tasks at those particular locations. Instead of trying to provide a user with a global mental map of the entire structure, we postulate it is sufficient to simply guide them towards information items that are interesting given the context of their current task.

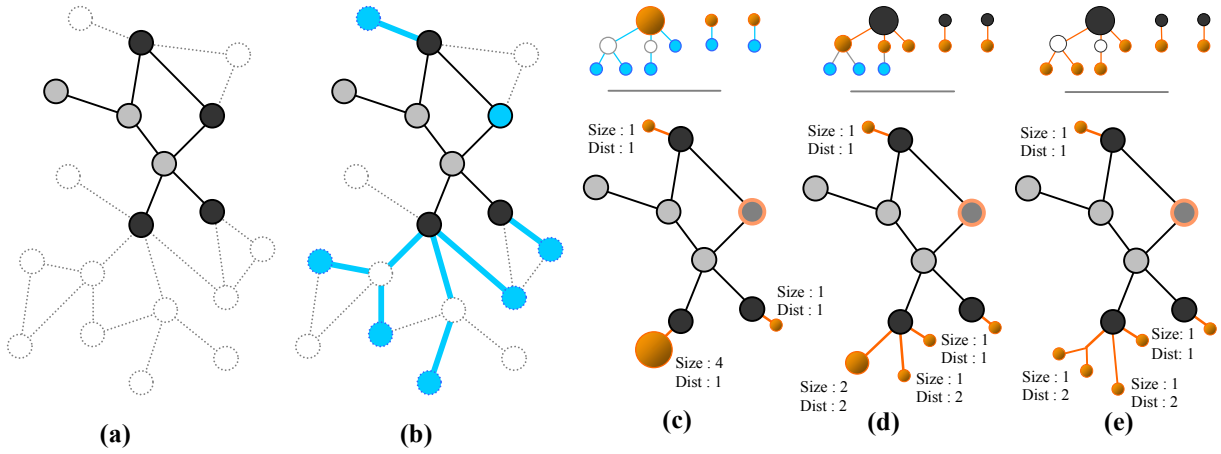

Figure 1: *Graphcue computation.* ***(a)*** *a partial view (grey) of a larger graph. The borderset is highlighted in dark grey* ***(b)*** *Search results (blue) and their shortest paths to nodes in the partial view.* ***(c)*** *Graphcues (orange) with shortest path search trees (top) aggregated at depth 0.* ***(d)*** *Graphcues with shortest path search trees aggregated at depth 1.* ***(e)*** *GraphCues with shortest path search trees aggregated at depth 2.*

In the following discussion we will assume that an information finding task can be specified with one or more keywords. In practice, this search-by-query method of information finding is the most prevalent method of finding digital information, so users are typically very familiar with it.

3.1 Graphcue computation

Given a large graph $G=(V,E)$ we define a *partial view* of that graph as a visualization of a subgraph $G'=(V', E')$ of G. In a typical setup, a data server is responsible for storing the entire graph G, while a visualization client connecting to this server only needs to be concerned with the local partial view G'. Partial views can for example be constructed by extracting a subgraph around an initial point of interest [HP09] or, alternatively, by specifying a search term and extracting the graph induced by all nodes that match that search term. A partial view can be expanded if a node G' has a neighbour that is in V but not yet in V'. The *border-set* $B \subseteq V'$ of G' are those nodes in V' that have one or more of such neighbours (see Figure 1a). Every search query Q that we execute on the full graph G returns a *result set* $R \subseteq$ V. How exactly this result set is obtained is beyond the scope of this paper, but practical implementations could include a simple search against a full text index or more complex graph mining algorithms.

We can obtain the set of graphcues that we need to display for a query by running a server-side multiple-source shortest path search from all nodes in border set B to all nodes in result set R (see Figure 1b). In the case of unweighted graphs, this can be implemented by a simple multiple-source Breadth First Search with worst-case complexity $O(|E|+|N|)$. In the case of weighted graphs, a multi-source adaptation [EKP96] of a Dijkstra shortest path search can be used with $O(|E|+k|N|)$ complexity, with k a small constant. For each node processed by the shortest path algorithm, we also keep track of the full path to its nearest source node. This administration can easily be updated each time an edge is traversed in the Breadth First Search, at no extra complexity cost. If two shortest paths are equal in length, we arbitrarily break ties and only keep one. Note that this means the total number of paths found will be equal to the number of nodes in R. The shortest path algorithm can be terminated if all nodes in R have been assigned their definitive shortest paths.

The result of the shortest path algorithm is a forest of search trees containing all the shortest paths from source set B to sink set $R - V'$. This forest is transferred to the visualization client for processing. Note that every node in this tree has precisely one associated nearest source in B, which we will designate as that node's *root*. Figure 1b shows a concrete example with the three separate search trees displayed above Figure 1c, one for each root.

