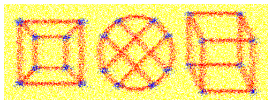

Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 20, no. 2, pp. 435–459 (2016)
DOI: 10.7155/jgaa.00401

Graph Layout with Versatile Boundary Constraints

Yani Zhang[1] *Alex Pang*[2]

[1]Expedia Inc.
[2]Computer Science Department, University of California at Santa Cruz

Abstract

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.

Submitted:	Reviewed:	Revised:	Reviewed:	Revised:
August 2015	November 2015	February 2016	February 2016	July 2016

Final:	Published:
July 2016	August 2016

Article type:	Communicated by:
Regular paper	Ulrik Brandes

E-mail addresses: yanizhang@soe.ucsc.edu (Yani Zhang) pang@soe.ucsc.edu (Alex Pang)

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 the fairly extensive body of literature about graph layout algorithms such as those described in field survey papers [7, 14], force-directed approach [2, 3, 18] 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 [4].

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 [9, 13, 17, 21]. 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 [21]. 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.

There are two motivations for this work. One is purely from an aesthetic or design perspective where the user is given the flexibility of specifying the boundary shape wherein a graph is to be laid out. The second is more a prospective use for information visualization where the boundary shape of the graph conveys some information about the graph. While most papers on graph drawing focus on the layout of nodes and edges, scaling to larger graphs, or speeding up the layout algorithms, there is one aspect that has not been investigated sufficiently. That is the use of the background space in conveying information. Most layouts are performed over an open ended domain with no boundary constraints and displayed on a rectangular window. What if the shape, e.g. as simple as the aspect ratio of the window, can be used to convey some information? Or more generally, what if the boundary constraint for the graph layout can be used to display some aspect of the data? The interactive capability described in this paper allows users to shape the boundary of graph layout to suit their application needs or aesthetic desires. But it also offers the possibility to explore data dependent boundaries in the future.

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 described 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

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 [19] 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) [22] 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 [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 there are very powerful, focusing mechanisms used in magnification, fisheye, or other layout lenses described in [23], they are different from what we are proposing in that they typically deal with only a subset of the graph or the layout space.

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.

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 [15]. 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.

Interestingly, layout constraints can also be used to provide both structural and dynamic stability [1]. Structural stability refers to the constraints specified by users or application programs. Constraints are specified as linear equations for each dimension independently. Dynamic stability refers to a small change in graph layout as a result of a small change in the structure of the graph.

A comprehensive approach for constrained graph drawing was presented by He and Marriot [13]. 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 [16] to support such placement constraints which is fast and produces good results in practice.

Two layout algorithms produce results that bear some potential resemblance to what our proposed approach can produce. Dwyer [8] introduced 3 novel constraints for graph layout that can support a wide variety of layouts. The third of these provides for constraints so that convex boundaries containing a number of nodes do not penetrate each other. Our proposed method supports this as well, but also supports concave boundaries as well as boundaries with different topologies. Simonetto et al. [20] adapts force directed algorithms to Euler like diagrams. A graph representing a set (where nodes are set elements and edges are set boundaries) is initially laid out. Set boundaries are generated automatically and constrain the set elements to stay within its boundaries. This differs from our work in that our boundaries are defined interactively by the user and uses significantly less boundary nodes and edges.

With the exception of [8, 20], 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 the complexity is dominated by the size of the graph. Using the same termination criteria, i.e. small enough change in energy function from the previous step or reaching a maximum number of iterations, we observe that the convergence rate with or without the boundary constraints are very similar. 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 users because the graph would take boundary constraints into consideration while converging to the final layout.

3 Force-directed Graph Layout

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 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_{Bp} is defined as a set 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, $E_{Bp} = \{e_p(1), e_p(2), \cdots, e_p(m)\}$ 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 [24] 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 [10] 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:

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

where c_1 represents a 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 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:

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

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 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.

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.

