Graph Layout with Versatile Boundary Constraints

Yani Zhang\textsuperscript{1} and Alex Pang\textsuperscript{2}

\textsuperscript{1}Expedia Inc., \textsuperscript{2}Computer Science Department, University of California Santa Cruz

Graph layouts are in general data dependent and help to reveal structural and attribute relationships in the data set. However, there are situations when one may wish to alter the layout e.g. to emphasize parts of the data set or for aesthetic reasons. This paper strives to meet that need for the case of force-directed graph layout algorithms. Our approach is to add boundary constraints to specify where graph nodes may or may not be positioned. Users can interactively draw one or more boundaries. Boundaries may self-intersect and define different topology e.g. donut or figure eight shapes. Additional control, subject to the density of nodes, can impart different density distributions within defined boundaries. We tested the feasibility of this concept on several data sets and different boundary constraints.

I. INTRODUCTION

Graph drawing has been an area of active research interest in the information visualization community as abstract graph models are widely used in various areas ranging from social networks, security, scientific applications, and others. Among those techniques \cite{1, 15, 7, 20, 2}, force-directed approach \cite{3, 9, 5, 21, 11, 12, 13, 4, 18, 23, 29, 16, 24} is the most well-known method for drawing undirected graphs due to their flexibility of adding constraints, ease of implementation and generally satisfactory layouts. Specifically, a graph is viewed as a neat drawing of the node particles that can be generated by minimizing an energy function \cite{20}.

Based on this force-directed idea, many practical graph drawing systems have been developed. In general, layout is dictated by using a repulsive force to separate vertices apart from each other and a pseudo-gravity force to hold the entire graph together but without constraining the size and shape of the graph. In certain situations, one would like the layout to meet some specific requirements or common aesthetics such as symmetry, minimum edge crossings, etc. while at the same time, obtain the desired graph with a minimum amount of time. With these goals in mind, various kinds of constraints were taken into account in constrained graph layouts \cite{14, 25, 19, 8}. Widely used graph drawing constraints include placing a given vertex in the center or on the outer boundary of the drawing, placing a group of vertices as a cluster, or aligning vertices horizontally or vertically \cite{25}. Thus, drawing a graph using force-directed methods can be formalized as a complex multi-objective optimization problem.

In this paper, we propose an alternative approach to specify constraints by allowing the users to interactively draw a boundary wherein the graph layout will be constrained. We model these boundaries as a set of additional environmental forces that contribute to the forces acting on the vertices in the graph. Since our approach is based on force-directed simulation, it can take advantage of the existing optimization results from other force-directed graph layout algorithms.

The remainder of this paper is organized as follows. We discuss related work with aspects of interactivity and constraints in graph drawing methods in section II. Section III provides a basic background on force-directed methods and the definition of our graph drawing problem. In section IV we give a formal definition of boundary constraints and present our new force-directed model with boundary constraints. Implementation work is described in section V and boundary constrained layout of graphs and analysis of results are presented in section VI. We provide conclusions and identify some avenues for future work in section VII.

II. RELATED WORK

A. Interactivity

There are many interactive constraint-based graph layout systems in existence today. However, they do not include outside environment force as additional constraints to the graph drawings. For example, the GLIDE (Graph Layout Interactive Diagram Editor) system \cite{22} is a graph editor for drawing medium-sized graphs that organizes the interaction within a vocabulary of specialized constraints for graph drawing. CGV (Coordinated Graph Visualization) \cite{26} is another graph visualization system that incorporates several interactive views to address different aspects of graph visualization. These graph drawing systems indeed focus on interaction but did not support interactively defining boundary constraints for graph layouts.

Alternatively, there are constraints-driven layout algorithms for network diagrams \cite{6}, which propose a variety of layout techniques to exhibit the Visual Organization Features (VOFs). VOFs are arrangements of related vertices in the diagram including horizontal and vertical alignment, axial and radial symmetries, etc. However, these VOFs do not include constraining the graph within an area where the desired shape can be achieved. Such boundary constraints can be useful in many applications such as automatic graph layout, network graph analysis and visual design. Note that while very powerful, focussing mechanisms used in magnification, fisheye, or other layout lenses \cite{27} are different in that they typically deal with only a subset of the graph or the layout space.

