UNIVERSITY OF CALIFORNIA

SANTA CRUZ

BOUNDARY CONSTRAINTS IN FORCE-DIRECTED GRAPH LAYOUT

A thesis submitted in partial satisfaction
of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

YANI ZHANG

June 2014

The Thesis of YANI ZHANG
is approved:

Table of Contents

List of Figures

List of Tables

Abstract

Boundary Constraints in Force-Directed Graph Layout

by

Yani Zhang

This paper focuses on graph layouts with constraints using force-directed simulations. Existing graph drawings with constraints include placement of a particular vertex or a group of vertices at a specified location, constraining placement of vertices and edges to specified rows and/or columns. We propose an alternative way of specifying constraints by allowing the user to interactively draw a boundary wherein the graph layout will be constrained. Such boundary constraints may be saved and applied to other graphs as well. In addition, the boundary may be of different topology such as a donut shape, or figure-eight shape, etc. We model these boundaries as a set of additional forces that contribute to the forces acting on graph vertices. Because our proposed approach is force-directed, it can take advantage of optimizations of other force-directed graph layout algorithms. Furthermore, one can utilize the knowledge of the size of the graph to be visualized and the size of the interior of the boundary region to scale the forces appropriately to achieve a uniform distribution of vertices. We tested this idea on several data sets and different boundary constraints.

Acknowledgments

I wish to extend my deepest gratitude to my advisor Professor Alex Pang for his encouragement, guidance and patience during the course of this project. I would like to thank Professor Suresh Kumar Lodha and Professor Roberto Manduchi for their feedback, advice, and time that have gone into this thesis. Finally, I am indebted to many people who helped me in finishing this thesis. I profited immensely from comments and reviews from my advisor and colleagues, and constant encouragement from my family.

1 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 [1–5], force-directed approach [6–18] is the most well-known method for drawing undirected graphs due to their flexibility of adding constraints, ease of implementation and relatively resultant layouts compared to other approaches [19]. Specifically, a graph is viewed as a neat drawing of the particles can be generated with by minimizing an energy function [4].

Based on this force-directed idea, many practical graph drawing systems have been developed [19–21]. One of them is a popular JavaScript library D3.js [22]. It simulates force-directed graphs using a repulsive charge force to separate vertices apart from each other and a pseudo-gravity force to hold the entire graph together in the visible area 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, to obtain the desired graph with a minimum amount of time. With these goals in mind, various kinds of constraints were taken into account in the graph layout problems [23–26]. 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 [24]. 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. With these boundary constraints, the layout of the graph will be updated such that it can be fitted into the user-defined boundaries. 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 2. Section 3 provides a basic background on force-directed methods and the definition of our graph drawing problem. In section 4 we give a formal definition of boundary constraints and present our new force-directed model with boundary constraints. Implementation work is discribed in section 5 and boundary constrained layout of graphs and analysis of results are presented in section 6. We provide conclusions and identify some avenues for future work in section 7.

2 Related Work

2.1 Interactivity

Existing constraint-based systems have been implemented to achieve the goal of interactively manipulating graph layout, but none of them include outside environment force as additional constraints to the graph drawings. For example, the GLIDE (Graph Layout Interactive Diagram Editor) system [27] 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) [28] 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 [29] (also known as node-link diagrams and circle-and-arrow diagrams), 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.

2.2 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 [11]. Regions were rectangular in shape and were represented as inequality constraints wherein graph vertices must lie. No forces were used in their formulation. In order to allow users to interactively define different kinds of constraints, a constrained graph layout model [23] which is an extension of the force-directed model was built. Similar to force-directed methods, these techniques find a layout minimizing a goal function subject to placement constraints on the vertices instead of force constraints.

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 [30]. Using this formalism, one can express several simple geometric constraints among the vertices, such as horizontal or vertical alignment, and circular arrangements. Dengler [29] 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 and is the most widely used approach at present.

A comprehensive approach to constrained graph drawing is presented by He and

Marriot [23]. 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 [14] 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 we can reduce the complexity of general the constrained force-directed graph models in each iteration, our boundary constraints produce forces that are processed and converge at the same rate as currently modelled forces. On the other hand, with the other methods mentioned above, m ore 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.

3 Force-directed Graph Layout

In this section, we first provide formal definitions of concepts that will be used later to define the graph drawing problem at hand, as well as some necessary background in force-directed methods.

3.1 Definitions

3.1.1 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 $E_G \subset V_G \times V_G$. 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.

3.1.2 Boundary

Similarly, a boundary B_p of the graph is defined also as a pair (V_{Bp}, E_{Bp}). Suppose boundary B_p has n vertices (which we will refer as boundary vertices to distinguish them from graph vertices) and m edges (which we will refer as boundary edges to distinguish them from graph edges), then V_{Bp} is defined as a sequence of boundary vertices $V_{Bp} = \{v_p(1), v_p(2), \cdots, v_p(n)\}$ and E_{Bp} is defined as a set of boundary edges that connects adjacent vertices and forms a connected path. That is, $E_{Bp} = \{e_p(1), e_p(2), \cdots, e_p(n)\}$ where $e_p(i) = \overline{v_p(i)v_p(i+1)}$ $(i = 1, 2, \cdots, m-1)$ and for special case $i = m$, $e_p(m) = \overline{v_p(m)v_p(1)}$. 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.

3.2 Common Forces

Now we introduce some common forces in classical force-directed methods. The graph drawing algorithm of Tutte [31] is 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 [7] 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. For *adjacent* vertices (vertices that connect each other with 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:

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

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

For *non-adjacent* vertices (vertices that are not connected 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:

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

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.

3.3 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 $\{B_1, B_2, \cdots, B_q\}$ to specify q distinct boundaries. The graph drawing algorithm is responsible for finding an assignment to variables representing the vertex coordinates that satisfies the boundary constraints and gives a good layout of the graph using a force-directed approach. More precisely, the layout algorithm should solve the optimization problem and satisfy the following criteria:

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

4 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.

4.1 Definition of Active Area

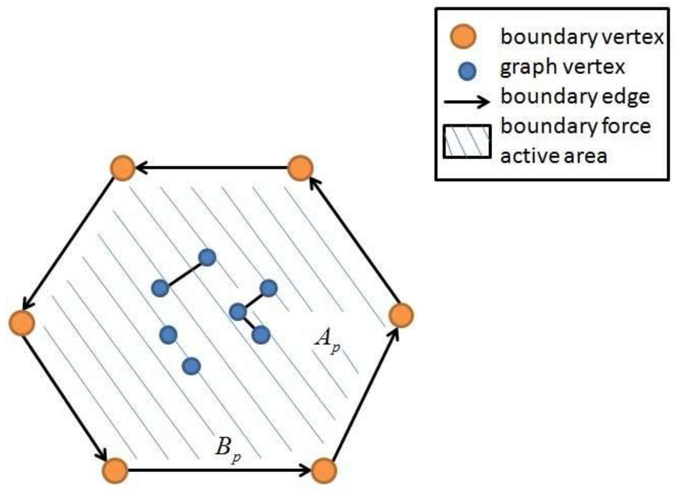
Boundary constraints are enforced via boundary forces on graph elements. In order to determine how these boundary forces affect graph elements, we must determine the set of boundary edges that can influence an individual graph element, or conversely, the set of graph elements affected by a boundary force.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