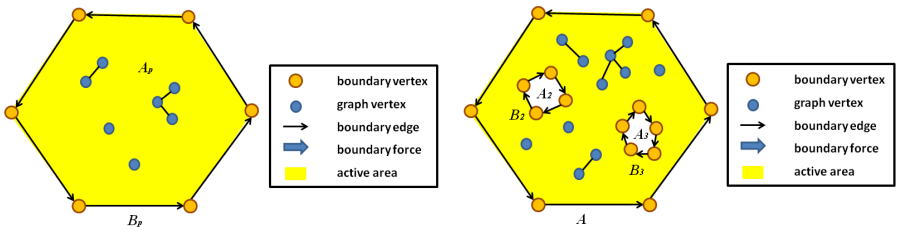
At a high level, the user interactively specifies a few boundary vertices that define a boundary. Each edge along the boundary then induces a boundary force on graph vertices that is added together with spring and gravitational forces. The boundary vertices are not affected by the forces from the graph elements and only serve to define the shape of the boundary. Because the boundary force calculations are from a small number of boundary edges and processed in the same way as spring and gravitational forces, adding boundary constraints does not significantly increase the computation costs.

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 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.



(a) One boundary case. (b) Multiple boundaries case.

Figure 1: Definition of active area of boundaries.

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 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 $f_{Bp}(v_G(j), e_p(i))$ (see Section 4.3) 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.

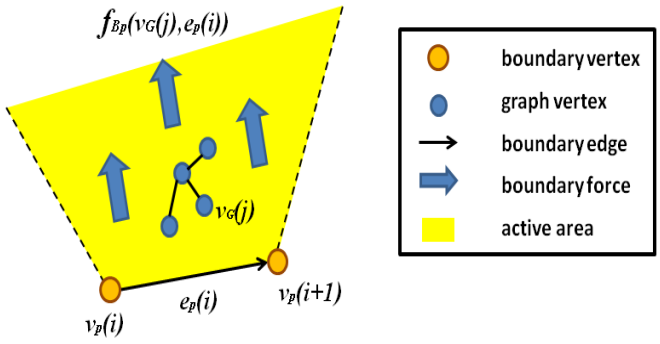


Figure 2: Definition of active area of boundary edges.

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. $v_p(i+1)$.

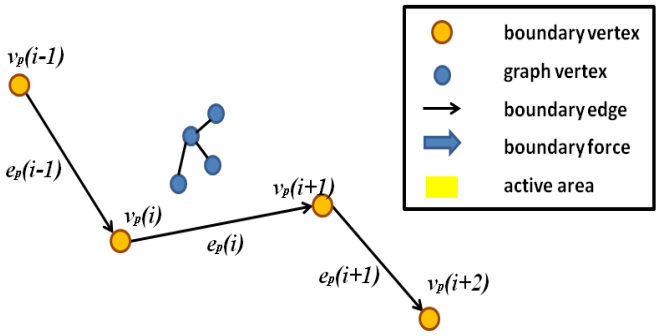


Figure 3: Convex and concave boundary vertices.

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 boundary vertices of $e_p(i)$ are convex (see Fig. 4(a)):

If both boundary vertices of boundary edge $e_p(i)$ are convex, then the active area of $e_p(i)$ is bounded by left vector $-e_p(i-1)$, boundary edge $e_p(i)$ and right vector $e_p(i+1)$ The active area is highlighted in yellow and extends through the half spaces inside of boundary edges $e_p(i)$, $e_p(i-1)$, and $e_p(i+1)$.

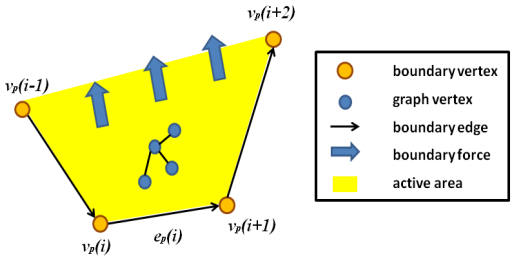
A force perpendicular to boundary edge $e_p(i)$ is applied to graph elements in the active area. Otherwise, boundary edge $e_p(i)$ does not contribute a force to the graph element.