A single *graphcue* on a node $b \in B$ for a result set R then consists of a tuple *(size,distance)*. *Size* represents the number of items in R that can be reached fastest via node b, while *distance* represents the minimum distance from b to any node in R. Concretely, a graphcue represents the number and distance of search results one can reach by following a particular edge or sequence of edges. Conceptually, a graphcue can be seen as a contextualized version of Furnas' proposed [FUR97] outlink-info that depends on the user's current information need.

3.2 Graphcue abstraction

Since we do not want to display the full path information from every node in B to every node in $R - V'$, we elect to only show detailed information on the first links in every path. We abstract the search tree for a single node such that it only displays aggregated high level information.

For every node in the tree we can define an aggregated leaf size (i.e. the total number of nodes in R one is able to reach by following shortest paths through this node) and distance (i.e. the minimum number of steps it will take to reach a node in R through this node). Both of these metrics can easily be defined recursively: The size of a leafnode is one, and the size of an internal node is equal to the sum of sizes of their children. The minimum distance of a leafnode is zero and the minimum distance of an internal node is the minimum of the distances of their children, plus one. Note that distances are in effect measured from the tree's root, not from the current internal node under consideration.

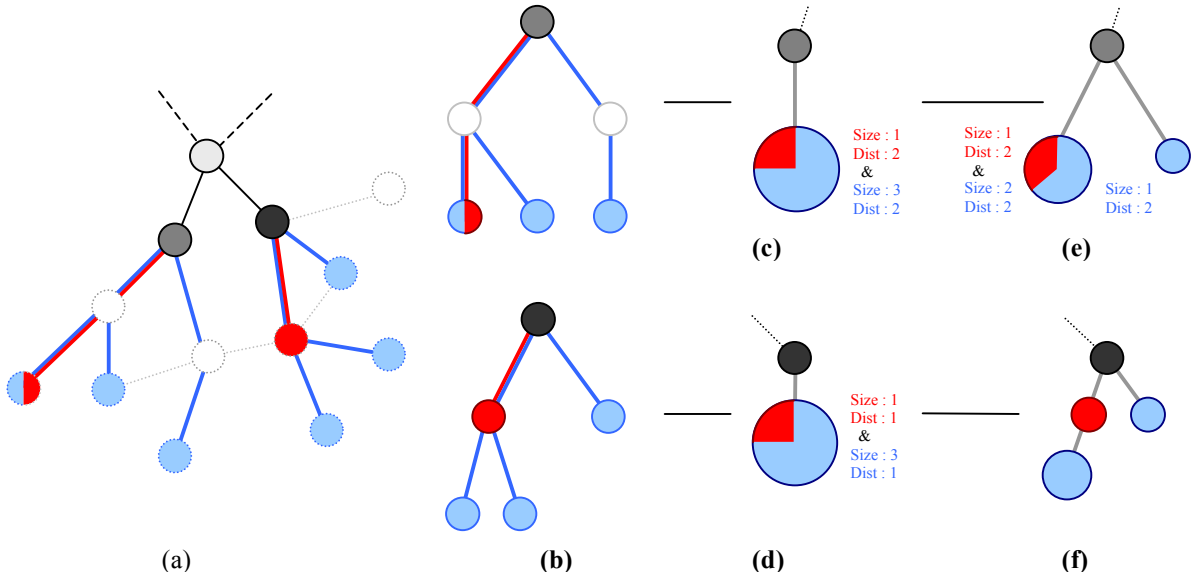

Figure 2: *Visual representation of multi query graphcues.* ***(a)*** *A partial view of a network (grey), showing the hypothetical results from executing two queries (blue and red) on a set of nodes and their shortest paths to the current view. Note that one node was included in both search results.* ***(b)*** *The two merged search trees corresponding to these shortest paths.* ***(c)*** *Top tree 0-graphcue : we can reach 4 matching search results, 3 blue and 1 red, all at distance 2. Note that we effectively counted the node matching both searches twice* ***(d)*** *Bottom tree 0-graphcue: We can reach 4 matching search results (3 blue and 1 red) in a minimum of 1 step.* ***(e)*** *Top tree 1-graphcue : by following the left branch of the tree we can reach 3 results (2 blue and 1 red) after 2 steps, by following the right path we can reach 1 blue node after 2 steps.* ***(f)*** *Bottom tree 1-graphcue: The right branch of the tree contains two aggregated tuples with depths 1 and 2. This results in a single link with two separate glyphs instead of a single combined pie glyph.*

We can then compute the graphcues for a node *b* in *B*, by examining the nodes in the search tree rooted at *b* at a particular depth *d*. This depth governs number and level of abstraction of the cues displayed. Each node *n* in the search tree at depth *d* then is mapped to exactly one graphcue, with aggregated size and depth. We define the *prefix* of that cue as the path in the searchtree from *b* to *n*. By definition, the length of this prefix will be equal to the chosen depth *d* and every cue will have a unique, but not necessarily non-overlapping, prefix.