B. Constraints in Graph Drawing Methods

Traditional methods that incorporate boundary constraints controlled the size of the graph layout by assuming that
the boundary of the pre-specified drawing region acted as a wall [12]. Regions were rectangular in shape and were represented as inequality constraints wherein graph vertices must lie. No forces were used in their formulation.

Other forms of constrained graph layout models have also been proposed. A formalism for the declarative specification of graph drawing with Prolog and an associated constraint-solving mechanism have been developed by Kamada [17]. Using this formalism, one can express several simple geometric constraints among the vertices, such as horizontal or vertical alignment, and circular arrangements. Dengler [6] provided a notation for describing the desired perceptual organization of a layout of a graph by means of a collection of layout patterns based on positions. These include clustering, zoning, sequential placement, T-shape, and hub shape. This method incrementally improves an initial randomly-generated drawing.

A comprehensive approach for constrained graph drawing was presented by He and Marriot [14]. They provide a general model that supports: (i) the specification of arbitrary arithmetic linear equality and inequality constraints on the coordinates of the vertices; (ii) suggests coordinates for the vertices, each with an associated weight, which denotes the strength of the suggestion. They show how to extend the force-directed approach by Kamada [18] to support such placement constraints which is fast and produces good results in practice.

The above mentioned force-directed methods are all supported by the position constraints of vertices or fixed-subgraph constraints in the graph, but our approach is based on adding an additional force to constrain the whole graph. The benefit of our approach is that the additional boundary forces are processed in the same way as forces acting on graph elements and therefore converge at the same rate. In contrast, with the other methods mentioned above, more work has to be done after each iteration to take into account position constraints. Thus, our approach achieves better interactivity with user because the graph would take boundary constraints into consideration while converging to the final layout.

III. FORCE-DIRECTED GRAPH LAYOUT

A. Definitions

1) Graph: In force-directed methods, a graph $G$ is defined as a pair $(V_G, E_G)$ where $V_G$ is a set of vertices and $E_G$ is a set of edges $E_G \subseteq V_G \times V_G$. A drawing of such graph $G$ on the plane is defined as a mapping $D$ from $V_G$ to $\mathbb{R}^2$, where $\mathbb{R}$ is the set of real numbers. Then for mapping $D$, each vertex $v \in V_G$ is placed at point $D(v)$ on the plane, and each edge $(u, v) \in E_G$ for $u, v \in V_G$ is displayed as a straight-line segment connecting $D(u)$ and $D(v)$. In our graph drawings, we use a dot on the plane to represent a vertex and a straight line connecting two vertices to represent an edge.

2) Boundary: Similarly, a boundary $B_p$ of the graph is defined also as a pair $(V_{B_p}, E_{B_p})$. Suppose boundary $B_p$ has $n$ vertices (which we will refer to as boundary vertices to distinguish them from graph vertices) and $m$ edges (which we will refer to as boundary edges to distinguish them from graph edges), then $V_{B_p}$ is defined as a set of boundary vertices $V_{B_p} = \{v_p(1), v_p(2), \cdots, v_p(n)\}$ and $E_{B_p}$ is defined as a set of boundary edges that connects adjacent vertices and forms a connected path. That is, $E_{B_p} = \{e_p(1), e_p(2), \cdots, e_p(m)\}$ where $e_p(i) = \{v_p(i) v_p(i+1)\}$ ($i = 1, 2, \cdots, m - 1$) and for special case $i = m$, $e_p(m) = v_p(m) v_p(1)$. Note that while we use the graph notation to represent a boundary, it is a special case where $n = m$, and it forms a closed loop. The coordinates of the boundary vertices in the mapping $D$ are defined interactively by users, so the boundary can be deformed into arbitrary shapes and may even self-intersect. Multiple boundaries in the same graph are allowed to support boundaries of different topologies. In these cases, the boundaries are represented by a set composed of $B_1, B_2, \cdots, B_q$ to specify $q$ distinct boundaries. While there is flexibility in terms of how boundaries are represented and the different types of topology that one can construct, one must nevertheless be careful about the semantics of these boundaries in terms of using them as boundary constraints for graph placement. Also, while an individual boundary may self-intersect, we do not allow a boundary to intersect with another boundary.