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

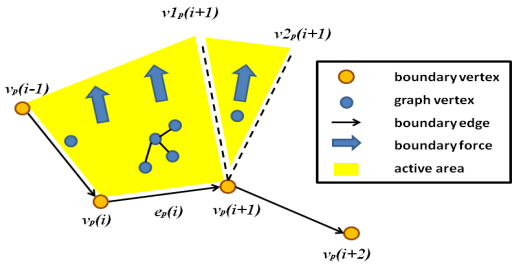
The active area is composed of two regions. The first region is bounded by left vector $-e_p(i-1)$, $e_p(i)$, and a vector from $v_p(i+1)$ to $v1_p(i+1)$ as the right vector. The second is a triangular region bounded by $\overline{v_p(i+1)v1_p(i+1)}$ and $\overline{v_p(i+1)v2_p(i+1)}$. $\overline{v_p(i+1)v1_p(i+1)}$ is perpendicular to $e_p(i)$ and $\overline{v_p(i+1)v2_p(i+1)}$ is perpendicular to $e_p(i+1)$. For graph elements falling in the first region, we apply a force perpendicular to boundary edge $e_p(i)$. For graph elements falling in the triangular region, 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)):

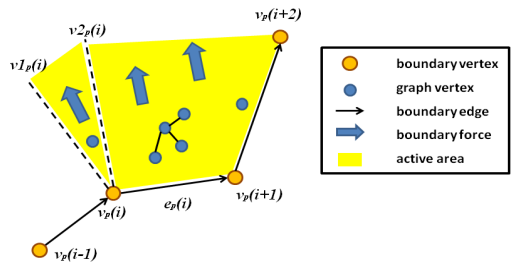
The active area is also composed of two regions. For graph elements in the triangular region, we apply a force in the direction from $v_p(i)$ to the graph element. Note that $\overline{v_p(i)v1_p(i)}$ is perpendicular to $e_p(i-1)$, while $\overline{v_p(i)v2_p(i)}$



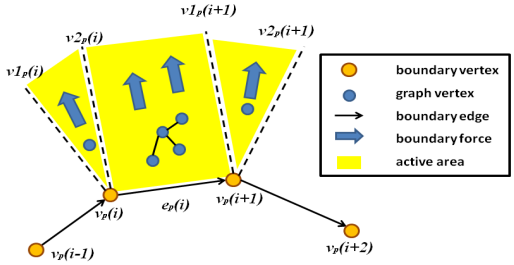
(a) Both boundary vertices of $e_p(i)$ are convex.



(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) Both boundary vertices of $e_p(i)$ are concave.

Figure 4: Four different cases of boundary edges and their corresponding active areas.

is perpendicular to $e_p(i)$. For graph elements in the half space bounded by $\overline{v_p(i)v2_p(i)}$, $e_p(i)$, and $\overline{v_p(i+1)v_p(i+2)}$. we apply a force perpendicular to boundary edge $e_p(i)$. 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 boundary vertices of $e_p(i)$ are concave (see Fig. 4(d)):

There are 3 active regions in this case – two triangular regions associated with concave boundary vertices and a non-triangular region. Just as in cases 2 and 3, a left concave boundary vertex will be bounded by a vector that is perpendicular to the preceding boundary edge. Likewise, a right concave boundary vertex will be bounded by a vector that is perpendicular to the next boundary edge. The middle region is the rectangular half space bounded by perpendicular vector $\overline{v_p(i)v2_p(i)}$, $e_p(i)$, and perpendicular vector $\overline{v_p(i+1)v1_p(i+1)}$. Forces are applied to graph elements as described in other cases.

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

(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. 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.

4.2.2 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 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.

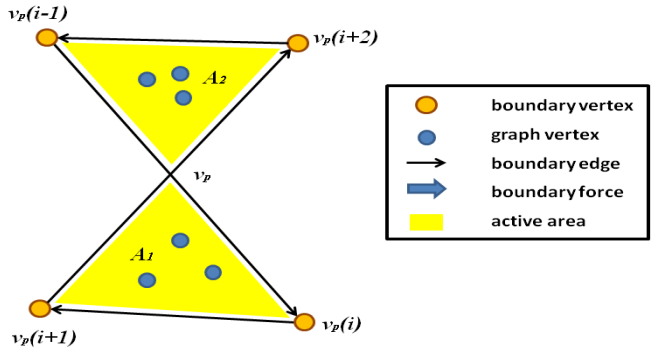


Figure 5: Single boundary and multiple active area.

4.2.3 Multiple Boundaries and Single Active Area

(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 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 4.3. 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.