4.1.1 Active Area of A Set of Boundaries

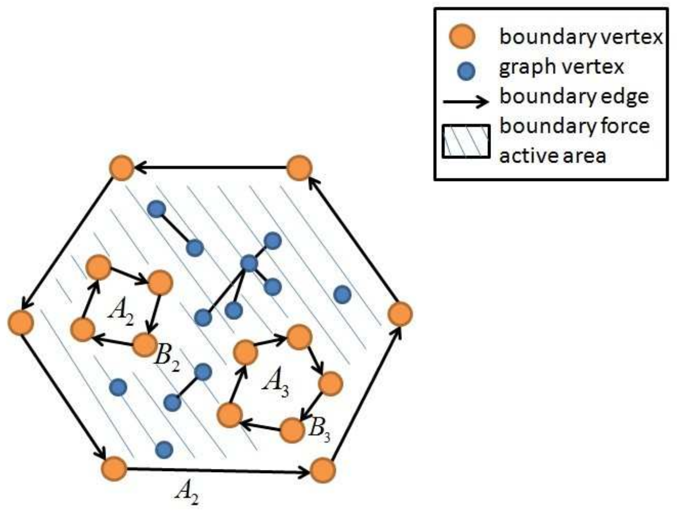
A boundary specifies a partitioning of the space wherein a graph is to be drawn. For closed boundaries, we need to distinguish between the inside and outside of the boundary. By convention, we will assume that boundary vertices are ordered in a counter-clockwise manner so that the inside is to the left of an 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). Thus, the active area in that figure is $A = B_1 - (B_2 + B_3)$. In general, assuming that boundaries do not intersect each other, that is, we assume that there is only a single connected area whose outer perimeter is specified by B_1, that $B_2 \cdots B_q$ do not intersect each other, and do not contain another boundary within each one. 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.

4.1.2 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 inside the boundary. 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 That means, if a graph vertex $v_G(j)$ is within the active area of boundary edge $e_p(i)$ in boundary B_p, then we assign a boundary force $f_{Bp}(i)$ to those vertices with a direction perpendicular to the boundary edge $e_p(i)$ and pointing towards the interior of the boundary. Take Fig. 2 as an example. Assume boundary edge $\overline{v_p(1)v_p(2)}$ has the active area indicated by dotted lines, so for all vertices in graph that are within that area, a boundary force $f_{Bp}(1)$ is assigned with direction that is perpendicular to boundary edge $\overline{v_p(1)v_p(2)}$ and pointing to the interior of the boundary in order to push the graph inside. To define active area of boundary edges and assign boundary force for each vertex within the active area, we need to consider three different cases. In general, a boundary edge will exert an inward force perpendicular to the boundary edge on graph elements in order to keep them inside the boundary. 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 by a vector from boundary vertex $v_p(i)$(which we call left vector), the segment $e_p(i)$, and another vector from boundary vertex $v_p(i+1)$ (which we call right vector) as indicated in Fig. 2. There are four graph vertices within the active area of $e_p(i)$, then we assign a boundary force $f_{Bp}(v_G(j), e_p(i))$ (see Section 4.3) to each one of them 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



(a) One boundary case.



(b) Multiple boundaries case.

Figure 1: Definition of active area of boundaries.

edge. That is, whether the boundary vertex is concave or convex.

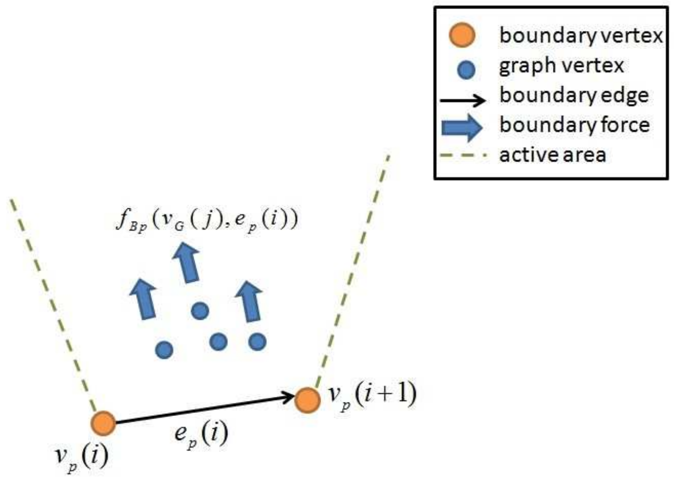


Figure 2: Definition of active area of boundary edges.

Consider 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. $v_p(i+1)$.

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

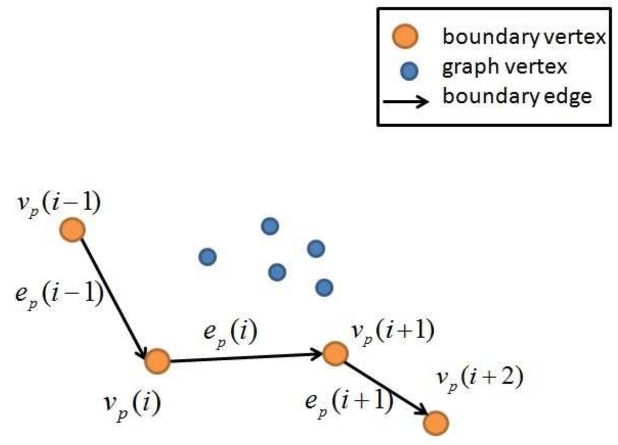


Figure 3: Convex and concave boundary vertices.

edge $e_p(i)$ does not contribute a force to the graph element. In this way, we do not have to divide the active area into different regions while maintaining a uniform distribution of the boundary forces. We tested this method and show our results in section 6.

Case 2: $v_p(i)$ is convex and $v_p(i+1)$ is concave (see Fig. 4(b)):
For this case, 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 $v2_p(i+1)$ as the right vector. $v1_p(i+1)$ is perpendicular to $e_p(i)$ and $v2_p(i+1)$ 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 $\overline{v_p(i+1)v1_p(i+1)}$, we apply a force perpendicular to boundary edge $e_p(i)$. For graph elements falling in the triangular gap bounded by vector $\overline{v_p(i+1), v1_p(i+1)}$, and right vector $\overline{v_p(i+1), v2_p(i+1)}$, 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)):
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. For case 3, we define the active area of boundary edge $e_p(i)$ to be bounded by a vector from $v_p(i)$ to $v2_p(i)$ as the left vector, boundary edge $e_p(i)$, and boundary edge $e_p(i+1)$ as the right vector. Graph elements in A_i are applied a force perpendicular to boundary edge $e_p(i)$. For graph elements in the triangular gap bounded by vector $\overline{v_p(i)v1_p(i)}$, and left vector $\overline{v_p(i), v2_p(i)}$, we apply a force in the direction from $v_p(i)$ to the graph element.

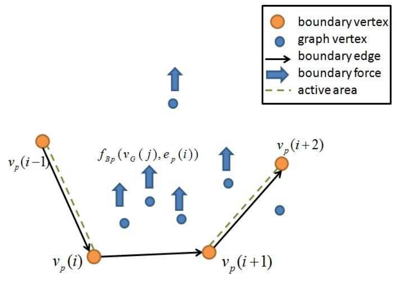
Case 4: Both $v_p(i)$ and $v_p(i+1)$ are concave (see Fig. 4(d)):
For this case, we define the active area of boundary edge $e_p(i)$ to be bounded by a vector from $v_p(i)$ to $v2_p(i)$ as left vector, boundary edge $e_p(i)$, and a vector from $v_p(i{+}1)$ to $v2_p(i{+}1)$ as right vector. Just as in 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.