B. Common Forces

Now we introduce some common forces in classical force-directed methods. The graph drawing algorithm of Tutte [28] is one of the earliest force-directed method in literature. The model they proposed partitions the set of vertices into two sets, a set of fixed vertices and a set of free vertices. By nailing down the fixed vertices as a strictly convex polygon and then placing each free vertex at the barycenter of its immediate neighbor during each iteration, the model is able to yield a nice drawing. Subsequently, Eades [9] proposed a simple spring embedder algorithm which most models today, including our proposed model, is built upon.

In that model, every pair of vertices is connected by a spring (although Eades’ model did not use Hooke’s Law). For adjacent vertices (vertices that are connected to each other by an edge), the intensity $f_s$ of the attractive spring force exerted on the two vertices depends on the current distance between them according to the following formula:

$$f_s(\|\overrightarrow{uv}\|) = c_1 \cdot \log \left( \frac{\|\overrightarrow{uv}\|}{c_2} \right) \cdot \frac{1}{\|\overrightarrow{uv}\|}$$

(1)

where $c_1$ represents a scaling constant for spring force, $c_2$ is the given spring natural length, and $\overrightarrow{uv}$ denotes the vector from vertex $u$ to vertex $v$.

For non-adjacent vertices (vertices that are not connected to each other by an edge), the spring has infinite natural length, thus always has a repelling force. The intensity $f_r$ of the repulsive force exerted on the two vertices depends on the distance between them:

$$f_r(\|\overrightarrow{uv}\|) = -\left( \frac{c_3}{\|\overrightarrow{uv}\|^2} \right) \cdot \frac{\|\overrightarrow{uv}\|}{\|\overrightarrow{uv}\|}$$

(2)

where $c_3$ is the scaling constant for repulsive forces.

In general, various modifications on force-directed approaches fall into two categories. One has to do with altering the repulsive force and the spring force models, while the other attempts to manipulate the local minima problem resulting...
from the equilibrium between repulsive forces and the spring forces. This paper is based on the first approach, where we add an additional force representing the boundary constraints into the graph layout optimization process.

C. Graph Drawing Problem

The graph drawing problem considered in our paper is addressed as follows: Suppose we begin with a randomly positioned drawing of a graph $G = (V_G, E_G)$ and a set of boundaries $\{B_1, B_2, \ldots, B_q\}$ to specify $q$ distinct boundaries, the layout algorithm should solve the optimization problem and satisfy the following goals:

- Minimize energy configuration.
- Every vertex of the graph is within the defined boundaries.
- The final layout of the graph preserves the properties of force-directed methods.

IV. LAYOUT WITH BOUNDARY CONSTRAINTS

In this section, we first give a formal definition of boundary constraints and the forces they induce on the graph elements. Then we discuss our layout algorithm in depth including how the attractive and repulsive forces are modified to account for boundaries, and how the boundary forces are calculated. Finally, we present the algorithm for handling boundary constraints.

A. Definition of Active Area

Boundary constraints are enforced via boundary forces on graph elements. In order to determine how these boundary forces affect graph elements, we must determine the set of boundary edges that can influence an individual graph element, or conversely, the set of graph elements affected by a boundary force. For this purpose, we define the active area of a set of boundaries, and the active area of each boundary edge.

1) Active Area of A Set of Boundaries: A boundary specifies a partitioning of the space wherein a graph is to be drawn. For closed boundaries, we need to distinguish between the inside and outside of the boundary. By convention, we will assume that boundary vertices are ordered in a counterclockwise manner so that the inside is to the left of a boundary edge (see Fig. 1(a)). Furthermore, each boundary $B_p$ encloses an area. For simplicity, we will use the notation $B_p$ to represent both the boundary and its enclosed area. Each boundary has its own active area $A_p$ which is encompassed by $B_p$. For the case where there is only one boundary, as illustrated in Fig. 1(a), the active area is $A_p$ ($p = 1$). In general, if we have more than one boundary and there are containments between those boundaries, then we define the active area $A$ as the biggest connected area where the graph layout can take place. For example, let us assume $B_1$ is the outer boundary, with smaller boundaries scattered within $B_1$ as shown in Fig. 1(b). Then the active area in that figure is $A = B_1 - (B_2 + B_3)$. In general, assuming that boundaries do not intersect each other i.e. there is only a single connected area whose outer perimeter is specified by $B_1$, and that $B_2 \cdots B_q$ do not intersect each other and do not contain another boundary within each of them, then the active area of these boundaries can be expressed as $A = B_1 - (B_2 + B_3 + \cdots + B_q)$. Violating these assumptions would mean that there are several disjoint areas, rather than a single contiguous area, where a single graph must be laid out. Any vertex falling inside the active area $A$ will have boundary forces acting upon it.