By varying the aggregation level of the search tree, we control the length of the prefix of the graphcue. To reuse the signpost analogy, we can also think of this length as the *lookahead* of a directional cue. Simple directional signs typically have a lookahead of 1, indicating a single direction (e.g. 'restroom is to the right'). Alternatively, more complex signs or verbal directions typically have a larger lookahead ('restroom is to the right, and then the first door on the left'). Note that we can effectively chain together multiple signs with lookahead of 1 to obtain the same effect as a single sign with a larger lookahead, but the user does not know what the next direction will be until they have followed the direction indicated by the current sign in the chain.

For graphcues we define lookahead in a similar manner: a graphcue's lookahead is the length of the cue's prefix. Since the prefix of a cue is the path from the corresponding node in the searchtree, the lookahead is equal to the depth at which we have aggregated the search tree. We introduce *n*-graphcue as shorthand for a graphcue with lookahead *n* (*n*>0). 0-graphcues (i.e. no lookahead) simply state which items are reachable at which distance but do no provide explicit directions to those items. In practice, we found that 1-graphcues (Figure 1d) provide a good trade-off between visual simplicity and information density. Graphcues with lookahead 2 or higher require us to render part of the searchtrees for each node with a cue (see Figure 1e) and generally introduce too much visual complexity into the scene.

3.3 Graphcue visual design

In the previous paragraph we described how to compute a set of n-graphcues for a single query. For a particular query, a single graphcue attached to a node *x* is a tuple (*size*, *depth*) that represents the number and distance of search results one can reach fastest through *x*. To keep consistent with our node-link representation of the graph itself, a graphcue for a single query can then be represented by a basic node-link style glyph. The node portion of the glyph represents the entire subset of the search results that can be reached through a particular sequence of links (the prefix), which itself is represented by the link. This allows us to map the properties of a graphcue to its visual representation in an intuitive manner. The size of the reachable result set is mapped to the size of the node, while the minimum distance to the reachable subset is mapped to the length of the link (see Figure 1c and 1d).

Now suppose that a user has requested graphcues for multiple queries. Each of these queries has its own result set *R*, which are not necessarily disjoint (Figure 2a). One option might be to process the search trees for each query separately, using color to indicate which cue belongs to which query. We can then simply attach all cues for each query to their respective roots. This poses a problem however, as we are no longer accurately representing directional information. It might be very relevant to the user that, by following a single prefix, they can reach nodes that match multiple queries. If we render separate cues for each query we give the incorrect impression that all prefixes are

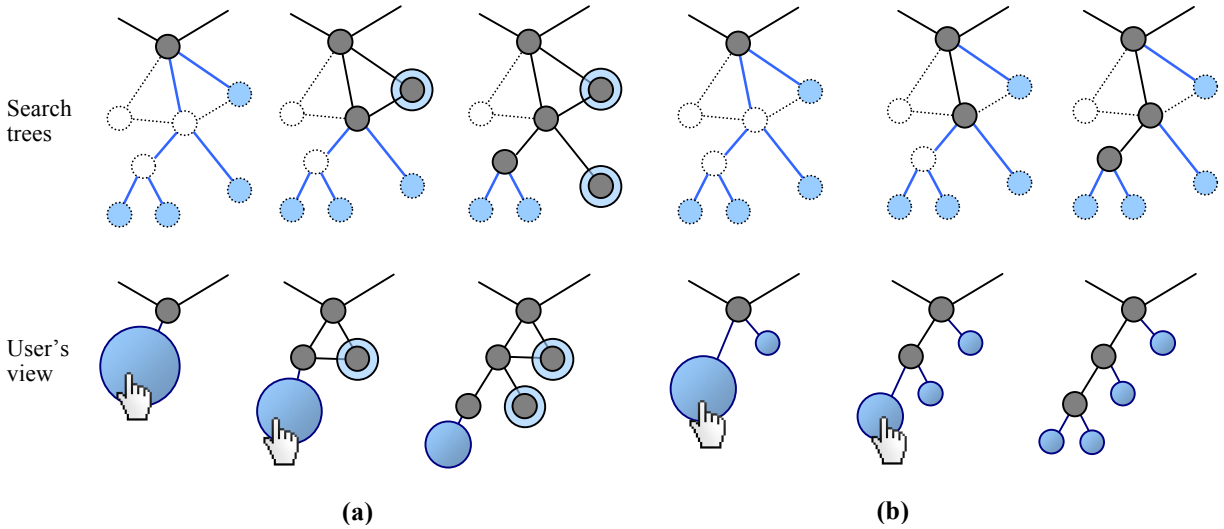

Figure 3: *Using graphcues to direct 2 steps of iterative expansion in the case of **(a)** 0-graphcues and **(b)** 1-graphcues. Top row shows the currently visible nodes and search paths to hidden search results (indicated in blue). Nodes in the partial view that directly match a search result are circled. Bottom row shows the user's view with the current partial graph and associated graphcues. Click actions are indicated by a hand icon. Note that clicking a 0-graphcue in **(a)** will bring in all search paths incident to the cue's root, possible resulting in multiple nodes being added to the graph. Clicking a 1-graphcue in **(b)** will only bring in those paths that lead to the indicated subset of search results. The prefix of each of those paths will be an incident edge to the root and the partial view will be expanded with at most one node for each click.*

mutually exclusive, and that there is not a single path from which we can reach nodes in both result sets.