4.2 Different Types of Active Area

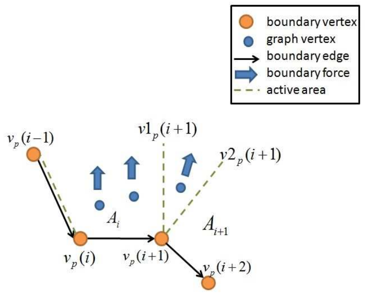
Here, we consider the number of active areas resulting from either a single or multiple boundaries, and whether their shape or arrangement result in a single or multiple active areas where graph elements will be constrained. There are four possible configurations which we discuss below.

4.2.1 Single Boundary and Single Active Area

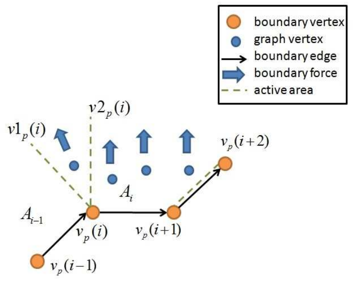
See in Fig. 1(a), This is the simplest case, where the boundary constraint is specified by a single boundary and where the edges in this boundary do not self-intersect. Graph elements will have forces applies to them if they are in the active area of each of the boundary edges of the boundary according to the rules set in Section 4.1.2. These forces are summed up to obtain the net boundary constraint forces acting on a



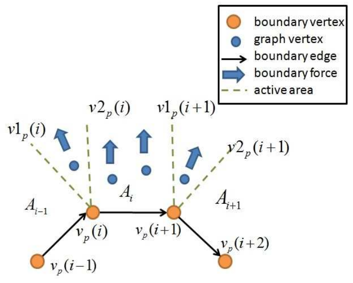
(a) Boundary vertex $v_p(i)$ is convex, boundary vertex $v_p(i+1)$ is also convex.



(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.

Figure 4: Boundary edges and their active area.

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.

4.2.2 Single Boundary and Multiple Active Area

See in 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 lay out 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.

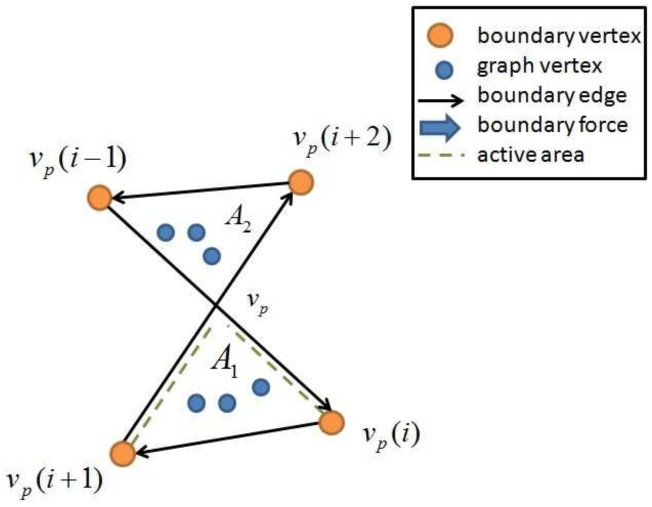


Figure 5: Single boundary and multiple active area.

4.2.3 Multiple Boundaries and Single Active Area

As in Fig. 6,recall that boundary vertices are specified in a counter-clockwise order so that a sense of what is inside or outside the boundary can be established. In the example 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 4.1.2, consist with the inside versus outside of a boundary edge. 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.

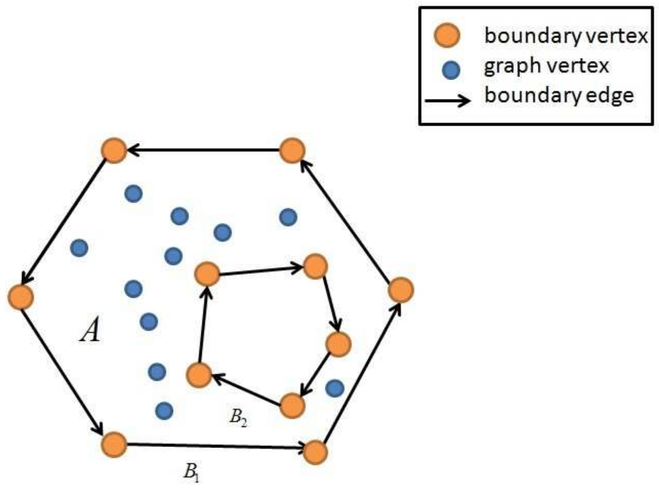


Figure 6: Multiple boundaries and single active area.

4.2.4 Multiple Boundaries and Multiple Active Area

Multiple active areas can arise from certain arrangements of multiple boundaries. For example, 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 behaviour as the described in section 4.2.2 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 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 tows of the enclosed boundaries intersect each other. We do not consider these cases in this paper.

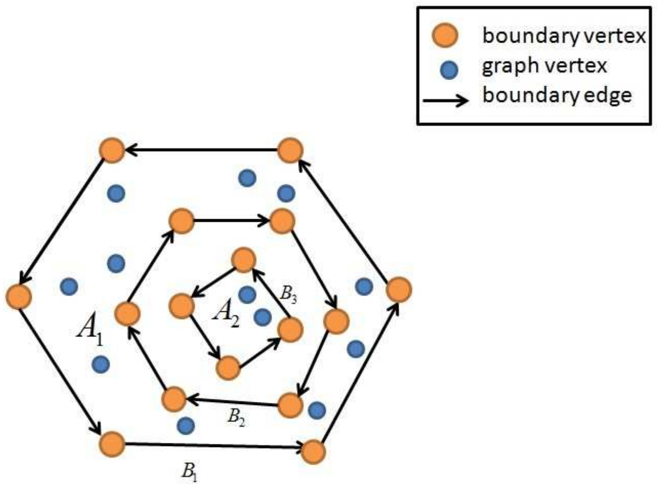


Figure 7: Multiple boundaries and multiple active area.

4.3 Boundary Forces

Now we know when to apply boundary force on each graph vertex that is when it falls into active area of each boundary edge. Here we discuss how to find out if the graph vertex is within active area and how to calculate the boundary forces acted on it.

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 [32]. 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. According to the type of boundary vertices as convex or concave, we give different formulas to calculate boundary forces. Assume we have n boundary vertices and m graph vertices. 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 $v1_p(i+1)$. We construct two new vectors $\overline{vec1(i)} = v_G(j) - v_p(i)$ and $\overline{vec2(i)} = v_G(j) - v_p(i+1)$, if both of them fall within active area A_i, i.e. $\overline{vec1(i)}$ is always between left vector of active area A_i and boundary edge $e_p(i)$, and $\overline{vec2(j)}$ is always between right vector of active area A_i and vector $\overline{v_p(i+1)v_p(i)}$, 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. Once we know the graph vertex $v_G(j)$ is within active area A_i, assume that the distance from $v_G(j)$ to boundary edge $e_p(i)$ is $de(i)$, and a vector $\overline{v_{pv}(i)}$ is perpendicular to boundary edge $e_p(i)$ and pointing the interior of boundaries, so we have:

$$de(i) = \frac{\overline{vec1(i)} \cdot \overline{v_{pv}(i)}}{|\overline{v_{pv}(i)}|} \tag{3}$$

Then the boundary force f_{Bp} acted on this graph vertex $v_G(j)$ from boundary edge $e_p(i)$ is:

$$\begin{gathered} f_{Bp}(v_G(j), e_p(i)) = c_5 \cdot \frac{1}{de(i)} \cdot \frac{\overline{v_{pv}(i)}}{|\overline{v_{pv}(i)}|} \\ i = 1, 2, \cdots, n, j = 1, 2, \cdots, m. \end{gathered} \tag{4}$$

where c_5 is scaling constant parameter.