2) Active Areas of A Boundary Edges: In general, a boundary edge will exert an inward force perpendicular to the boundary edge on graph elements in order to keep them within the boundary. However, not all boundary edges will affect graph vertices at all times. A graph vertex $v_G(j)$ is influenced by a boundary edge only when it is within the active area of that edge. Given a boundary edge $e_p(i)$ belonging to boundary $B_p$, the active area of $e_p(i)$ is the half space bounded on the left by a vector (which we call the left vector) from boundary vertex $v_p(i)$ and on the right by a vector (which we call the right vector) from boundary vertex $v_p(i + 1)$ as indicated in Fig. 2. The four graph vertices within the active area of $e_p(i)$ are each assigned a boundary force $f_{B_p}(v_G(j), e_p(i))$ (see Section IV-C) with a direction perpendicular to $e_p(i)$ and pointing towards the interior of the boundary. For full
consideration of how to define left and right vectors, we need to look at the types of boundary vertices that make up a boundary edge. That is, whether the boundary vertex is concave or convex.

![Diagram](image)

**Fig. 2.** Definition of active area of boundary edges.

In Fig. 3, boundary edge \( e_p(i) \) is drawn as a solid vector from boundary vertex \( v_p(i) \) to \( v_p(i+1) \). It is bounded on the left by boundary edge \( e_p(i-1) \) and on the right by boundary edge \( e_p(i+1) \). A boundary vertex, e.g. \( v_p(i) \), is a convex boundary vertex if the interior angle of the boundary edges that share that vertex i.e. angle from \( e_p(i-1) \) to \( e_p(i) \), is less than or equal to 180 degrees. Otherwise it is a concave boundary vertex, e.g. \( v_p(i+1) \).

![Diagram](image)

**Fig. 3.** Convex and concave boundary vertices.

The active area of a boundary edge depends on the type of boundary vertices that make up an edge. As we process the boundary edges in a counter-clockwise manner, there are four cases to handle:

**Case 1:** Both \( v_p(i) \) and \( v_p(i+1) \) are convex (see Fig. 4(a)):

If both boundary vertices of boundary edge \( e_p(i) \) are convex, then the active area of boundary edge \( e_p(i) \) is bounded by left vector \( -e_p(i-1) \), boundary edge \( e_p(i) \) and right vector \( e_p(i+1) \). A force perpendicular to boundary edge \( e_p(i) \) is applied to graph elements that are on the half space inside of boundary edge \( e_p(i) \), and also inside of boundary edge \( e_p(i-1) \) and boundary edge \( e_p(i+1) \). Otherwise, boundary edge \( e_p(i) \) does not contribute a force to the graph element.

**Case 2:** \( v_p(i) \) is convex and \( v_p(i+1) \) is concave (see Fig. 4(b)):

We define the active area of boundary edge \( e_p(i) \) to be bounded by left vector \( -e_p(i-1) \), boundary edge \( e_p(i) \) and a vector from \( v_p(i+1) \) to \( v_2p(i+1) \) as the right vector. \( v_1p(i+1) \) is perpendicular to \( e_p(i) \) and \( v_2p(i+1) \) is perpendicular to \( e_p(i+1) \). For graph elements falling in the area bounded by left vector \( -e_p(i-1) \), boundary edge \( e_p(i) \) and vector \( v_p(i+1) v_1p(i+1) \), we apply a force perpendicular to boundary edge \( e_p(i) \). For graph elements falling in the triangular gap bounded by vector \( v_p(i+1) v_1p(i+1) \) and right vector \( v_p(i+1) v_2p(i+1) \), we apply a force in the direction from \( v_p(i+1) \) to the graph element.

