Abstract

Most graph layout algorithms strive to present an uncluttered view of the graph that reflects the structural relationship between nodes and edges comprising the graph. Very few focus on providing a layout that integrates graph structure with attribute values associated with a node or edge. To fill this gap, we propose an interactive force-directed graph layout strategy that adaptively draws graphs taking into account the attributes of the nodes and edges. The main contribution of this paper is the flexible design of transfer functions based on node attributes to alter the effects of different forces on the graph layout. By editing the transfer functions according to some guidelines, we can get different layouts in a flexible manner. The flexibility of the transfer functions allows one to create clustering based on node attributes as well as other layouts that show influence from both graph structure and node attributes. As graphs get larger and more complex, the flexibility for exploring different relational properties of graph elements will allow us to understand them better. Our experimental results demonstrate the effectiveness of exploring different datasets using transfer functions.

Index Terms: E.1 [Data]: Data Structures—Graphs and Networks; H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval—Clustering; H.5.2 [Information Interfaces and Presentation]: User Interfaces—Graphical user interfaces (GUI)

1 Introduction

A graph can be used to model many real-world applications where node-link relations exist, such as biological protein networks, email networks, terrorist cells, complex social networks. The extremely large growth rate of the Internet, particularly social networks, has led to the ubiquity of network data as well as the evolving definition of the graph. The nodes and edges of graphs can contain more than one attribute, e.g. the people in a social media website usually have additional information, such as the different topics they are interested in and the music they listen to. These multiple attributes make the graph more complex and heterogeneous [5] since the layout of this kind of graph should reflect node attributes. For example, we would like the layout of the graph to reveal different clusters of people according to similarity of preference for topics or music in order to gain better insights on the different roles that they play in the community. To extract knowledge from such complex data, besides the node relationships represented via edges, the overall graph layout should also be related to the node attributes. So for large and complex graphs it is difficult to visualize the graph in one static layout from which the end users can get all the information they want to know. Instead, there are multiple informative graph layouts in terms of different interested node attributes.

While excellent methods exist for analyzing such complex, or heterogeneous, networks [19] [24] [34] [35], they are often specialized for a particular task [31]. In this paper, we propose a new visualization tool to aid in the understanding of these complex networks by tailoring the visualizations using different transfer functions. Similar to the idea of transfer functions in volume rendering for scientific visualization, we can design different transfer functions here to produce the desired graph layouts. Different transfer functions are designed to correspond to different attributes to influence the layouts, such that node attribute values and graph topology are displayed to the user in a meaningful way that relates the two. This paper introduces transfer functions for flexible graph layouts and presents a scalable visualization framework that support a multitude of layouts that illustrate different node relationships for different contexts.

The force-directed method is one of the most popular techniques to draw the structures of graphs due to their ease of implementation, ability to fine tune, and producing aesthetically pleasing results. In this paper we examine how the force-directed method is incorporated in our visualization framework to produce graph layouts using force transfer functions. Other methods aside from force-directed methods can also benefit from our proposed approach. Traditional force-directed techniques only consider graph structure and node placements. In this paper, we design force transfer functions that also relates node attributes to force parameters. Users can interactive-
ly modify force transfer functions to adjust force parameters based on attribute values. The work in [15] proposed a density-based approach to data clustering by combining attribute information and connectivity information. Their algorithm proposed to compute the density of single objects which takes attributes and graph information into account. The clusters are detected by merging all objects located in the same dense region. However, no actual graph layout results were shown in their paper. In this paper we propose a totally different approach using force transfer functions. We apply our method to real datasets to produce the graph layouts which are shown to users for visual investigation.

A common problem with force-directed layouts is when the number of nodes is large. For example, Eades’ [10] method produces a very cluttered display for large, dense graphs. The Fruchterman-Reingold technique [13] improves upon this, but it is still difficult to visualize graphs where the number of nodes is sufficiently large. With our method, we can produce meaningful layouts as the number of nodes gets large (e.g. 1000+ nodes) by tuning force transfer function. In summary, the contributions of this paper are:

1) Combine node attribute values and graph topology to determine graph layout.
2) Facilitate this process via transfer functions, and in this paper, illustrated using force-directed layout.
3) Demonstrate wide range of layouts on different datasets.