Instead of processing each query separately, we compute the complete search forests for each query and then merge them (Figure 2b). Aggregation can be done in a similar manner as explained in the previous paragraph, with the exception that we keep track of the aggregated values per query. This results in each internal node now being assigned a set of tuples {(size, distance)}, one for each query executed, instead of a single tuple.

The visual representation of this set of tuples can then be done by a combined node-link glyph. Each query is assigned a separate color and we encode the size and depth variables in a similar manner as for a single query. More precisely, assume that we have executed n queries and for a single cue we need to represent a set of tuples $\{(s_0, d_0), (s_1, d_1) \ldots (s_n,d_n)\}$. Note again that there is a difference between this set of tuples and the collection of singleton tuples $\{(s_0,d_0)\}, \{(s_1,d_1)\}, \ldots, \{(s_n, d_n)\}$. The latter represents a set of n distinct cues, each corresponding to a unique prefix and represented by a single graphcue. The former represents a set of n reachable subsets on paths having a common prefix.

Like the single query case, we render this common prefix as a line. If all of the depth values are unique we can render the result sets as a set of n spheres on that line, where the distance between the sphere and the line's origin represents the minimum distance to that particular result set (Figure 2f). However, in most cases there will be tuples that have equal distance to the root. In this case we opt to represent all those tuples as a single sphere whose radius is proportional to the sum of the tuples' sizes. A piechart subdivision then indicates the relative contribution of each query to the total size of the result set at that distance.

Figure 2 illustrates these mappings. We ran two different queries Q_1 and Q_2 on the graph partially displayed in figure 2a. Nodes matching Q_1 are indicated in red, nodes matching Q_2 are blue. Note that one node matched both queries. Figure 2b shows the two shortest path search trees originating from the dark grey nodes in the border set. In figure 2c we computed the 0-graphcue for the top search tree. For the root node in the tree we obtain tuples Q_1:(size:1, dist:2) and Q_2:(size:3,dist:2). Since both of these tuples have equal minimum distance, we render them as a single circle glyph, with a piechart indicating the relative sizes. A similar computation can be done for the graphcue in figure 2d. Figure 2e shows the 1-graphcues for the top searchtree. For each node in the first level of the tree we compute the aggregated values. For the left branch this yields Q_1:(size:1, dist:2) and Q_2:(size:2, dist:2), resulting in a single pie glyph of size 3 as distances are identical. The right branch is represented by a single graph clue Q_2:(size:1, dist:2). Figure 2f shows a case where distances in the set differ. The right branch of the bottom searchtree yields a simple 1-graphcue of Q_2:(size:1, dist:1). The node on the left branch has associated tuples Q_1:(size:1, dist:1) and Q_2 (size:2, dist:2). Since these two tuples have the same prefix but different distances, we render them as two separate colored node glyphs of sizes 1 and 2, aligned on a single line.

3.4 Graphcue layout

The positions of graphcues are calculated by using the same layout algorithm as used for the nodes, but with modifications to the procedure. A new dummy node and edge are added to the graph for each graphcue. However, in order to preserve the user's current mental model of the graph, all non-graphcue nodes' positions are fixed. Within these constraints, the dummy node positions are optimized. After a new set of positions has been computed, dummy nodes and edges are removed and graphcues are drawn in their place.

3.5 Graphcue interaction and expansion

Expansion is what allows us to browse a partial view and is similar to following a hyperlink in a webpage. Expanding a node b in the border set of a partial view consists of adding

a number of *b*'s direct neighbours to the partial view. In this section we show that we can also use graphcues as triggers for expansion of the current partial view.

By definition, the set of the first links of the prefixes of all cues attached to a node form a subset of that node's incident edges. Clicking a cue attached to node *x* then amounts to the user indicating that he or she is interested in following paths that lead to the result set represented by this graphcue. By 'peeling off' and explicitly rendering all of the first edges in this set of paths (along with their end nodes) we have created a bigger partial view for which we now need to display new graphcues.

Conveniently enough, we do not have to recompute the search trees every time a user chooses to expand the current partial view by clicking a graphcue. Since we are expanding our graph along the edges of a search tree, our new search trees will be subtrees of the original one. New graphcues can then be recomputed at the client side by reusing part of the original search tree.

Figure 3 illustrates a concrete example of this, using both 0-graphcues and 1-graphcues. The top row shows the relationship of the partial view to the full graph, the result set from running a search and the shortest paths to the nodes in the result set. The bottom row shows the user's perspective of the partial view, where all paths have been abstracted with graphcues. Clicking the 0-graphcue in Figure 3a will add two edges to the partial view, and the graphcues can be recomputed in a straightforward manner. In the case of 1-graphcues (Figure 3b), the prefix of the cue always matches a single incident edge. By clicking a 1-graphcue the user indicates that he or she wants to expand all search paths that start with this specific incident edge.