**Case 3:** \( v_p(i) \) is concave and \( v_p(i+1) \) is convex (see Fig. 4(c)):

We define the active area of boundary edge \( e_p(i) \) to be bounded by a vector from \( v_p(i) \) to \( v_2p(i) \) as the left vector, boundary edge \( e_p(i) \), and boundary edge \( e_p(i+1) \) as the right vector. Graph elements in \( A_1 \) are applied a force perpendicular to boundary edge \( e_p(i) \). For graph elements in the triangular gap bounded by vector \( v_p(i) v_1p(i) \) and left vector \( v_p(i) v_2p(i) \), we apply a force in the direction from \( v_p(i) \) to the graph element. Note that if \( v_p(i) \) is a left concave boundary vertex of boundary edge \( e_p(i) \), then it is also the right concave boundary vertex of boundary edge \( e_p(i-1) \) which is handled by case 2 above.

**Case 4:** Both \( v_p(i) \) and \( v_p(i+1) \) are concave (see Fig. 4(d)):

For this case, we define the active area of boundary edge \( e_p(i) \) to be bounded by a vector from \( v_p(i) \) to \( v_2p(i) \) as left vector, boundary edge \( e_p(i) \), and a vector from \( v_p(i+1) \) to \( v_2p(i+1) \) as right vector. Just as in cases 2 and 3, a left concave boundary vertex will be bounded by a vector that is perpendicular to the edge. Likewise, a right concave boundary vertex will be bounded by a vector that is perpendicular to the next boundary edge.

**B. Different Types of Active Area**

Here, we consider the number of active areas resulting from either a single or multiple boundaries, and whether their shape or arrangement result in a single or multiple active areas where graph elements will be constrained. There are four possible configurations which we discuss below.

1) **Single Boundary and Single Active Area:** (Fig. 1(a)) This is the simplest case, where the boundary constraint is specified by a single boundary and where the edges in this boundary do not self-intersect. Graph elements will have forces applies
(a) Boundary vertex \(v_p(i)\) is convex, boundary vertex \(v_p(i+1)\) is also convex.

(b) Boundary vertex \(v_p(i)\) is convex, boundary vertex \(v_p(i+1)\) is concave.

(c) Boundary vertex \(v_p(i)\) is concave, boundary vertex \(v_p(i+1)\) is convex.

(d) Boundary vertex \(v_p(i)\) is concave, boundary vertex \(v_p(i+1)\) is also concave.

Fig. 4. Boundary edges and their active area.

Fig. 5. Single boundary and multiple active area.

3) Multiple Boundaries and Single Active Area: (Fig. 6) Recall that boundary vertices are specified in a counterclockwise order so that a sense of what is inside or outside the boundary can be established. In Fig. 6, boundary \(B_2\) is fully inside boundary \(B_1\). The active area \(A = B_1 - B_2\), is a single connected active area. Each of the boundary edges will exert a boundary constraint force in the direction described in Section IV-C. A graph to be laid out, whether it is a single connected graph or a forest, will be constrained to fully reside within the active area \(A\).

4) Multiple Boundaries and Multiple Active Area: Multiple active areas can arise from certain arrangements of multiple boundaries. If boundaries are nested with alternating inside-outside orientations as in Fig. 7, then one can obtain multiple disjoint active areas. For such cases, the same behavior as illustrated in Fig. 5 can be expected. That is, the final layout is sensitive to the initial layout of the graph. Also, if the graph is a single connected graph, it will end up in one of

to them if they are in the active area of each of the boundary edges of the boundary. These forces are summed up to obtain the net boundary constraint forces acting on a graph element. The resulting boundary constraint force is then factored in together with other force-directed components i.e. spring and gravitation forces, to effect a change in the position of the graph element.