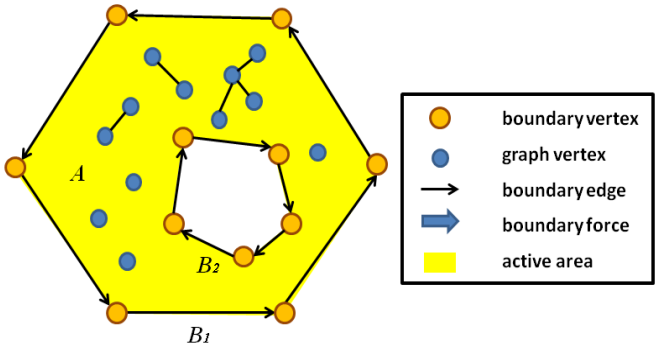


Figure 6: Multiple boundaries and single active area.

4.2.4 Multiple Boundaries and Multiple Active Area

Multiple active areas can arise from certain arrangements of multiple boundaries. 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 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.

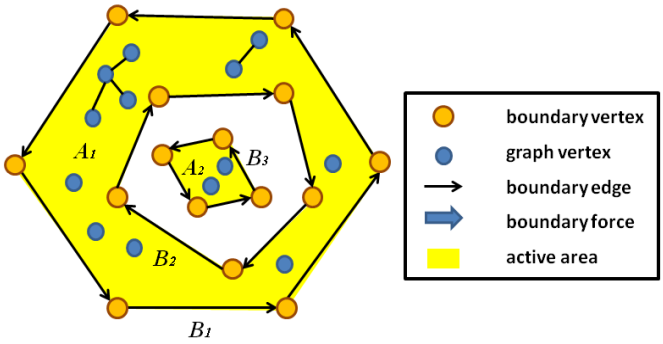


Figure 7: Multiple boundaries and multiple active area.

4.3 Boundary Forces

All graph vertices are affected by boundary forces. However, the force depends primarily on two factors: (i) whether the graph vertex is inside or outside the boundary, and (ii) how far the graph vertex is from the boundary. This results in two boundary force formulations depending on whether a graph vertex is inside or outside the boundary constraint, as well as two distance calculations depending on what type of active region the graph vertex falls in.

Before we assign boundary forces to a graph vertex, we run a point-in-polygon test such as the crossing test described by Haines [12] to determine whether it is inside or outside the boundary. This is an efficient $O(m)$ test that depends on the number of edges defining a boundary and can handle self-intersecting polygons properly.

In Fig. 8, we see that with respect to $e_p(i)$, $v_G(1)$ is in the active area (interior vertex) but $v_G(2)$ is not (exterior vertex). A graph vertex $v_G(j)$ that is *inside* the boundary will receive a boundary force f_{Bp} from boundary edge

$e_p(i)$ defined by:

$$f_{Bp}(v_G(j), e_p(i)) = c_5 \cdot \frac{1}{de(i)} \cdot \frac{\overline{ver(i)}}{|\overline{ver(i)}|} \quad i = 1, 2, \cdots, n, j = 1, 2, \cdots, m. \tag{3}$$

For a graph vertex $v_G(j)$ that is *outside* the boundary, we assign a *stronger* boundary force towards the interior of the boundary to pull them inside. That boundary force is defined by:

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

where $\overline{ver(i)}$ is a vector that is perpendicular to $e_p(i)$, $de(i)$ is the distance from the boundary edge $e_p(i)$ to $v_G(j)$, and c_5 is scaling parameter. Intuitively, c_5 is a weighting factor for how important the boundary force should be relative to the spring and gravitational forces acting on graph vertices. If it is set too large, the boundary forces will dominate and the graph may be tightly packed within the boundary. Judicious use of c_5 can result in a more aesthetic layout of the graph within the specified boundary. The distance $de(i)$ is defined by:

$$de(i) = \frac{\overline{vec(j)} \cdot \overline{ver(i)}}{|\overline{ver(i)}|} \tag{5}$$

where $\overline{vec(j)} = v_G(j) - v_p(i)$.

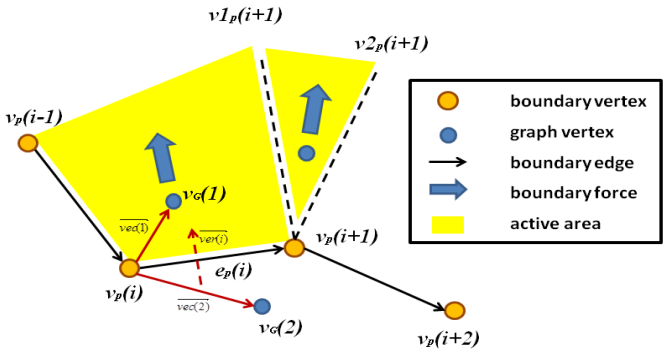


Figure 8: Illustration for calculating boundary forces of interior vs exterior graph vertices, and triangular vs non-triangular active areas.