2 RELATED WORK

The work presented in this paper is related to both graph drawing and clustering. In the following parts of this section, we give the introduction of various techniques of graph drawing as well of clustering. However, readers should bear in mind that the clustering algorithms are not the main focus of this paper. Our work focuses on the use of transfer functions which provides the flexibility for users to alter the layout and discover different clusters, views, and aspects of the graph. For context and reference for interested readers, we present a brief overview of clustering at the end of this section.

2.1 Graph Drawing Algorithms

There are many different algorithms for drawing the structure of graphs ranging from physical simulations to spectral techniques utilizing matrices related to graphs.

Simulation methods utilize physical force-directed models to draw the graph which are intuitive to understand and generate graph layouts with nearly uniform edge lengths and few edge crossings. A downside to using such methods is their costly O(n2) running time which makes them not scale well to graphs with thousands of nodes or more. Frick et al. present several heuristics to improve the convergence of the force-directed model [12]. Harel et al. took force-directed algorithms to graphs with over 15,000 vertices and significantly improve the speed by a new multi-scale scheme for drawing graphs [16]. Quigley and Eades present an O(m+nlogn) method [32] for grouping nodes (n and m are numbers of nodes and edges respectively). They accomplish this by modifying the Barnes-Hut [4] algorithm. While this is a fast and efficient algorithm for producing a graph layout, it does not take into account node and edge attributes. Heer and Boyd’s Vizster application [22] uses a force-directed layout combined with a hierarchical, agglomerative clustering approach [28] to cluster nodes into communities. However, these tools primarily involve the use of color and/or opacity to highlight information within the graph. This is fundamentally different from our work since we wish to find a positioning of nodes that relates node attribute values to graph structure. Noack defined a new energy model such that a layout with minimum energy would yield clear graph clustering [30]. In their work, the term clustering refers to groups of nodes with high cohesion and low coupling (same as [32]). While this method produces a desirable clustering layout, it is not designed to incorporate node attribute values. Boden et al. utilizes the Gephi Platform to visualize the clustering results of attributed graphs [6]. This approach is different from our work which computes the clusters through visualization, while they computed the clustering results prior to visualization by using the graph mining algorithms GAMer [16] and DB-CSC [15].

To deal with large graphs, hierarchical methods have been proposed by laying out a coarser version of a graph, then progressively refining the coarse graph, until the entire original graph is laid out. These algorithms produce similar results with force-directed approaches but with much faster speed. The GRIP algorithm [14] produces multilevel coarsened hierarchies based on a maximal independent set filtration of vertices of the graph. The Fast Multipole Multilevel Method, or FM3, algorithm [17] produces hierarchical graphs by partitioning the graph into subgraphs called solar systems. These solar systems are contracted down to single nodes and the process is repeated to create a hierarchy. Abello et al. [2] developed a system for Ask.com to interactively explore large graphs. They achieved this by imposing a hierarchy on a general input graph. The algorithm proposed in our paper requires node-node comparisons to be made between every pair of nodes (and their attribute values). This makes hierarchical methods (based on graph structure) a less desirable choice as these comparisons would be lost in the coarsening and refinement stages.

Spectral methods use spectral graph theory [8] to efficiently produce a graph layout. Harel and Koren [20] proposed a High-Dimensional Embedding technique to project a node position from a high dimensional to two dimensional space. Koren et al. present an extremely fast layout algorithm for large graphs in [25]. They accomplish this by iteratively coarsening the graph until the number of the vertices of the coarsened graph is small enough to solve the generalized Eigen projection problem. However, as shown in the survey by Hachul and Junger [18], these algorithms can fail to produce a good layout in some cases, particularly when the graph is dense. While methods of this form can handle many more nodes than the technique presented in this paper, they do not address arbitrary node attribute values as a part of the graph.

2.2 Data Clustering

In unsupervised learning, the goal of clustering is to categorize the unlabelled data into a certain number of groups. There is also extensive work done in the area of clustering graphs [11] [33].

The K-means algorithm [21] is the most popular clustering algorithm used in scientific and industrial applications. It represents each of the clusters by its centroid, i.e. the mean of the points in the cluster. This algorithm is simple and easy to implement. Many extensions and modifications have been proposed based on K-means, such as k-medoids [29] to handle the sensitivity to outliers and noise and non-numerical variables.