2) Single Boundary and Multiple Active Area: (Fig. 5) This scenario happens when boundary edges intersect each other. As an example, boundary edge \(e_p(i - 1)\) and boundary edge \(e_p(i + 1)\) intersect each other at \(v_p\). This results in two separate active areas \(A_1\) and \(A_2\), which we treat as two single active areas. In this case, the resulting layout of a graph will be highly dependent on the initial configuration or position of the graph elements. If the graph to be laid out is a single connected graph, i.e. all the nodes are connected together, then the final layout of the graph will be constrained to be in either \(A_1\) or \(A_2\). If the graph to be laid out contains multiple disjoint components i.e. the graph is actually a forest, then different parts of the forest will end up in \(A_1\) or \(A_2\) depending on their initial positions prior to activating the boundary constraint forces.
the active areas; and if it is a forest, then different parts will go to different active areas.

Aside from this scenario, there are other ways to obtain multiple active areas using multiple boundaries. For example, one of the enclosed boundaries may be a self-intersecting boundary, or two of the enclosed boundaries intersect each other. We do not consider these cases in this paper.

C. Boundary Forces

Before we assign boundary force to each graph vertex we have to know if the graph vertex is within the boundary. This test can be carried out using a point in polygon test such as the one described by Franklin [10]. This is an efficient \(O(m)\) test that depends on the number of edges defining a boundary, and can handle self-intersecting polygons properly.

If the graph vertex \(v_G(j)\) is within the boundary \(B_p\), then we start testing if it is within each active area of boundary edges. Take Fig. 8 as an example, we have a convex boundary vertex \(v_p(i)\), and a concave boundary vertex \(v_p(i + 1)\). First we consider boundary edge \(e_p(i)\) and its active area \(A_i\) bounded by its left vector \(-e_p(i - 1)\), boundary edge \(e_p(i)\) and vector \(v_1 p(i + 1)\). We construct two new vectors \(\text{vec}1(i) = v_G(j) - v_p(i)\) and \(\text{vec}2(i) = v_G(j) - v_p(i + 1)\), if both of them fall within active area \(A_i\), i.e. \(\text{vec}1(i)\) is always between left vector of active area \(A_i\) and boundary edge \(e_p(i)\), and \(\text{vec}2(j)\) is always between right vector of active area \(A_i\) and vector \(v_p(i + 1) v_p(i)\), then the graph vertex \(v_G(j)\) is considered within the active area \(A_i\). So in Fig. 8, \(v_G(1)\) is considered within active area \(A_i\), but \(v_G(2)\) is not. Let the distance from \(v_G(j)\) to boundary edge \(e_p(i)\) be \(\text{de}(i)\) and a vector \(\text{ver}(i)\) be perpendicular to boundary edge \(e_p(i)\) and pointing towards the interior of the boundaries. That is,

\[
\text{de}(i) = \frac{\text{vec}1(i) \cdot \text{ver}(i)}{||\text{ver}(i)||}
\]

Then the boundary force \(f_Bp\) acting on the graph vertex \(v_G(j)\) from boundary edge \(e_p(i)\) is defined as:

\[
f_Bp(v_G(j), e_p(i)) = c_5 \cdot \frac{1}{\text{de}(i)} \cdot \frac{\text{ver}(i)}{||\text{ver}(i)||}
\]

where \(c_5\) is scaling constant parameter.

Now we consider the triangular active area bounded by vector \(v_p(i + 1) v_1 p(i + 1)\) vector \(v_p(i + 1) v_2 p(i + 1)\) in Fig. 8. If the graph vertex \(v_G(j)\) falls in this active area, we replace \(\text{de}(i)\) with distance from the graph vertex \(v_G(j)\) to the nearest boundary vertex \(v_p(i + 1)\), using:

\[
\text{de}(i) = ||v_p(i + 1) v_G(j)||
\]
e_p(i - 1) from v_p(i) to v_p(i - 1), the active area consists of A_i and the triangular gap between A_{i-1} and A_i.

Note that outside graph vertices that are further away will have stronger boundary force acting on them, so the outside boundary force f_{Bp} is defined as:

\[ f_{Bp}(v_G(j), e_p(i)) = c_5 \cdot \frac{\text{de}(i) \cdot \text{ver}(i)}{|\text{ver}(i)|} \]  

where c_5 is scaling constant parameter. By iterating through all the boundary edges, the total boundary forces for vertex v_G(j) from q boundary edges will be \( \sum_{i=1}^{n} f_{Bp}(v_G(j)) \). By iterating through all the boundary edges in each boundary B_p, the total boundary force for vertex v_G(j) from q boundary edges will be \( \sum_{p=1}^{q} \sum_{i=1}^{n} f_{Bp}(v_G(j), e_p(i)) \).