Now we consider the active area bounded by vector $v1_p(i+1)$ and vector $v2_p(i+1)$ in Fig. 8. If the graph vertex $v_G(j)$ falls in this active area, we replace $de(i)$ with

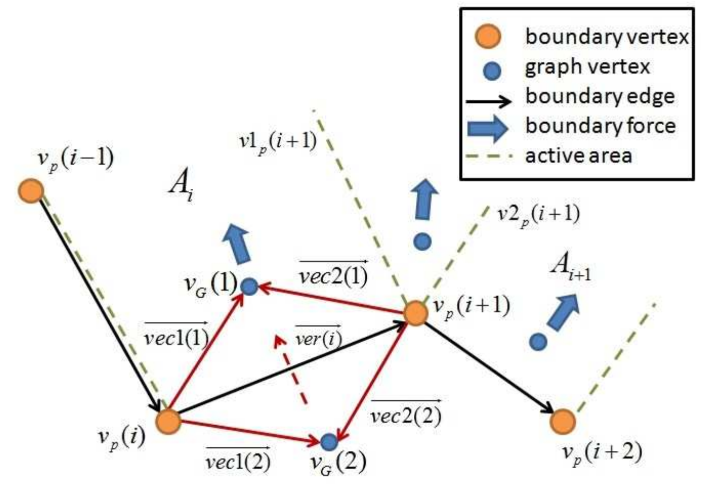


Figure 8: Inside boundary forces on graph vertices.

distance from the graph vertex $v_G(j)$ to the nearest boundary vertex $v_p(i+1)$, as in:

$$de(i) = |\overline{v_p(i+1)v_G(j)}| \tag{5}$$

And plug it into Equation (4) as their boundary forces. For boundary edge $e_p(i+1)$, we use the same method to calculate its boundary force as with boundary edge $e_p(i)$. For the graph vertices that are not within the boundary, we assign a force towards interior of the boundary and pull them inside. The mechanism to assign outside boundary forces within each boundary edges active area is exactly the same as discussed in previous section. It can be considered the same with multiple boundary cases where the interior boundaries have outside active areas. The only difference is that here the direction of boundary forces is from the graph vertex and pointing towards interior of boundary but in interior boundaries, the direction is the opposite. Thus for each boundary edge, we have an inside active area and an outside active area as illustrated in Fig. 9. For convex boundary vertex $v_p(i-1)$, boundary edge

$e_p(i-1)$ has an outside active area A_{i-1} and for concave boundary vertex $v_p(i)$, boundary edge $e_p(i)$ has two active area taken into consideration, that is active area A_i and the gap between A_{i-1} and A_i.

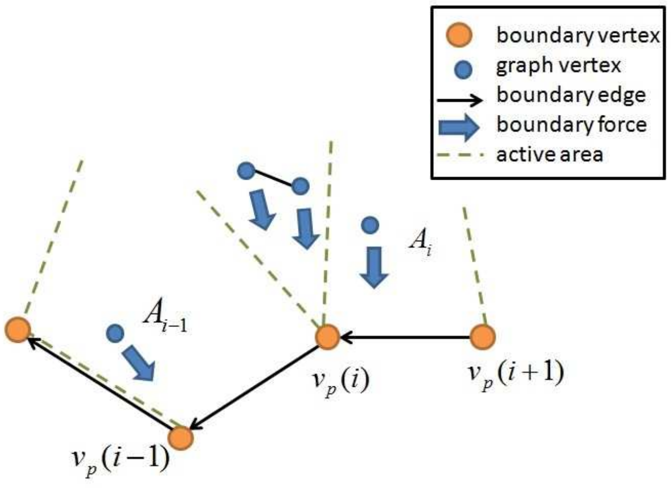


Figure 9: Outside boundary forces on graph vertices.

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

$$f_{Bp}(v_G(j), e_p(i)) = c_5 \cdot de(i) \cdot \frac{\overline{v_{pv}(i)}}{|\overline{v_{pv}(i)}|} \qquad (6)$$
$$i = 1, 2, \cdots, n, j = 1, 2, \cdots, m.$$

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 $\sum_{p=1}^{q} f_{Bp}(v_G(j))$. By iterating through all the boundary edges in each boundary B_p, the total boundary forces for vertex $v_G(j)$ from q boundary edges will be $\sum_{p=1}^{q}\sum_{i=1}^{n} f_{Bp}(v_G(j), e_p(i))$, as shown in Fig. 10, the total boundary force for vertices $v_G(j)(j = 1, 2, 3)$ will be the total force of $\sum_{i=1}^{3} f_{Bp}(v_G(j), e_p(i))(p = 1)$.

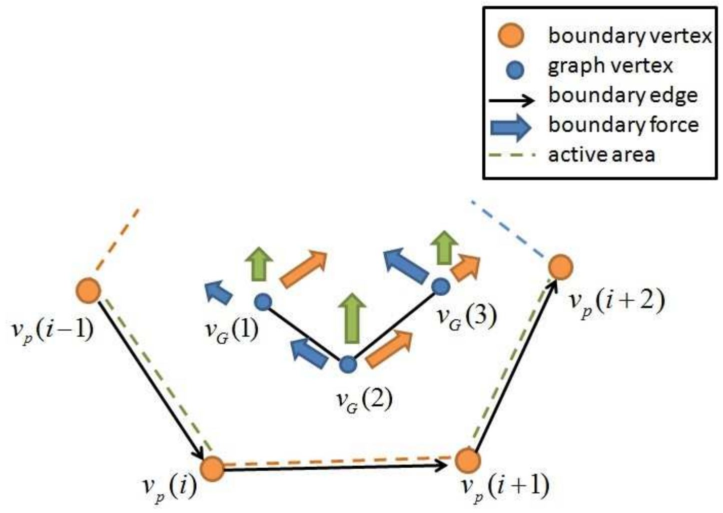


Figure 10: The total boundary force for graph vertices.

4.4 Modified Force Components

Here we modified conventional spring force and repulsive force in order to combine with our boundary force. We utilize the knowledge of the size of the graph to be visualized and the size of the interior of the region to scale the forces appropriately to achieve a uniform distribution of graph vertices.

4.4.1 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 the force exerted by a spring is:

$$f_s(v_G(i), v_G(j)) = \left(c_1 \cdot \frac{\alpha}{\beta} \cdot \log\left(\frac{|\overline{v_G(i)v_G(j)}|}{c_2}\right)\right) \frac{\overline{v_G(i)v_G(j)}}{|\overline{v_G(i)v_G(j)}|} \qquad (7)$$
$$i, j = 1, 2, \cdots, n$$

where c_1 is scaling constant for spring force, c_2 is the given spring natural length. α is for the size of the graph, depends on the number of the vertices in the graph.

β is the total area of the active area. Together α/β represents density of the graph drawing. This modification of Equation (1) allows the attractive force to scale with the layout density.

4.4.2 Repulsive Force

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

$$f_r(v_G(i), v_G(j)) = c_3 \cdot \frac{\beta}{\alpha} \cdot \frac{1}{|\overline{v_G(i)v_G(j)}|} \frac{\overline{v_G(i)v_G(j)}}{|\overline{v_G(i)v_G(j)}|} \qquad i,j = 1,2,\cdots,n \tag{8}$$

where c_3 is scaling constant for the repulsive force as before. This time the force is inversely related to the density α/β.