Alternatively, we can use graphcues to improve on incremental navigation. As a single graphcue represents a set of searchresults along a unique path from the current node, we can also jump directly to the first node in that set, and skip expansion of every single node along the path. In this manner, users can navigate the graph by iteratively specifying textual queries and then following up by a complete expansion to the nearest result set. In out prototype we have implemented this feature under a contextual menu for a single graphcue.

4 Application: Enterprise Social Network Discovery

To evaluate the potential benefits of our approach we applied it to a massive social network of a large multinational software company. Recent work suggests that enterprise social networks can be utilized for a variety of tasks, such as expertise location [ES08], finding sociable individuals to collaborate with [CTL*09], and suggesting paths to request introductions to persons of interest [LEG*08]. However, such tasks do not require understanding the global pattern of connections across the company but instead understanding a partial set of connections relevant to the user. We wanted to see if augmenting such views with graphcues would assist in basic navigational tasks.

The following case study is conducted on an enterprise social network consists of approximately 500,000 people and 30 million relationships. Individuals are connected based on organizational ties (i.e. sharing the same manager), co-authorship ties (i.e. writing a paper together) and friendship ties (i.e. friending on an online social network site). The relationships are weighted based upon the aggregation of all these ties and stronger relationships are rendered as thicker links. All data is stored centrally on a data server which listens to client request for data.

The visualization client runs in a standard web browser using Adobe's Flash framework. A force-directed algorithm using a stress majorization [GKN04] optimization method computes the node layouts. Instead of showing the full graph, we show partial views based on the user's currently expressed interest. Users express interest by entering an initial multi-keyword query and the server returns the top *n* employees (by default, *n*=25) matching the topic and all of the relationships connecting them. Note that in order to protect user privacy, names and images have been anonymized with fake data, but the connections are based on actual data.

When users want to navigate beyond this graph, they have two options. They can browse without guidance by clicking a node, bringing the node's neighbours into the view. This naïve interaction model is similar to existing "expand-on-demand" systems [TOU10, BRA10]. However, bringing all neighbours into the view may increase the partial view by hundreds of nodes resulting in a visually incomprehensible display. An advanced approach utilizes interest functions to bring in only the most important nodes relevant to the user task [HP08]. In our system, the top 3 neighbours will be brought into view using an interest function based on the neighbour's relevance to the user's initial search query from which the graph was originally extracted. While such browsing may result in interesting discoveries, it is difficult for users to reach interesting portions of the graph far away from their current location.

We therefore augmented the system by providing 1-Graphcues whenever users specify additional keywords describing nodes that interest them. In our system, content associated with each node is used to find proper matches (e.g. searching for 'visualization' will return people who feature the term visualization in their papers, patents, blog posts, bookmarks, etc.). The system will find the top *r* nodes (by default, *r*=10) that match the user's new search. The data server then calculates the shortest path to each of these nodes, as described in Section 3.1. In our implementation, a full shortest path search took a few seconds. Finally the client attaches a graphcue to the border nodes on the shortest path. For any nodes that match those already present in the partial view, the nodes are surrounded by a halo that matches the color of the graphcues. In the next sections we illustrate different usage scenarios of how graphcues aid in the navigation of graphs.

4.1 Location: Show me how a node connects.

In the first scenario, we demonstrate a very simple task using graphcues: navigating to a specific node of interest, which might be hidden. The task is non-trivial in partial view navigation because users must remember the precise sequence of steps if they wish to return to a node of interest.

The partial view shown is a social network of the top 10 visualization experts across the enterprise. A manager, Fred Jameson, is not in this original network of experts, but is interested in engaging one of these visualization experts into his project. However, he is uncomfortable contacting these experts directly out of the blue. Instead, he wishes to find any contacts he may have in common with this group of experts. By running a simple textual search, he can quickly find the shortest path between him and the experts. As Figure 4 shows, one of the persons closest to him is Dan Misawa, whose node is attached to the graphcue. By expanding outward from Dan Misawa, he can see the

common connections and then leverage his existing social network to become introduced to Dan.

4.2 Discovery: Show me how to reach a set of nodes.

Navigation from the current partial view to a specific node, as outlined in the previous paragraph, is a specific instance of a more general use case. In practice, a user cannot always uniquely identify a node he wants to navigate to, but may have an idea of the general characteristic of the set of nodes he wants to reach.

Concretely, imagine the same manager once again exploring the partial view of the top visualization experts. However, he is also interested in bringing data mining experts into his project. Ideally, he would like to find visualization and data mining experts that already share existing ties and thus may have a better chance of working well together. By specifying an additional contextual query (i.e. 'data mining'), his current view of the visualization experts graph is augmented with shortest paths leading to the results of the 'data mining' query.

From the image, he deduces that there are three paths to the 'data mining' subset – the orange graphcues via Amar and Betty (Figure 5). He decides that Amar would be a good person to contact since he has worked with him before. Figure 5 also shows the expansion of one of the graphcues emanating from Amar, leading to a data mining expert three steps out.