D. Modified Force Components

Here we modify the conventional spring and repulsive forces in order to combine with our boundary forces. We utilize the knowledge of the size of the graph and the size of the active area to scale the forces appropriately to achieve a uniform distribution of graph vertices.

1) Spring Force: Given a graph G, each vertex is placed in some initial random layout with coordinates P_i(x, y). Once released, the spring forces act to move the system to a minimal energy state. We use the logarithmic strength springs and modify Equation (1) to:

\[ f_s(v_G(i), v_G(j)) = c_1 \cdot \frac{\alpha}{\beta} \cdot \log \left( \frac{|v_G(i)v_G(j)|}{\text{deg}(i)\text{deg}(j)} \right) \]  

where \( \alpha \) is the number of vertices in the graph. \( \beta \) is the total active area. Together \( \alpha/\beta \) represents the average density of vertices within the active area. This modification of Equation (1) allows the attractive spring forces to scale with the layout density.

2) Repulsive Force: Equation (2) is modified in a similar manner to take into account graph density:

\[ f_r(v_G(i), v_G(j)) = c_3 \cdot \frac{\beta}{\alpha} \cdot \frac{v_G(i)v_G(j)}{|v_G(i)v_G(j)|^3} \]  

This time the force is inversely related to the density \( \alpha/\beta \).

3) Boundary Force: Equations (4) and (6) are also modified in a similar manner to take into account graph density. For graph vertices that are within the boundary:

\[ f_{Bp}(v_G(j), e_p(i)) = c_5 \cdot \frac{\alpha}{\beta} \cdot \frac{\text{de}(i) \cdot \text{ver}(i)}{|\text{ver}(i)|} \]  

And for graph vertices that are outside the boundary:

\[ f_{Bp}(v_G(j), e_p(i)) = c_5 \cdot \frac{\alpha}{\beta} \cdot \frac{\text{de}(i) \cdot \text{ver}(i)}{|\text{ver}(i)|} \]  

E. Graph Drawing Algorithm

Our drawing algorithm includes three main parts: (1) compute the attractive spring force of each graph edge and the repulsive gravitational force for each pair of graph vertices; (2) compute boundary forces for each vertex; then (3) add the three different kinds of forces together. At each iteration, make a step towards the direction where the total force is pointing for each vertex, and draw the updated graph.

V. IMPLEMENTATION AND TESTING TOOLS

A. Implementation

We developed a prototype using Matlab and its graphic library. Tests were ran on an Intel Core i5 3GHz PC with 8GB of memory running Windows 7. Processing 2.0 was used to realize the interactive part with users.

B. Synthetic Graphs

For testing purposes, we created a synthetic graph generation program. Input to this program is the number of graph vertices and edges. The vertices are assigned an initial random position, while pairs of vertices are connected randomly using uniform sampling of the vertices with replacement.

In Fig. 10, we show the initial random layout of a graph which has 13 graph vertices and 5 boundary vertices. Coordinates of the graph vertices are randomly generated while the boundary is user-defined. Red boundary vertices are connected to form the boundary. Blue graph vertices scattered randomly have black arrows representing the total force and direction acting on them after each iteration. After 150 iterations, we can see the graph is indeed totally within the predefined boundary in Fig. 11.
VI. RESULTS AND ANALYSIS

In this section, we apply our boundary constraints to several graph datasets using both synthetic graphs and publicly available graph datasets. First we demonstrate how the graph layout changes with different boundary constraints. Then we ran it on different scales of graph data. Lastly, we show some visual results of arbitrarily shaped boundaries and dynamic response to altering the boundary during the graph layout process.