In the probabilistic view, data points are generated as a sample drawn from a mixture model of several probability distributions [26]. Each data point is assumed to belong to only one cluster and the possibility of the assignment is estimated through the likelihood of the data from a given mixture model which can include Bernoulli, Poisson, Gaussian and log-normal distributions [7]. The two-step iterative Expectation-Maximization (EM) method [27] is generally used to solve the maximization of the objective function of log-likelihood.

3 FORCE TRANSFER FUNCTIONS

In force-directed simulations to layout the graph [10] [13], the forces between nodes are modelled to compute the minimum potential energy of the graph. In this paper, we define the force between nodes as a function \( F_{FTF}: \Omega \times \Omega \rightarrow \mathbb{R} \) that maps a pair of physical bodies to a force intensity. \( \Omega \) is a group of physical bodies where
each \( \omega \in \Omega \) has some location in \( \mathbb{R}^2 \) or \( \mathbb{R}^3 \) and a set of attributes. The minimum potential energy of the graph is determined solely based on the construction of the force functions. Different force functions that relate node or edge attribute values to force parameters in a different way will generate different layouts even with the same data. We call FTF force transfer functions since they specify how node attribute values are eventually mapped to forces in order to flexibly produce different graph layouts.

Before we describe our force transfer functions, we first introduce some definitions used in this paper.

3.1 Graph Definitions
A graph \( G \) is defined as \( G = (V, E, A, B) \) such that:
\[ V = \text{a set of unimodal (single type) nodes}, |V| = n \]
\[ E = \{ (v_i, v_j) : v_i, v_j \in V \text{ and } i \neq j \} \text{ is a set of relationships, or edges, within } G, |E| = m \]

\( A \) is an \( n \times k \) matrix representing the attributes of every \( v_i \in V \)
\( B \) is an \( m \times l \) matrix representing the attributes of every edge \( e_k \in E \).

In other words, \( G = (V, E) \) is an undirected graph in the traditional sense. The matrices \( A \) are what make \( G \) more general by allowing each node to have \( k \) attributes. That is, the \( a_{ij} \in A \) entry is the value of the \( j^{th} \) attribute for node \( i \). Similarly, the \( b_{kl} \in B \) is the value of the \( k^{th} \) attribute for edge \( s \).

Singh et al. describe in [36] a \( M \times 3 \) graph is a multi-modal, multi-featured, multi-relational graph. That is, there are multiple types of nodes (modes), multiple attributes for each node (features), and more than one type of edge (relationships). These graphs are essentially the focus of this paper, but we make the simplifying assumption that \( G \) is not multi-modal, i.e. there is only one type of node. However, multiple node types could be accounted for by appending a type attribute to \( A \) and assigning each node type a value.

3.2 Force Transfer Function
Force-directed methods construct drawings of general graphs using a physical model where the vertices and edges of the graph are viewed as physical objects subject to various forces. In the traditional force layout method presented by Eades [10], every node in the graph is treated as a body with some mass, and edges act as springs connected to each body. According to the mechanical simulation system constructed by the vertices and edges of the graph, two kinds of forces are needed to be considered in order for the system to reach an equilibrium state of minimum energy. One is the gravitational force between every pair of nodes (whether connected by an edge or not), and the other is the spring force which only occurs between nodes connected by an edge. Our force transfer functions are mapped to these two kinds of forces and can be designed as the functions in equations (3) and (4) which depend on a pair of nodes as input.

\[
gravity(v_i, v_j) = gravity_{TF}(v_i, v_j) \tag{1} \\
spring(e_k = (v_i, v_j)) = spring_{TF}(e_k) \tag{2} \\
\]
In our framework, the force transfer functions \( gravity_{TF} \) and \( spring_{TF} \) can be arbitrary functions. Different transfer functions produce different layouts. As for the implementation in this paper, we focus our investigation on force-directed layout principles [10] to design concrete force transfer functions. Let \( m(v_i) \) be the mass of node \( v_i \), \( K \) be the spring constant for every spring, \( L \) be the rest length of every spring, \( G \) be the gravitational constant, \( \mathbf{p}(v_i) \in \mathbb{R}^2 \) be the position of node \( v_i \) at time \( t \), and \( r = ||\mathbf{p}(v_i) - \mathbf{p}(v_j)|| \) . Then, we have the following equations from traditional graph layout:

\[
gravity(v_i, v_j) = -Gm(v_i)m(v_j) \frac{v_i - v_j}{r^2} \tag{3} \\
spring(e_k = (v_i, v_j)) = -K(r - L) \tag{4} \\
\]