If the graph vertex $v_G(j)$ falls within a triangular active area, $de(i)$ is replaced with the distance between the the graph vertex and the associated concave boundary vertex as follows:

$$de(i) = |\overline{v_p(i+1)v_G(j)}| \tag{6}$$

The boundary force formulation in Eqn. 3 is also changed slightly by replacing $\overline{ver(i)}$ with $\overline{v_G(j) - v_p(i+1)}$ for the case where $v_p(i+1)$ is a concave boundary vertex.

By iterating through all the boundary edges $e_p(i)$ in each of the B_p boundaries, the total boundary force acting on vertex $v_G(j)$ will be $\sum_{p=1}^{q} \sum_{i=1}^{n} f_{Bp}(v_G(j), e_p(i))$.

4.4 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.

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 modify Eqn. 1 to:

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

α is the number of vertices in the graph. β is the total active area. Together α/β represents the average density of vertices within the active area. This modification of Eqn. 1 allows the attractive spring forces to scale with the layout density.

4.4.2 Repulsive Force

Eqn. 2 is modified in a similar manner to take into account graph density:

$$f_r(v_G(i), v_G(j)) = c_3 \cdot \frac{\beta}{\alpha} \cdot \frac{\overline{v_G(i)v_G(j)}}{|\overline{v_G(i)v_G(j)}|^3} \tag{8}$$
$$i, j = 1, 2, \cdots, n$$

This time the force is inversely related to the density α/β.

4.4.3 Boundary Force

Eqn. 3 and Eqn. 4 are also modified in a similar manner to take into account graph density. For graph vertices that are within the boundary:

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

And for graph vertices that are outside the boundary:

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

4.5 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.

5 Implementation and Testing Tools

5.1 Implementation

We developed a prototype using Matlab and its graphic library. Tests were ran on an Intel Core *i*5 3GHz PC with 8GB of memory running Windows 7. Processing 2.0 was used to realize the interactive part with users.

It should be noted that the setting of parameters $c_1 \cdots c_5$ 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. 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, 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 and hence a slower convergence. On the other hand, a large value of c_4 may cause erratic and non-convergent behavior. Parameters $c_1 \cdots c_4$ are similar to the parameters used in conventional force-directed methods. The new parameter c_5 controls the weight of boundary forces. If c_5 is set to 0, there are in effect no boundary forces and the convergence falls back on the standard spring and gravity model. If c_5 is set too high (and assuming that $c_1 \cdots c_4$ were set to values that produce a stable layout), then the graph will be compressed into a

very tight ball within the boundary – which is not aesthetically pleasing. The setting of these parameters should be coordinated so as to achieve a stable and pleasing graph layout.

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. 9, 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. 10.

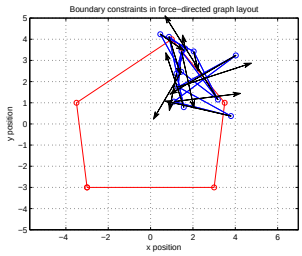


Figure 9: Initial layout of graph.

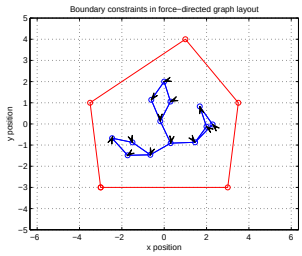


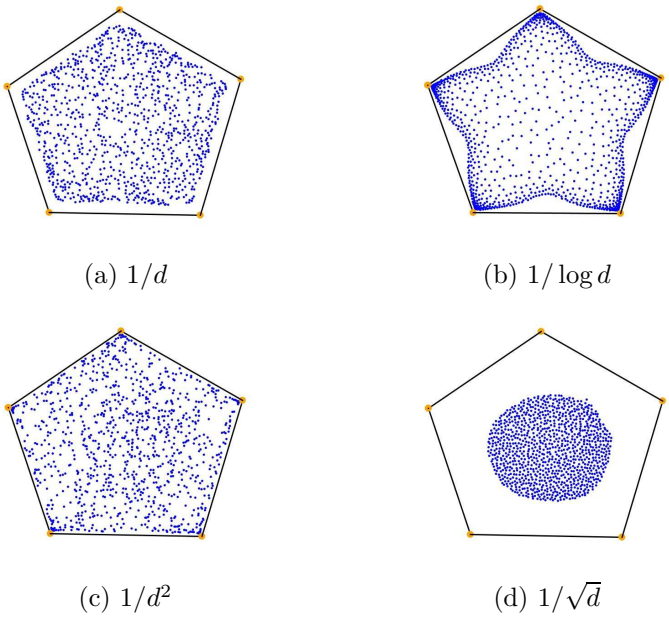
Figure 10: After 150 iterations.