4.4.3 Boundary Force

Equation (4) and (6) of boundary force is also modified in a similar manner to take into account graph density. For graph vertices that are within the boundary:

$$f_{Bp}(v_G(j), e_p(i)) = c_5 \cdot \frac{\alpha}{\beta} \cdot \frac{1}{de(i)} \cdot \frac{\overline{v_{pv}(i)}}{|\overline{v_{pv}(i)}|} \tag{9}$$

And for graph vertices that are outside the boundary:

$$f_{Bp}(v_G(j), e_p(i)) = c_5 \cdot \frac{\alpha}{\beta} \cdot de(i) \cdot \frac{\overline{v_{pv}(i)}}{|\overline{v_{pv}(i)}|} \qquad i = 1,2,\cdots,n, j = 1,2,\cdots,m. \tag{10}$$

where c_5 is scaling constant for the boundary force and α/β is the spatial density of graph drawings as before.

4.5 Graph Drawing Algorithm

Our drawing algorithm is outlined in Algorithm I.It includes three main parts: 1). compute the spring force of each edge and compute the repulsive forces due to each pair of neighboring edges incident to the vertex; 2). compute boundary forces for each vertex, then 3). add the three different kinds of forces together. In the end, make a step towards the direction where the total force is pointing for each vertex, and draw the updated graph according to certain set of parameters.

In Algorithm I, parameter c_4 are used to control the step of movement of every vertex. Recall that parameters c_1 and c_2 in Equation (7) (logarithmic spring force formula) are used to control the force magnitudes and natural lengths of springs, respectively; parameter c_3 in Equation (8) (the magnitude of repulsive force between each two vertices) is used to control the scale of the repulsive force magnitude; parameter c_5 in Equation (9) and (10) (the magnitude of boundary force) is used to control the scale of the boundary force magnitude. Note that the algorithm can reach convergence if parameters $c_1 - c_5$ and ϵ are set appropriately.

5 Implementation

5.1 Tools

Based on the formulas and algorithm detailed in the previous sections, we develop a prototype using Matlab and its graphic library. Tests were ran on an Intel Core $i53.00$ GHz PC with memory of size 8.00 GB running Windows 7. Processing 2.0 was used to realize the interactive part with users. In this section, we will detail the implementation of our algorithm, and show some promising experimental results with Matlab.

The analysis on the convergence and the adjustments of parameters in force-directed methods has been discussed intensively in previous works (e.g., see [8, 13]). On theoretical aspect, Eades and Lin [33] have shown that the general framework of

Algorithm 1: Boundary Constraints (Adding boundary constraints in force-directed graph layout of graph)

```
Input  : A randomly drawing of $G = (V_G, E_G)$ and user-defined boundary
         set $B = \{B_p\}$
Output: A nice graph drawing within the boundaries
begin
    assign initial coordinates of vertices in $V_G$;
    while the maximum iteration number is achieved do
        converged $\longleftarrow$ false;
        oldPos $\longleftarrow$ newPos;
        initialize springForce[$|V_G|$] as zeros matrix;
        initialize repelForce[$|V_G|$] as zeros matrix;
        initialize boundForce[$|V_G|$] as zeros matrix;
        for each vertex $v_G(i)$ in $V_G$ do
            springForce[i] $\longleftarrow$ springForce[i] + $f_s$ (calculate and update its
            springForce[i]);
            repellForce[i] $\longleftarrow$ repelForce[i] + $f_r$ (calculate and update its
            repelForce[i]);
            if the vertex $v_G(i)$ is within the active area of each boundary edge
            then
                for each boundary vector $\overline{v_p(i)v_p(i+1)}$ do
                    calculate its boundary force $fb$ to this vertex and update
                    boundForce[i];
                end
            end
        end
        totalForce[vi] $\longleftarrow$ springForce[i] + repelForce[i] + boundForce[i];
        calculate the step for each vertex and its new position in next
        iteration;
    end
    if $||newPosnoldPosn|| > \epsilon$ then
        converged $\longleftarrow$ 0
    end
end
```

force-directed methods can lead to a stable drawing in which many symmetries are displayed. Similarly, our force-directed approach also can be shown to be stable, under appropriate setting of parameters.

In Algorithm I, it should be noted that the setting of parameters c_1c_5 not only influence the run times but also the convergence rate of our approach and the quality of the final drawing. In the following, we briefly explain how to set those parameters in Algorithm I to achieve convergence. Recall that c_1 is the scaling constant for spring force,c_2 is the given spring natural length, c_3 is the scaling constant for the repelling force, c_4 is used to control the step size of movement of every vertex, and c_5 is the scaling constant for the boundary force. Parameters c_1, c_2, c_3, and c_4 are similar to the parameters used in conventional force-directed methods. Their settings should be coordinated so as to achieve a stable configuration layout of the graph. Parameter c_4 controls the magnitude of movement i.e. the range in which vertices can move. If c_4 is set smaller, then the range of movement of vertices is also smaller.

5.2 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. 11, 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 pre-defined. Red vertices are connected to form the boundary. Blue 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. 12.

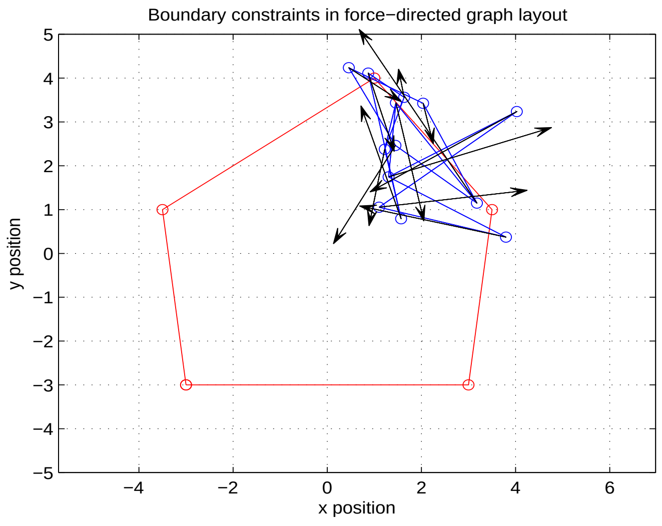


Figure 11: Initial layout of a random graph.

6 Results and Analysis

In this section, we apply our boundary constraints on several graph datasets using the graph generation method discussed in previous section and also other graph datasets available online. First we plug in different boundary force functions to show how the graph layout changes with different boundary constraints. , then we discuss the complexity of this algorithm by running it on different scales of graph data, and in the last part, we show some visual results of arbitrarily shaped boundaries and animation sequence of altering the boundary during the graph layout process.