or
\[
spring(e_k = (v_i, v_j)) = -K \log \frac{r}{L} \tag{5} \\
\]
Each of these functions represent forces between pairs of nodes. Notice that while we refer to equation (5) as a gravitational force, it is actually a repulsive force due to the sign of \( G \). Equations (6) and (7) are two alternative spring forces between two vertices connected by an edge. Equation 6 computes the forces according to the Hooke’s Law while equation 7 adopts the non-physically based logarithmic strength of the spring. In theory, either spring force function can be adopted as our force transfer functions. Additional non-physically based parameters such as graph-theoretic distance between nodes can also be used. In this paper, we specifically demonstrate how to use Equation 6 when calculating spring forces.

Traditional graph layouts usually use constant \( G, K, \) and \( L \). In this paper, we want to produce graph layouts that also take into account node attribute values. Hence in our design, we make \( G, K, \) and \( L \) be functions of node attribute values, i.e., given that \( A \) represents the set of attributes for all the nodes (as described in section 3.1) and \( A(v_i) \) is the attribute value of node \( i \), we replace \( G, K, \) and \( L \) in equations (5) and (6) with functions of the form \( V \times V \mapsto \mathbb{R} \) to get:

\[
gravity_{TF}(v_i, v_j) = G(A(v_i), A(v_j)) \frac{m(v_i)m(v_j)}{r^2} \tag{6} \\
spring_{TF}(e_k = (v_i, v_j)) = K(A(v_i), A(v_j))(r - L(A(v_i), A(v_j))) \tag{7} \\
\]

With the modifications above, the force transfer functions \( G, K, \) and \( L \) allow us to produce layouts that are also dependent on node attribute values. It is also worth mentioning this layout is largely independent of the initial position of each node.

Before we show how to compute the force transfer functions based on node attributes, we first examine these three force parameters which serve specific purposes in Equations (8) and (9):

1. Gravity(\( G \)): controls how much nodes attract/repel each other.
2. Spring constant(\( K \)): controls how stiff, i.e. how strict or important edge length is in the layout.
3. Spring length(\( L \)): controls the desired edge length.

Figure 2: An array of images illustrating how the modification of \( L, K, \) and \( G \) affect the relative positions of the nodes.

Figure 2 shows how the values of these three parameters influence the graph layout. In the first row of Figure 2, we show the graph layout results of changing \( L \) while fixing the other two parameters. We find that the nodes are compressed together for low \( L \) and repelled apart for high \( L \) values. Similar observation can be applied to parameter \( G \), while the opposite is true for \( K \).

According to such observations, we can map the above three FTFs to different semantics. For instance, if two nodes are considered to be similar based on some similarity metric, then the repulsive force \( G \) between them needs to be low to make similar nodes...
cluster together, vice versa for dissimilar nodes. Similar associations can be made with the spring-force related parameters, where it only applies to nodes that are connected together by an edge. The above three functions can be manipulated to perform nodes clustering based on their attribute values. In general, nodes with similar attributes to nodes can be clustered by
1) lowering the repulsive force,
2) increasing the spring coefficient, and
3) decreasing the spring length.

With these guidelines, we can design our $FTFs$ to effect a desired graph layout. For example, to cluster nodes with high-dimensional attributes, we can use $G$ by itself to compute the layout of the nodes. We will illustrate this in Section 4.4.

4 Design and Manipulation with Force Transfer Functions

In this section we show how to design force transfer functions to associate node attributes to force parameters. We also show how to manipulate these force transfer functions to produce different graph layouts.

4.1 2D Transfer Functions

The three $FTFs$, i.e. $G$, $K$ and $L$, all require two inputs and produce one output. An intuitive way to express the relationships among the 3 variables is with 2D transfer functions. We provide the interface for a user to manipulate force transfer functions via a simple, bi-linear patch editor as shown in Figure 3. The horizontal $xy$-axes are mapped to attribute values of node $v_i$ and node $v_j$, while the vertical $z$-axis represents the strength of the output force parameter. In our implementation, users can specify and interactively manipulate such surface $\Phi$ by simply clicking and moving the 4 corner end points. In this paper, $\Phi$ is a bi-linear surface by default, though this does not need to be the case in general. With $\Phi$ defined, the $z$ component of $\Phi(A(v_i), A(v_j))$ indicates the force magnitude between nodes $v_i$ and $v_j$. The node attributes for 2D transfer functions are assumed to be real values in the range $[0,1]$.