6 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.

6.1 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 the graph layout. Eqn. 9 specifies an inverse distance



(a) $1/d$ (b) $1/\log d$

(c) $1/d^2$ (d) $1/\sqrt{d}$

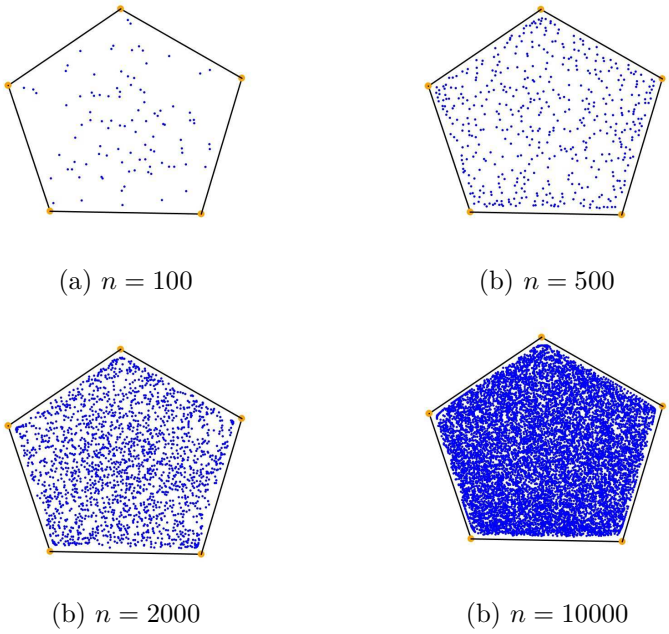
Figure 11: Different boundary force functions.

relationship of boundary force on a graph vertex. Fig. 11 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. 11(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. 11(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. 11(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. 11(d).

6.2 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



(a) $n = 100$ (b) $n = 500$

(b) $n = 2000$ (b) $n = 10000$

Figure 12: Layout of datasets with have same ratio of vertices to edges.

number of vertices and number of edges proportionately. The running times are listed in Table 1 and the resultant layouts are shown in Fig. 12. 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 1: Running time of datasets that have same ratio of vertices and edges

Number of vertices n	Number of edges m	Average 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.

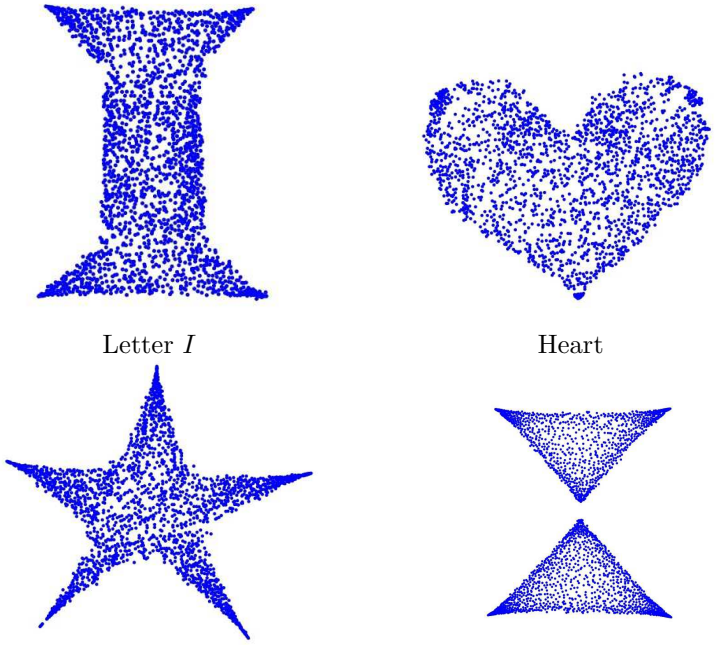
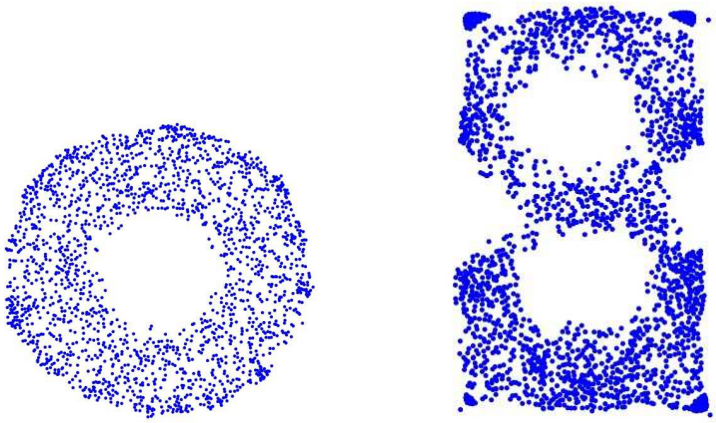