6.1 Experimental Results for Different Boundary Force Functions

We know the boundary forces are applied to each graph vertex according to which active area of the boundary edge it falls into, and it also depends on the distance from the graph vertex and to the boundary edge. Here, we demonstrate how flexible this arrangement is by changing the boundary force functions to affect the layout of a graph. Equation (7) specifies an inverse distance (d) relationship of boundary force on a graph vertex. Fig. 13 illustrates how the positions of the graph vertices

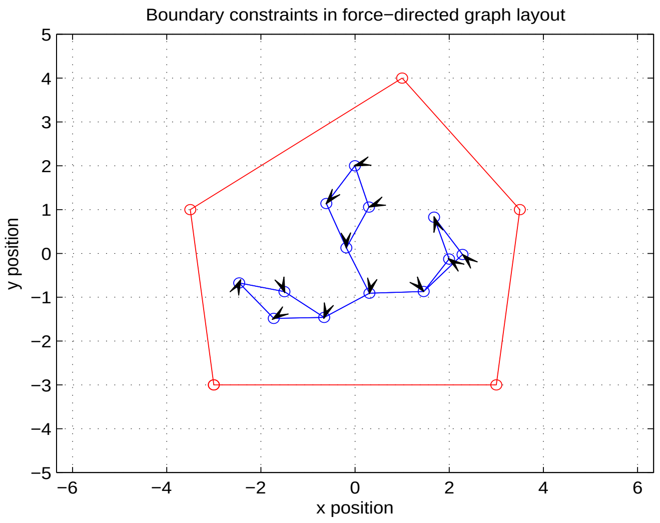


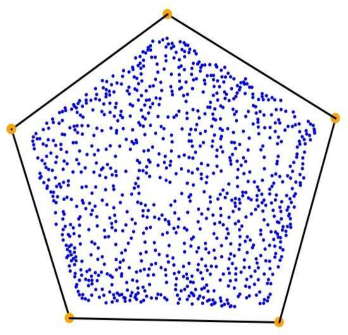
Figure 12: Final layout of the graph after 150 iterations.

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 same set of parameters, and the same shape of boundary of a regular pentagon for this experiment.

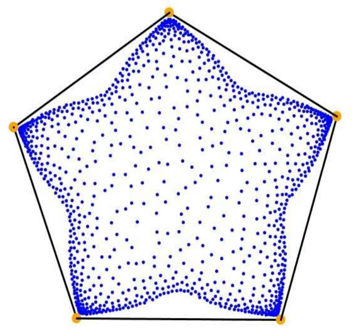
In Fig. 13(a), we are using inverse of d, by changing it to inverse of the logarithm of d, the vertices are pushed further away from the middle part of the boundary edges resulting in curved silhouettes as shown in Fig. 13(b). Since boundary force of inverse of d^2 is dropping much faster than inverse of d, we see graph vertices are closer to the boundary as indicated Fig. 13(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. 13(d).

6.2 Experimental Results for Different Scales of Graphs

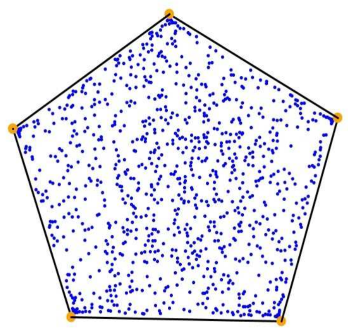
Similar to conventional force-directed methods as discussed in section 2, the complexity of this algorithm depends on number of vertices and edges in the graph [8]. Because the number of boundary vertices are much less than the number of vertices



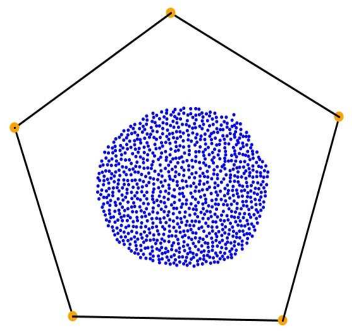
(a) Boundary force function of $1/d$



(b) Boundary force function of $1/\log d$



(c) Boundary force function of $1/d^2$



(d) Boundary force function of $1/\sqrt{d}$

Figure 13: Different boundary force functions.

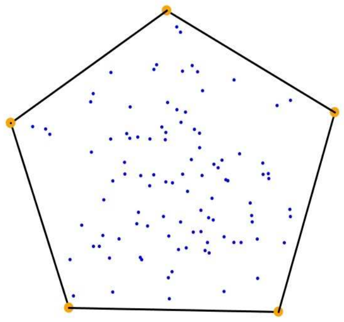
in the graph, the running time of our algorithm remains on the same level. We ran some experiments to obtain some actual running times. We first set the same ratio of vertices and edges in the graph, then increase the number of vertices and number of edges proportionately. Then we set the number of vertices as constant and increase the number of edges. We list the running times in Table 1 and show the resultant layouts in Fig. 14. 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. The more complicated topology of boundaries and graphs with edges showing will be discussed in the next section.

Table 1: Running time of datasets that have same ratio of vertices and edges

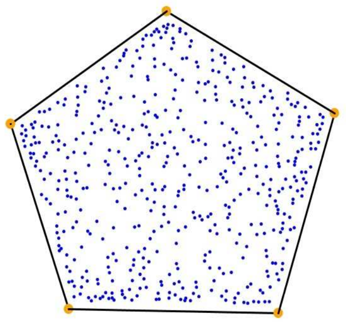
Number of vertices n	Number of edges m	Aver. Running time t (seconds)
100	200	0.87
500	1000	3.56
2000	4000	6.33
10000	20000	53.91

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. Then we fix the number of vertices in the graph to be constant, but increase the number of edges for each graph, the results are showing in Fig. 15. Note that we are using a graph dataset that has 2000 vertices and 6000 edges and 10000 edges respectively, and we increased the scale factor for spring (c_1) from previous experiments so a more clear view of the distribution of edges and properties of force-directed methods can be seen. Edges are drawn transparently.

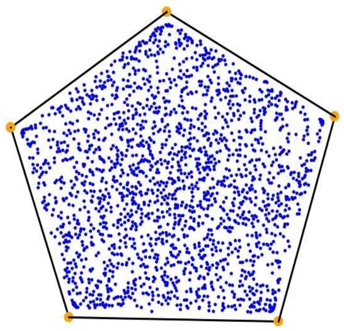
The results show that with fixed number of vertices, but increasing number of edges in the graph does not affect the time dramatically as increasing number of vertices in the graph. The complexity of the algorithm depends more on the number of vertices



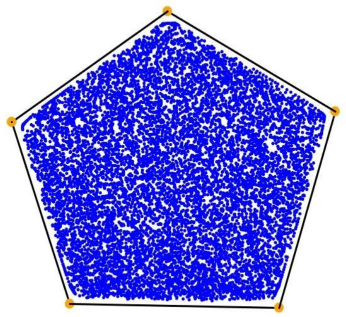
(a) Number of vertices $n = 100$



(b) Number of vertices $n = 500$

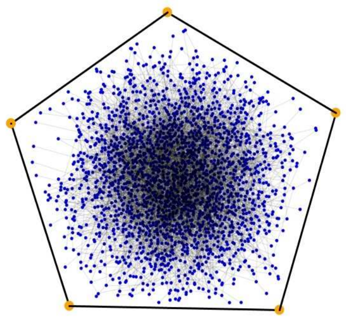


(c) Number of vertices $n = 2000$

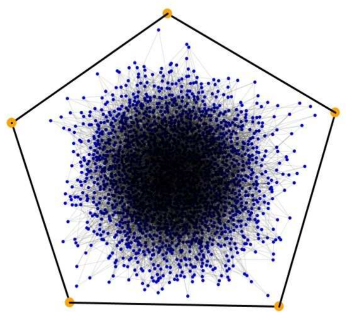


(d) Number of vertices $n = 10000$

Figure 14: Layout of datasets with have same ratio of vertices to edges.