While the 2D transfer function allows one to easily see the relationship of a single pair of node attribute values to a force parameter, it is not as intuitive if we need to relate multiple node attribute value pairs.

4.2 Serialized Transfer Functions

In order to facilitate the manipulation of the force transfer functions to include multiple attributes, we decompose the 2D transfer functions above into two transfer functions by separately computing the attribute metric function $\delta$ and 1D force transfer function $F$. The two transfer functions are serialized in that the output of one is the input to the other. That is, the output $m$ of $\delta$ becomes the input of the force transfer function $FTF$.

\[
m_{ij} = \delta(A(v_i), A(v_j))
\]

\[
FTF = F(m_{ij}) \quad F \in \{G, K, L\}
\]

This decomposition allows the computation based on the two nodes’ attributes (e.g. metrics) to be separated from the force computations. The function $\delta(\cdot)$ can be replaced by an arbitrary function to compute the distance (similarity) between a pair of node attributes. The attributes can vary in their dimensions from 1D to high dimensions. Any meaningful distance metric such as Euclidean-based, Manhattan-based or Chamfer-based distance function can be adopted to compute the distances of two nodes in their attribute space. In this paper, the output of the attribute metric function $\delta(\cdot)$ is computed as a real number normalized in the range of $[0,1]$. Since $F(\cdot)$ is a 1D transfer functions, they can be easily edited as well using other published methods e.g. representing them as polylines or curves.

4.3 Comparisons between 2D and Serialized Transfer Functions

In this section, we apply both 2D and serialized transfer functions to the same dataset. We then show some graph layouts that will be difficulty to specify using 2D transfer functions. For each graph layout produced in this paper, the attribute value is mapped to the color of the node by linearly interpolating between blue (low value) and red (high value). All the edges are set to the color of green.

We use the author cooperation data from the citation database (http://www.cdblp.cn) to demonstrate 2D vs serialized transfer function designs. This data includes 1000 authors in Computer Science where each author is represented by a node. The number of published papers of each node is a node attribute. A single edge between two authors represents at least one joint paper. An author can have single-authored paper increasing the node attribute value without increasing the degree of the node. A multi-authored paper will result in as many edges amongst the co-authors even if it is just for a single paper. Because of these, trying to find the author with the most number of papers is difficult if one simply considers the presence or absence of edges, and do not take node or edge attribute values into account.

* The design of 2D transfer functions
  With 2D transfer functions proposed in section 4.2, one can design force transfer functions as shown in Figure 4 to make highly productive authors stand out from their peers. So when the node attribute values are both 0, $G$ and $L$ have the minimum values and $K$ have the maximum values. When either node has an attribute value of 1, $G$ and $L$ have the maximum value and $K$ has the minimum value. In Figure 5 we show the corresponding visualization result which identifies the author with most publications. The red nodes correspond to authors with many published papers while the blue nodes correspond to authors with few published papers.

* The design of serialized transfer functions
  For serialized transfer functions, we design the attribute metric function as shown in Equation (12) to make highly productive authors stand out from their peers. In the following equation, $a_i$ and $a_j$ represent the attribute of published papers of each node.

\[
\delta = \max(a_i, a_j)
\]
Equation (12) simply returns the larger of the two attribute values. The 1D force transfer function is shown in Figure 6. With such 1D transfer function $G$ and $L$ are set to be the minimum values when $\delta$ is zero. This implies that for both nodes with few papers the repel force is very small, while for either nodes which has a lot of papers, the repel force is large. This transfer function makes the author that has most papers repelled from other nodes. We get the corresponding visualization result which identifies the author with most publications in Figure 7.

In Figure 8 we show the graph layout result of the same author cooperation data that is produced under a different attribute metric function (the force transfer function is the same as the one shown in Figure 6). The attribute function is shown in Equation (13), where two attributes of published papers $a_i$ and node degrees $b_i$ (the number of collaborators) are linearly combined.

$$\delta = \max(a_i, a_j) \times 0.5 + \max(b_i, b_j) \times 0.5$$  \hspace{1cm} (11)

With this attribute metric function, the repelled peripheral nodes in the graph layout identifies the authors with the most published papers and collaborators in Figure 8.