Figure 13: Graph layout with different boundary shapes.

6.3 Visual Results for Boundaries with Different Topology

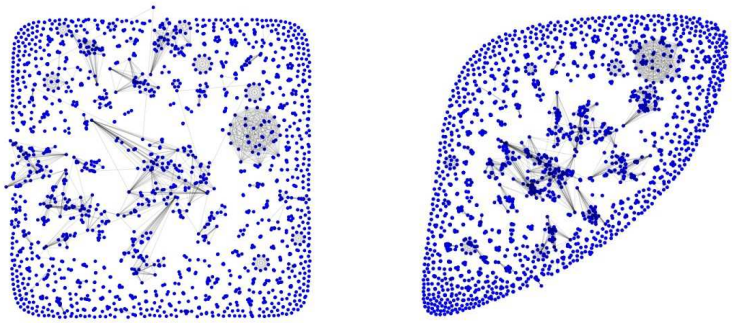
In this section, we define several convex and concave boundary shapes. The visual results are listed in Fig. 13. We also defined multiple boundaries with layouts shown in Fig. 14. Fig. 14(a) with one interior boundary and Fig. 14(b) with two interior boundaries.

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. 15). In Fig. 16, a graph inside a circular sun shape is rearranged to conform to a moon shaped boundary. Another example illustrated in Fig. 17 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.



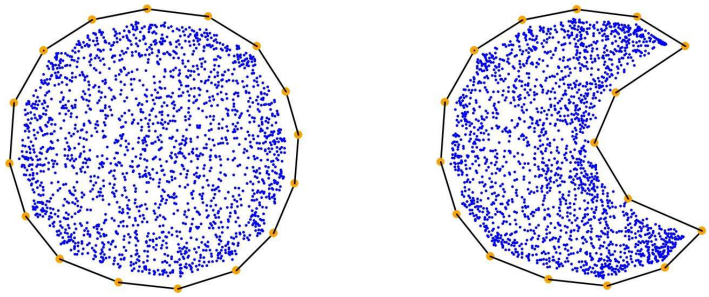
(a) Donut (b) Number 8

Figure 14: Different topologies with multiple boundaries.



Initial layout After animation

Figure 15: Layouts from a Facebook user dataset.



Initial sun shape Changed to moon shape

Figure 16: Layout changes from “sun” shape to “moon” shape.

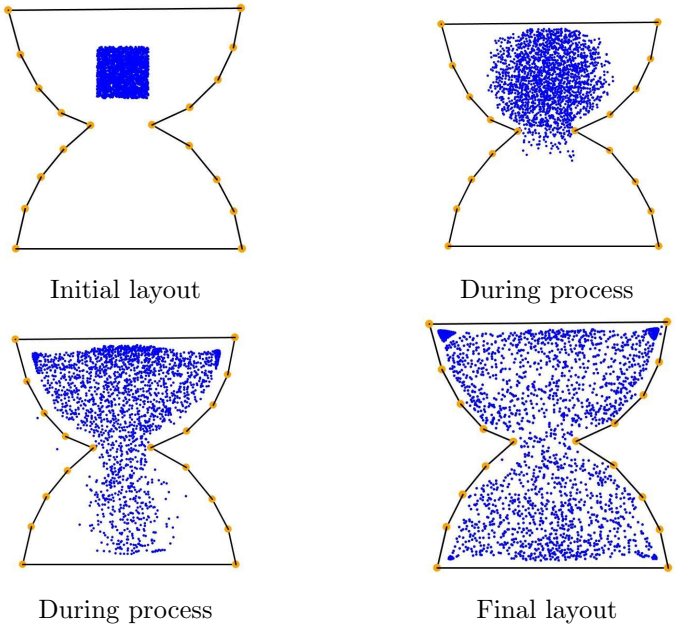


Figure 17: Simulation with an hour glass shape.