4.3 Intersection: Show me how to reach the intersection of multiple disjoint groups of nodes.

Because we can display the graphcues for two different queries in the same view, we can use them to correlate different sets of search results. The information that a single direction in the current view leads to a set of people matching two separate queries is useful in practice. As an example, suppose our manager returns to the subgraph of top visualization experts. However, this time he is interested in searching for data mining experts as well as machine learning experts. Because data mining and machine learning are related fields, he is curious if he can find experts with strong ties to both communities in his organization.

Figure 6 shows the partial view with graphcues pointing to the manager himself (blue), 'data mining' (orange) and 'machine learning' (green). Again, Betty is on the shortest path to the highest number of data mining and machine learning matches. In fact, one unique path leads to matches of both types. Figure 7 shows the graphcue expanded two steps, leading to Frank Adams – who has direct connections among both data mining and machine learning people. These attributes may suggest to the manager that this colleague would serve as a great source of information about experts in both fields.

Note that this type of useful contextual information cannot always be obtained by more precisely specifying the initial search query. A user might be interested in the relations between people that match keyword 'machine learning' and people that match keyword 'visualization'. Simply searching for these people with a single query 'machine learning AND visualization' might yield an empty result set, when there is no individual who matches both keywords. On the other hand, searching for 'machine learning OR visualization' will results in a single large set where the overlap is unclear. However, a cluster of people that each have one of the desired specialties but work closely together might exist. Whereas textual queries only examine the

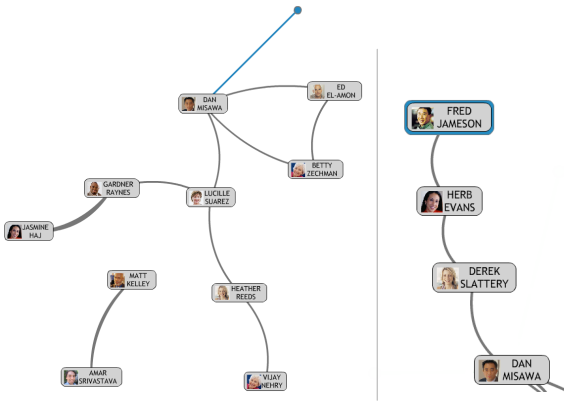

Figure 4: *Using a graphcue to locate a single node that is outside the current partial view. Left image shows a closeup of the graph after one-click expansion of the cue.*

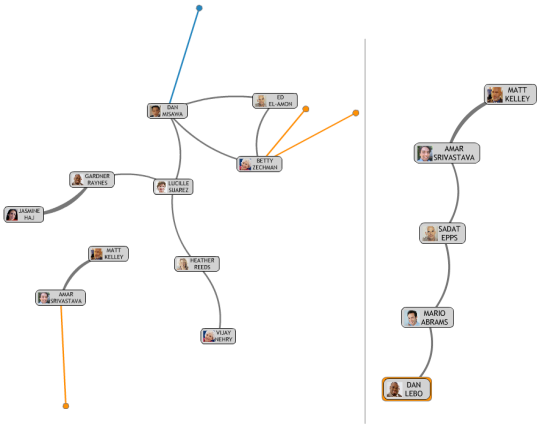

Figure 5: *Displaying cues to a result set in a partial view. The orange result set has total size three. The left image shows the result of expanding the bottom most graphcue.*

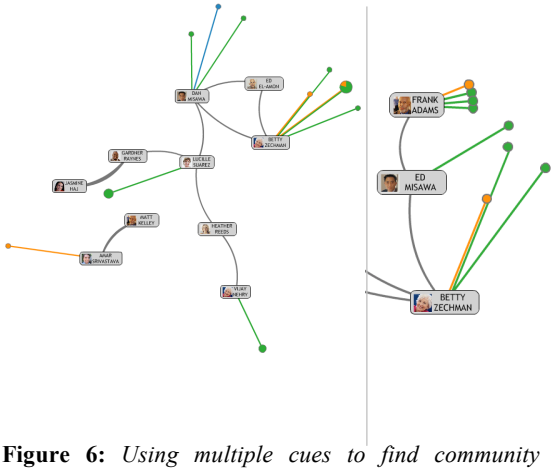

Figure 6: *Using multiple cues to find community overlap. By following the piechart graphcue matching two queries, we find an expert (Frank Adams) with connections to both.*

associated information for each node, graphcues also take the structural properties of the graph into account.

5 Related Work

As mentioned in the introduction, one potential solution for visualizing large graphs is to not aim for a global overview of the entire graph, but rather to show a number of integrated local views around a point of interest. In the context of graph drawing this idea has been used many times before [ECH97] and samples can be found online at [BRA10] or [THE10]. A smaller number of commercial technologies offer componentized partial view layouts, such as [PAL09] or [TOU10]. By adding interactive capabilities, users can decide which point in their current partial view they are interested in exploring next – and then either bring in more data around it or re-center their view of the graph on that point. Although partial views are very useful for obtaining local views of graphs, they have two main weaknesses. Firstly, they do not deal well with graphs in which the median number of neighbours per node is relatively high (more than 10). Repeatedly expanding nodes would quickly produce an unreadable partial graph.