### 4.4 Clustering High-dimensional Data

If the attribute metric function $\delta$ is defined as the distance function to compute the similarity between two nodes in attribute space, we can achieve data clustering of nodes with similar attribute values. Since $\delta$ can be specified to deal with node attributes of arbitrary dimensions, this results in a method which produces a clustering of $k$-dimensional data, regardless of how large $k$ is.

To demonstrate the clustering ability, we apply our method to cluster users from last.fm [1], a social music service website. We use the last.fm data released in [23] that contains the subset of 1892 users from the website.

We would like to visualize users according to their musical tastes. Users form clusters based on the similarity of their musical tastes. Suppose node A’s favorite artists are \{a_1, a_2, \ldots, a_m\}, node B’s favorite artists are \{b_1, b_2, \ldots, b_n\}. We can retrieve the similarity $S(a_i, b_j)$ between any two artists $a_i, b_j$ through the last.fm web
service API. In this example, the similarity between any two nodes \( \delta(A,B) \) is computed using the following equation:

\[
\delta(A,B) = \sum_{i=1}^{m} \max(S(a_i,b_1),S(a_i,b_2),\ldots,S(a_i,b_n)) + \sum_{j=1}^{n} \max(S(b_j,a_1),S(b_j,a_2),\ldots,S(b_j,a_m)) - \frac{m}{n}
\]

(12)

Figure 9: Clustering of the users of Last.fm based on their musical interests that takes three forces into account.

The visualization of the graph layout is shown in Figure 9. The red nodes represent the users with many favorite artists while the blue nodes represent the users with few favorite artists. We use the same force transfer function as the one shown in Figure 6. From Figure 9, we find that except the small number of nodes at the outer ring which have few connections with the other nodes, the majority nodes form a large inner large circle where no clear cluster of interesting patterns can be found. To achieve node clustering based on attribute similarity, we can solely use gravity force without taking edge data into account. Such force function can give one an idea of how large each cluster is (i.e., total number of points in a cluster) and how disparate clusters are. This method will also create clusters without specifying a priori the number of desired clusters. In contrast, one must specify the desired number of clusters (K) for the popular K-means algorithm.

However, we cannot directly use the force transfer function in Figure 6 where the parameter \( G \) is always the positive value (repel force). Instead, we design the force transfer function as shown in Figure 10. When \( \delta \) is within the range of \([0,1]\), \( G \) is negative for attractive force. Otherwise, \( G \) is positive for repelling force. Furthermore, we make a distinction between attractive and repulsive force in our calculation for gravity forces so as to avoid the layouts that are greatly dependent on the initial position of each data point. To further avoid the problem of each cluster being represented as a single point, we add a distance threshold \( T \geq 0 \) in the gravity force function:

\[
\text{gravity}_{TFC} = \begin{cases} 
0 & \text{if } G > 0 \text{ and } r < T \\
\frac{gm(v_i)m(v_j)r^2}{T^2} & \text{if } G > 0 \\
\frac{gm(v_i)m(v_j)}{r^2} & \text{if } G \leq 0
\end{cases}
\]

(13)

Figure 10: The force transfer function of \( G \)

\( T \) allows us to control how visually dense/sparse each cluster will be. If \( T = 0 \), the first condition of \( \text{gravity}_{TFC} \) will never be satisfied, hence each cluster will still be displayed as a single point. If \( T \) is too large, then the first condition of \( \text{gravity}_{TFC} \) will always be true, resulting in a force magnitude of 0 between all similar nodes.

The advantages to this technique (i.e., attribute and force transfer functions) arise when dealing with higher dimensional data (e.g., \( k \geq 3 \)). Even when \( k = 3 \), it can be difficult to visualize all of the clusters in a dataset simultaneously due to occlusion, and any dataset where \( k > 3 \) becomes increasingly difficult to understand, much less visualize in any way. Our method can provide a visual, 2D clustering, which depends only on the attribute metric function \( \delta \).

For the data of Last.fm, we computed the gravitational force between any two nodes using Equation (15). Note that the range of \( \delta(A,B) \) is not guaranteed to be within \([0,1]\). So we need to normalize \( \delta(A,B) \) to the range of \([0,1]\) before we adopt Equation 15.

Figure 11: Clustering of last.fm users using only gravitational forces.