(a) Number of edges $m = 6000$



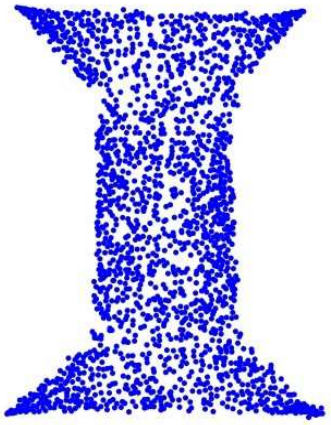
(b) Number of edges $m = 10000$

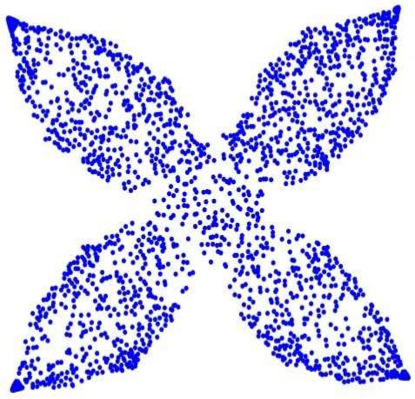
Figure 15: Layout of graphs with increasing number of edges.

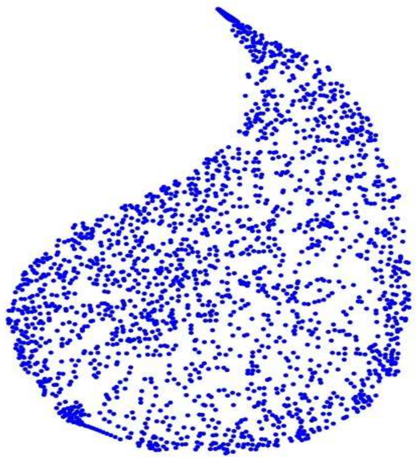
as indicated in previous sections.

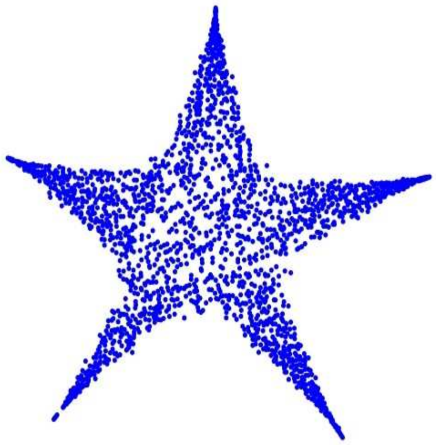
6.3 Visual Results for Arbitrary Topology of Boundaries

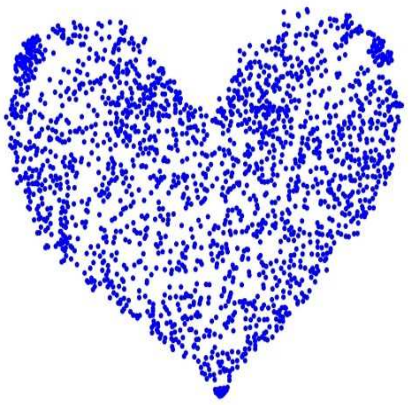
In this section, we defined several typical convex and concave shape of boundaries and tried our datasets on them. The visual results are listed in Fig. 16.
And we also defined multiple boundaries with layouts shown in Fig. 17. Fig. 17(a) with one interior boundary and Fig. 17(b) with two interior boundaries.
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. 18). In Fig. 19, have a graph inside a circle. As the user move some boundary vertices to make it look like a shape of moon, the layout changes accordingly. Another example illustrated in Fig. 20 simulates graph vertices spreading out to fill a funnel shape. At first, all the vertices in graph are at the top, as the algorithm runs, they expand and spread out over this funnel.


(a) Letter "I"


(b) Leaf


(c) Rain drop


(d) Star


(e) Heart

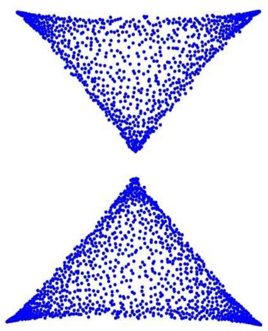
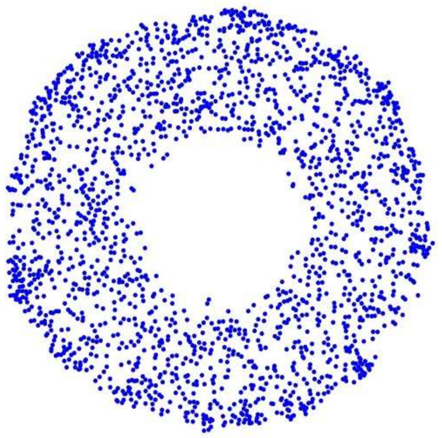
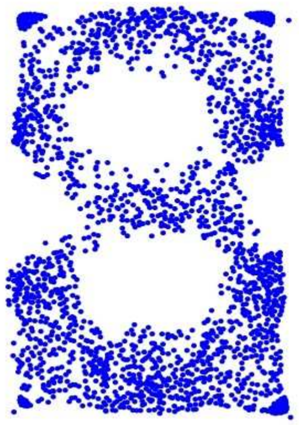

(f) Intersected

Figure 16: Graph layout with different boundary constraints.

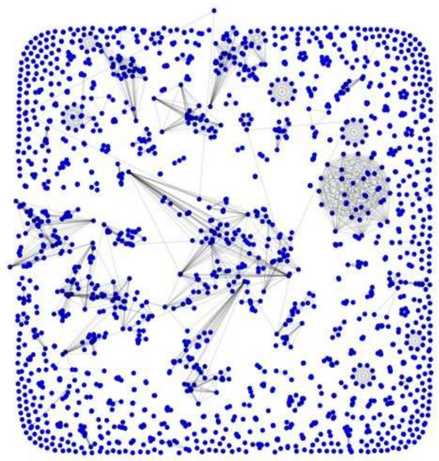


(a) Donut

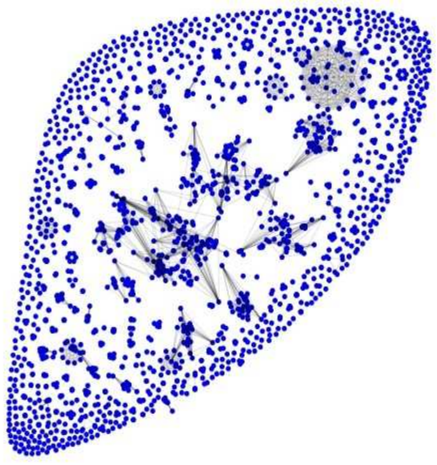


(b) Number 8

Figure 17: Different layouts with multiple boundaries.

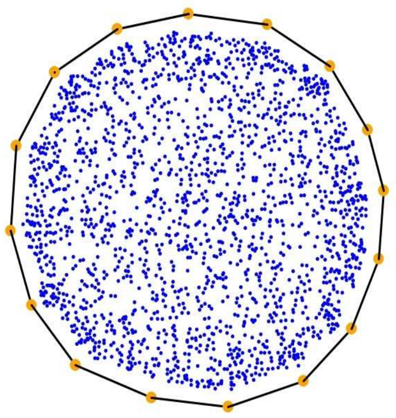


(a) Initial layout

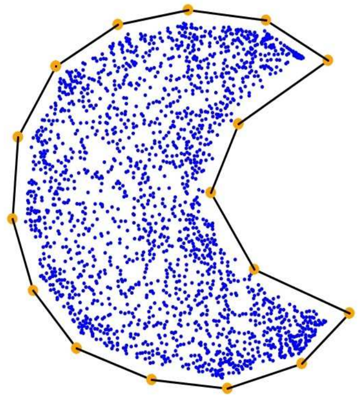


(b) After animation

Figure 18: Layouts from a facebook user dataset.