Secondly, because partial graphs only provide localized views, it is difficult for users to orient themselves within the graph. One option is to develop targeted layout algorithms that enforce consistency between the layout of the local view and the global topology of the entire network [DMS*08]. Other interaction research has dealt with navigational aids that help in efficiently navigating from one node to a possibly invisible neighbour. Bring & Go or the Bring Neighbors Lens [TAS*06, MCH*09] deal with cases where a neighboring node is invisible because the entire graph is too large to fit the viewport. TugGraph [AMA09] attacks the same problem for hierarchical graphs, where a node may be invisible because it is contained in an abstracted cluster. Although all above mentioned techniques certainly improve the efficiency of navigating from one node to another, neither of them touches upon the core problem of deciding **which** node to navigate to next in a partial context. In this paper we are addressing precisely this shortcoming by adding cues that show users directions to nodes that match their current information needs, even if these nodes are distant.

In terms of information representation, our proposal is similar to the Spacetree [GPB02] but generalized to graphs. Spacetrees also use the concept of visual abstractions of subtrees which show breadth and depth of the tree, combined with a textual search that highlights matches in the data. Instead of using the tree datastructure directly we compute BFS search trees based on a textual query and integrate those in the current partial view of our graph.

Probably most relevant to the ideas outlined here are the concepts behind ScentTrails [OC03]. ScentTrails combine the advantages of query based search and web browsing by augmenting a traditional web interface with information scent based cues. ScentTrails are annotations on links to pages that more closely match the user's specified interest. They guide the user towards pages that might contain the information he or she is looking for, but at the same time allow him or her to maintain contextual awareness. Annotation is done by changing the visual appearance of links, for example by changing the font size or changing background color. In this paper, we generalized the concepts of ScentTrails to arbitrary partial view networks. That is, we proposed a combination of query based search and browsing to guide users through a visualization of an arbitrary graph, visually indicating which paths lead to results matching a textual query. There are some notable differences with the ScentTrails algorithm, however. Firstly, we wish to better separate out the information contained in the textual query. To that end, we divide the resulting hits for each query part and code both distance to the current view and total relevance separately in our visualization. ScentTrails encodes total relevance for the entire query in a single variable (scent) that represents an aggregate value of both relevance and distance. Secondly, where ScentTrails uses a pre-computed scent conduit matrix using an expensive $O(n^3)$ iterative method, we compute our visual indicators online by using a simple shortest path search and aggregation. Finally, ScentTrails can only show the most relevant paths emanating from a single currently visible webpage, while we show how paths relate to a larger subset of the entire graph.

6 Discussion & Future Work

In the scenarios above, we have demonstrated how graphcues could be used as a navigational aid when browsing partial views. Without graphcues, these simple tasks become almost impossible to accomplish as the user lacks both a mental map of the global structure and navigational aids. However, we acknowledge that there are still outstanding challenges in improving the navigational experience.

In our implementation, we rely on the user to search for nodes of interest. There are situations when users may not be able to express their interest with keyword searches. Similarly, there are graphs where node information is not rich and thus the capabilities to search are limited.

Graphcues may also be implemented as landmarks in addition to guides. For instance, if users came across an interesting node and wished to annotate it as a landmark, they could continue to freely browse a graph while maintaining a reminder of where they came from.

For certain scenarios, our methods for calculating graphcues may not be optimal. For instance, in our current design, only one graphcue is shown for each path, even if there are multiple shortest paths to a node in the result set. This drawback is particularly obvious in star-like graphs, where many paths lead to the same central node. In our current implementation, only one node will show a shortest path to that central node, implying that this is the closest node. In effect, other visible nodes might have equal distance and this fact is currently obscured. A potential solution only requires a simple modification to the path algorithm, as we can instruct it to store all shortest paths, not just a single one. This would then yield a set of search DAGs instead of search trees, which requires occasional duplication of a node on the search path over two DAGs. Actual graphcue computation and visualization can still be done in a similar manner. While the resulting display is a more accurate rendition of all shortest paths to a node of interest, in practice we found that the potentially large number of cues proved a distraction. Another drawback of this solution is that, whenever the current partial view changes (e.g. when a user expands a node), we need to rerun the breadth first search algorithm to account for new paths that might have appeared. As an example, after clicking the first graphcue in Figure 4b, the newly added node should also display a cue to the existing node at depth 1, since both shortest paths to that node have length 1. A full update of the search trees requires an additional $O(|E|+|N|)$

computation on the server side after each node expansion, which proved prohibitive in our case.

Although we have provided a number of sample use cases on a realistic and large dataset, formal and longitudinal evaluation of the effectiveness of integrated browsing and searching would constitute a paper by itself. One option would be to design a synthetic task that requires both browsing and searching paradigms to solve and compare task execution times with and without cues. The same setup could be used to evaluate our current graphcue design to arrive at alternates.