A. Experimental Results for Different Boundary Force Functions

We know the boundary forces depends on the distance from the graph vertex to the boundary edge. Here, we demonstrate the effects of changing the boundary force functions on the graph layout. Equation (9) specifies an inverse distance relationship of boundary force on a graph vertex. Fig. 12 illustrates how the positions of the graph vertices are affected by changing the boundary force functions without changing the boundary constraints. Note that graph edges are not drawn in these illustrations and convergence times vary as well. In this example, we are using the same graph with 2000 vertices and 4000 edges under the same set of parameters and the same boundary of a regular pentagon.

In Fig. 12(a), we are using inverse of \( d \). By changing it to inverse of the logarithm of \( d \), the vertices are pushed further away from the middle part of the boundary edges resulting in curved silhouettes as shown in Fig. 12(b). Since boundary force of inverse of \( d^2 \) is dropping much faster than inverse of \( d \), we see graph vertices are closer to the boundary as shown Fig. 12(c). And since the boundary force of inverse of \( \sqrt{d} \) is dropping slower than inverse of \( d \) we see the graph is further pushed further away from the boundary in Fig. 12(d).

B. Experimental Results for Different Scales of Graphs

Similar to conventional force-directed methods as discussed in Section 2, the complexity of our approach depends on number of vertices and edges in the graph [5]. Because the number of boundary vertices are much less than the number of vertices in the graph, the running time of our approach remains at the same level. We ran some experiments to obtain some actual running times. First, we fixed the ratio of vertices to edges in the graph, then increase the number of vertices and number of edges proportionately. The running times are listed in Table 1 and the resultant layouts are shown in Fig. 13. Note that we are using the same topology of boundary as a regular pentagon and inverse distance to the boundary edge as the boundary force function for these experiments. We also hide the edges of each graph in order to have a more clear view of the distribution of vertices.

<table>
<thead>
<tr>
<th>Number of vertices</th>
<th>Number of edges</th>
<th>Average Running Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>200</td>
<td>0.87</td>
</tr>
<tr>
<td>200</td>
<td>400</td>
<td>3.56</td>
</tr>
<tr>
<td>1000</td>
<td>20000</td>
<td>53.91</td>
</tr>
</tbody>
</table>

Table 1: Running Time of Datasets that have Same Ratio of Vertices and Edges

We can see that the graph vertices are distributed evenly within each boundary. With increasing density of vertices, the graph can reveal the shape of the boundary more clearly.

C. Visual Results for Boundaries with Different Topology

In this section, we define several convex and concave boundary shapes. The visual results are listed in Fig. 14. We also defined multiple boundaries with layouts shown in Fig. 15. Fig. 15(a) with one interior boundary and Fig. 15(b) with two interior boundaries.

Animation of both changing boundaries and graph layout process are also an integral part of the visual feedback for the users. Allowing users to adjust and manipulate boundary vertices where the graph is to be constrained can be very helpful. Here, we took several screen shots of the graph layout process with changing boundary shape. First we used a Facebook dataset of 1589 vertices and 2732 edges and changed the layout from an initial square boundary constraint to a triangular boundary constraint (see Fig. 16). In Fig. 17, a graph inside a circular sun shape is rearranged to conform to a moon shaped boundary. Another example illustrated in...
Fig. 13. Layout of datasets with have same ratio of vertices to edges.

Fig. 14. Graph layout with different boundary shapes.

Fig. 15. Different topologies with multiple boundaries.

Fig. 16. Layouts from a Facebook user dataset.

Fig. 18 simulates graph vertices spreading out to fill an hour glass shape. At first, all the vertices in graph are at the top, as the algorithm runs, they expand and spread out over this hour glass shape. Note that there are no downwards gravitational forces modeled into the simulation.

VII. CONCLUSION AND FUTURE WORK

This paper presented a novel way for manipulating and specifying graph layout with the use of boundary constraints. This is incorporated within a force-directed simulation and does
During process

Final layout

not significantly increase the cost of graph layout. The force-directed simulator is self-contained and can be substituted with a more efficient implementation. The boundary constraints are quite general and can support arbitrary shapes including self-intersections and boundaries with different topologies.

The current work focuses on constraining graph vertices to lie within specified boundaries. No consideration is made for constraining graph edges to lie within boundaries as well. A possible extension is to constrain graph edges to lie within boundaries as well.

REFERENCES