In Figure 11 we show the clustering result using our method. We find that compared with Figure 9 there is one obvious cluster in the center of the image while the other nodes are repelled and distributed evenly on the ring of circle. The central cluster is enlarged to give a clear view of the inner structure of the cluster (the number of nodes in the central cluster is 472). Looking at the statistics of the favourite artists tags of the central cluster nodes, we learn that Rock and Pop are the two most frequently appearing tags. From this layout result, we also learn that the other nodes who enjoy other styles of music have little interests in Rock and Pop and furthermore, most of the other nodes themselves are not similar to each other in musical tastes since they repel each other on the ring of the layout.

5 Case Study

We applied our transfer function technique on a political blog network [3]. This dataset has a single attribute value for each node aside from node centrality metrics computed using igraph [9]. Besides, we computed the following standard centrality metrics for each node in the network:

- Degree centrality: number of adjacent edges to a particular node.
- Betweenness centrality: the number of shortest paths a node lies on.
displaying them farther from the other nodes; that is, they visually become outliers. This supports Adamic and Glance’s findings\cite{3} by showing some red nodes with many links to blue nodes with some red nodes actually having much fewer links to other red nodes than to blue nodes. Besides the above insight, we also make the sense of how large the main blue and red clusters are in the visualization by increasing the repulsive force between red-blue and red-red nodes to the value larger than 0. To avoid the larger clusters enveloping the outliers as in the default layout, we also increase the edge length between red-blue nodes even further. The result of this interactive fine-tuning is shown in Figure 1(b).

The graph layouts in Figure 1(a) and 1(b) are based on the political value attribute only. Here, we wish to highlights nodes with high betweenness and yet laid out such that blue nodes will be placed near each other and red nodes will be placed far away from each other as well as from blue nodes. This can be achieved by assigning a minimum repulsion force between blue-blue nodes and a maximum repulsive force between blue-blue and red-red nodes. Similarly, we assign the minimum spring rest length to blue-blue edges and maximum rest length for red-blue and red-red edges. The result can be seen in Figure 1(c). The single node within the dotted circle was George W. Bush’s campaign blog during the 2004 election.

5.2 Serialized Transfer Function for Political Blog Network

So far, all of the above examples are produced by 2D transfer functions using either the political value or the betweenness attribute. To compute a graph layout using both attributes simultaneously, we can design the attribute metric function as:

\[
\delta = 0.5 \times |a_i - a_j| + 0.5 \times \max(b_i, b_j)
\]

(14)

where \(a_i\) is the left/right value of node \(i\) and \(b_i\) is the betweenness centrality of node \(i\). The above function extends the attributes metric function to deal with both left/right value attribute and connectivity information. The force transfer function is shown in Figure 14. From the visualization result shown in Figure 1(c) we observe that this graph layout highlights the node with the high betweenness as well as the interaction of left and right groups. Actually, the visualization in Figure 1(c) shows the combination of the insights in Figure 1(b) and Figure 1(c). This demonstrates the flexibility of the serialized transfer functions to utilize multiple attributes to influence the graph layout.

To compare our graph layout with that of MDS (multi-dimensional scaling), we use MDS to layout the nodes based on the metrics computed by Equation (16) and the result is shown in Figure 15. We find that with the same metrics of any two nodes,
MDS and our method give two totally different results. For MDS, only two clusters are shown and no additional insights can be extracted.

6 Conclusion And Future Work

This paper presents a method for dynamically changing a graph layout based on node and/or edge attributes using force transfer functions. This strategy allows a user to dynamically highlight nodes based on various node attribute relationships e.g., (low, low), (low, high), etc. These functions are extremely flexible and theoretically, one can generate an infinite combination of transfer functions. To help users get started, we provide some standard template transfer functions e.g. for clustering. While the flexibility of these transfer functions makes them a powerful tool, manipulating them in 3D can be cumbersome. And while we provide some template functions that can be further customized, it would also be beneficial to be able to generate transfer functions programmatically. For instance, allowing the user to enter a logical expression that can then be interpreted as a transfer function. Another avenue of future research is on automatic generation of transfer functions that produce aesthetically pleasing layouts.

The technique presented in this paper is restricted to using numerical attributes in the range [0; 1]. There are many cases where it does not make sense to translate categorical data into numerical data. Accounting for categorical data is left as future work.

While there are many avenues to explore further, the flexibility of using force transfer functions for graph layout can be used in other visual analysis tools for understanding the increasingly complex network data that is generated on a daily basis today.

References