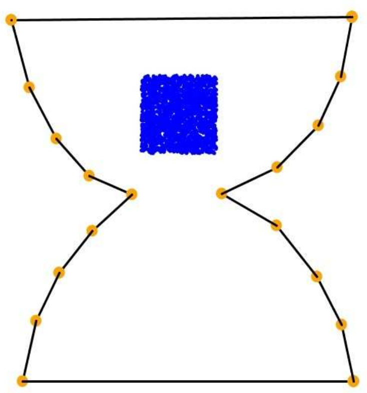


(a) Initial as "*sun*"

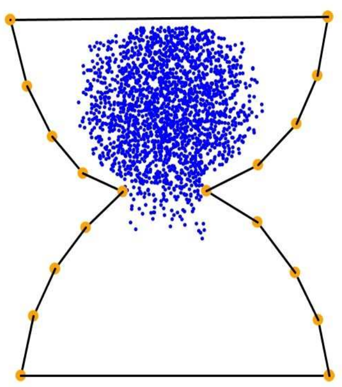


(b) Changed to "*moon*"

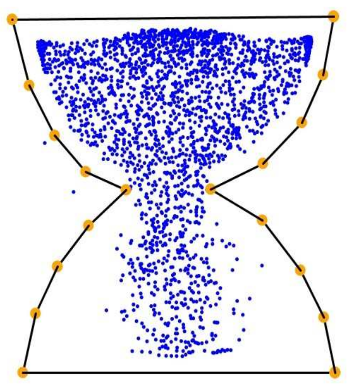
Figure 19: Layout changes from "sun" shape to "moon" shape.



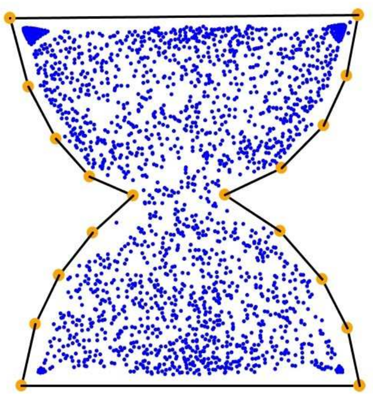
(a) Initial layout



(b) During process



(c) During process



(d) Final layout

Figure 20: Simulation of a funnel.

7 Conclusion and Future Work

This thesis presented a novel way for manipulating and specifying graph layout with the use of boundary constraints. This is incorporated with a force-directed simulation and does not significantly increase the cost of graph layout. 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. Future work is to constrain graph edges within boundaries as well. This will necessitate consideration of forces on graph edges, particularly around concave boundary vertices, self-intersecting boundaries and boundary topologies of holes.

References

[1] Battista, Di Giuseppe, Eades Peter, Tamassia Roberto, and G. Tollis Ioannis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry* 4, (5):235–282, 1994.

[2] Herman, Ivan, Guy Melanon, and Marshall M. Scott. Graph visualization and navigation in information visualization: A survey. visualization and computer graphics. *Visualization and Computer Graphics, IEEE Transactions on*, 6(1):24–43, 2000.

[3] Daz, Josep, Jordi Petit, and Serna Maria. A survey of graph layout problems. *ACM Computing Surveys (CSUR)*, 34(3):313–356, 2002.

[4] Kobourov and Stephen G. Spring embedders and force directed graph drawing algorithms. *arXiv preprint arXiv:*1201.3011, 2012.

[5] Battista, Di Giuseppe, Eades Peter, Tamassia Roberto, and G. Tollis Ioannis.

Graph drawing: algorithms for the visualization of graphs. Prentice Hall PTR, 1998.

[6] Franois Bertault. A force-directed algorithm that preserves edge crossing properties. *Graph Drawing. Springer Berlin Heidelberg*, 1999.

[7] Peter Eades. A heuristics for graph drawing. *Congressus numerantium*, (42):146–160, 1984.

[8] Ron Davidson and David Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics (TOG)*, 15(4):301–331, 1996.

[9] Chun-Cheng Lin, Yi-Yi Lee, and Hsu-Chun Yen. Mental map preserving graph drawing using simulated annealing. *Information Sciences*, 181(19):4253–4272, 2011.

[10] Arne Frick, Andreas Ludwig, and Heiko Mehldau. A fast adaptive layout algorithm for undirected graphs (extended abstract and system demonstration. *Graph Drawing Springer Berlin Heidelberg*, 1995.

[11] Fruchterman, Thomas MJ, and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[12] Stefan Hachul and Michael Jnger. Drawing large graphs with a potential-field-based multilevel algorithm. *Graph Drawing. Springer Berlin Heidelberg*, 2005.

[13] Koren David and Harel Yehuda. A fast multi-scale method for drawing large graphs. *Journal of graph algorithms and applications*, 6(3), 2002.

[14] Tomihisa Kamada and Satoru Kawai. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989.

[15] Kozo Sugiyama and Kazuo Misue. Graph drawing by the magnetic spring model. *Journal of Visual Languages and Computing*, 6(3):217–237, 1995.

[16] Chris Walshaw. A multilevel algorithm for force-directed graph drawing. *Graph Drawing. Springer Berlin Heidelberg*, 2001.

[17] Yifan Hu. Efficient, high-quality force-directed graph drawing. *Mathematica Journal*, 10(1):37–71, 2005.

[18] K. Sugiyama and K. Misue. A simple and unified method for drawing graphs: Magnetic-spring algorithm. *Proceedings of Graph Drawing, GD* 94, 894:364–375, 1995.

[19] Aaron Quigley and Peter Eades. Fade: Graph drawing, clustering, and visual abstraction. *In Graph Drawing*, pages 197–210, 2001.

[20] Jiggle Daniel Tunkelang. Java interactive general graph layout environment. *Sixth International Symposium on Graph Drawing (McGill University, Canada)*, August 1998.

[21] Mao Huang. On-line animated visualization of huge graphs. *Ph.D. thesis The University of Newcastle, Australia*, 1999.

[22] Michael. Bostock. D3. js-data-driven documents. (2012):2013.

[23] Weiqing He and Kim Marriott. Constrained graph layout. *Graph Drawing. Springer Berlin Heidelberg*, 1997.

[24] Roberto Tamassia. Constraints in graph drawing algorithms. *Constraints*, 3(1):87–120, 1998.

[25] Thomas Kamps, Joerg Kleinz, and John Read. Constraint-based spring-model algorithm for graph layout. *Graph drawing. Springer Berlin/Heidelberg*, 1996.

[26] Tim Dwyer, Kim Marriott, and Michael Wybrow. Topology preserving constrained graph layout. *Graph Drawing. Springer Berlin/Heidelberg*, 2009.

[27] Kathy Ryall, Joe Marks, and Stuart Shieber. An interactive constraint-based system for drawing graphs. *In Proceedings of the* 10*th annual ACM symposium on User interface software and technology*, pages 97–104, 1997.

[28] Christian Tominski, James Abello, and Heidrun Schumann. Cgvan interactive graph visualization system. *Computers and Graphics*, 33(6):660–678, 2009.

[29] Edmund Dengler, Mark Friedell, and Joe Marks. Constraint-driven diagram layout. *In Visual Languages,* 1993.*, Proceedings* 1993 *IEEE Symposium on*, pages 330–335, 1993.

[30] Tomihisa. Kamada. Visualizing abstract objects and relations. *World Scientific Publishing Company*, 1989.

[31] W. T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, pages 743–768, 1963.

[32] W. Randolph Franklin. Pnpoly - point inclusion in polygon test.

[33] P. Eades and X. Lin. Spring algorithms and symmetry. *Theoretical Computer Science*, 2:379405, 2000.