7 Conclusion

As graphs become larger, partial views become an attractive solution in managing visual complexity. In this paper, we attempt to relieve some of the cognitive burden on users by integrating two information finding paradigms: querying and browsing. We also formalize the concept of a graphcue: a navigational aid to help users browse graphs towards nodes of interest. We have integrated querying and browsing into a system designed to support exploration of an enterprise social network of over 500,000 nodes and 30 million edges. By demonstrating our ideas in a number of different user scenarios, we show how an integration of querying and browsing allow tasks to be completed that would have otherwise been nearly impossible, suggesting our approach is effective for navigation in partial graph visualizations.

Acknowledgments

We thank Erel Uziel for his assistance in utilizing the enterprise social network data for our application.

References

[AMA09] D. ARCHAMBAULT, T. MUNZNER AND D. AUBER. TugGraph: Path-Preserving Hierarchies for Browsing Proximity and Paths in Graphs. In *Proceedings IEEE Pacific Visualization*, pp. 113—121, 2009.

[BRA10] THE BRAIN. http://www.thebrain.com. Accessed December 2010.

[CTL*09] D. CHEN, J. TANG, J. LI, AND L. ZHOU. Discovering the staring people from social networks. In Proceedings of WWW '09 pp. 1219-1220, 2009.

[CPP00] E. CHI, P. PIROLLI AND J. PITKOW. The scent of a site: A system for analyzing and predicting information scent, usage and usability of a Web site. In *Proceedings CHI 2000*. ACM Press, New York, pp. 161-168. 2000.

[DMS*08] T. DWYER ET AL. Exploration of Networks using overview+detail with Constraint-based cooperative layout. *IEEE Transactions on Visualization and Computer Graphics*, vol. 14 (6), pp. 1293-1300, 2008.

[ECH97] P. EADES, F. COHN AND M.L. HUANG. Online animated graph drawing for web navigation, *Springer Lecture Notes in Computer Science*, Volume 1353, 1997

[EKP96] P.W. EKLUND ET AL. A Dynamic Multi-source Dijkstra's Algorithm for Vehicle Routing. *Australian and New Zealand Conference on Intelligent Information Systems (ANZIIS '96)*, pp. 329-333, IEEE press, 1996.

[ES08] K. EHRLICH AND N. SHAMI, Searching for expertise. In *Proceedings of the SIGCHI Conference on Human factors in Computing Systems*. ACM Press, New York, pp. 1093-1096, 2008.

[FUR97] G.W. FURNAS. Effective View Navigation. In *Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pp. 367-374, 1997.

[GKN04] E. GANSNER, Y. KOREN, AND S. NORTH, Graph Drawing by Stress Majorization, *Proc. 12th Int.Symp. Graph Drawing (GD'04),* pp. 239–250, 2004.

[GPB02] J. GROSJEAN, C. PLAISANT, AND B. BEDERSON, SpaceTree: Supporting Exploration in Large Node Link Tree, Design Evolution and Empirical Evaluation. In *Proc. of IEEE InfoVis 2002*, pp. 57 -64, 2002.

[HP09] 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*, vol. 15 (6), pp. 953-960, 2009.

[LPP06] B. LEE ET AL. TreePlus: Interactive Exploration of Networks with Enhanced Tree Layouts. *IEEE Transactions on Visualization and Computer Graphics*, vol. 12 (6), pp. 1414-1426, 2006.

[LEG*08] C. LIN, K. EHRLICH, V. GRIFFITHS-FISHER AND C. DESFORGES. SmallBlue: People mining for expertise search. In *IEEE MultiMedia* 15(1), pp. 78-84, 2008.

[MS88] G. MARCHIONINI AND B. SHNEIDERMAN. Finding Facts vs. Browsing Knowledge in Hypertext Systems. *IEEE Computer*. Vol. 21(1), pp. 70-80, 1988.

[MCH*09] T. MOSCOVICH, F. CHEVALIER, N. HENRY, E. PIETRIGA, AND J.D. FEKETE. Topology-aware navigation in large networks, In *Proceedings of CHI '09*, pp. 2319-2328, 2009.

[OC03] C. OLSTON AND E. CHI. ScentTrails: Integrating Browsing and Searching on the Web. *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 10(3), pp. 177 – 197, 2003.

[PAL09] PALANTIR TECHNOLOGIES. http://www.palantirtech com/. Accessed December 2010.

[PIR97] P. PIROLLI. Computational Models of Information Scent-following in a Very Large Browsable Text Collection. In *Proceedings of CHI '97,* pp. 3-10, 1997.

[SHN96] B. SHNEIDERMAN, The eyes have it: a task by data type taxonomy for information visualizations. In *Proceedings of the IEEE Symposium on Visual Languages*, pp. 336-343, 1996.

[TAS*06] C. TOMINSKI, ET AL. Fisheye Treeviews and Lenses for Graph Visualization. *International Conference on Information Visualisation (IV06)*, London, UK, 2006.

[TOU10] TOUCHGRAPH LLC. http://www.touchgraph.com. Accessed December 2010.

[THE10] VISUAL THESAURUS. http://www.visualthesaurus. com. Accessed December 2010.