6.4 Convergence Properties

While we do not have a formal proof of convergence for the layout algorithm with boundary constraints, we have carried out multiple experiments with various boundary layouts and parameter sets. Figure 18 shows the asymptotic behavior of forces when there are no boundary constraints. Figures 19 (convex pentagon shape) and 20 (moon shape with concave boundary nodes) are representative results showing how the algorithm reaches an asymptotic stable behavior when boundary forces are introduced. The convergence rate is affected by the amount of empty space within the boundary and user interactions, and to a lesser degree by the number of boundary points and the shape of boundary.

7 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 not significantly increase the cost of graph layout. The force-directed simulator is self-contained and can be substituted with a more efficient implementation. We ran experiments on both synthetically generated graphs and publicly available graphs of fairly small size (10,000 nodes) where one can still make visually meaningful observations. The boundary constraints are quite general and can support arbitrary shapes including self-intersections

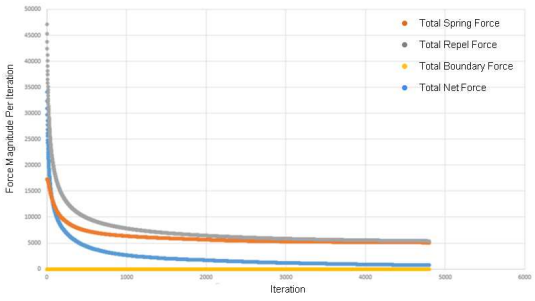


Figure 18: Asymptotic behavior when no boundary forces are present.

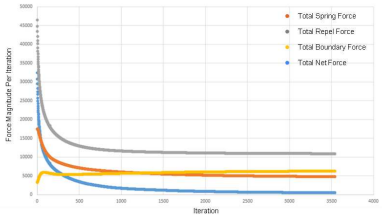


Figure 19: With pentagon boundary.

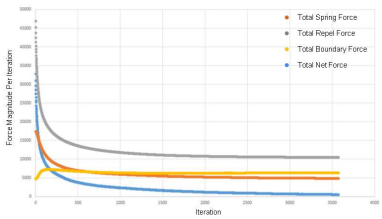


Figure 20: With moon boundary.

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 particular around concave boundary vertices, or to minimize edge crossings if different topologies are involved.

The shape of the boundary constraint is currently defined by the users. It may be possible to have a data dependent boundary constraint as a result of some initial graph analysis, or perhaps dependent on the context of the data e.g. outline of a country if the data is about the demographics of that country.

Orthogonal to the boundary constraints is the aesthetics of the graph layout within the boundaries. So, another possible extension is to incorporate aesthetic metrics of graph layout together with the boundary constraint considerations.

Acknowledgements

We thank the reviewers for their constructive comments to help improve the paper. We also thank Shweta Philip for her help with obtaining experimental evidence of algorithm convergence under various conditions.

References

[1] K.-F. Böhringer and F. N. Paulisch. Using constraints to achieve stability in automatic graph layout algorithms. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, pages 43–51, New York, NY, USA, 1990. ACM. doi:10.1145/97243.97250.

[2] U. Brandes. Drawing on physical analogies. In M. Kaufmann and D. Wagner, editors, *Drawing Graphs*, pages 71–86. Springer-Verlag, London, UK, 2001. doi:10.1007/3-540-44969-8_4.

[3] U. Brandes. Force-directed graph drawing. In M.-Y. Kao, editor, *Encyclopedia of Algorithms : A Springer Live Reference*, pages 1–6. Springer, New York, NY, 2014. doi:10.1007/978-3-642-27848-8_648-1.

[4] L. Chen and A. Buja. Stress functions for nonlinear dimension reduction, proximity analysis, and graph drawing. *Journal of Machine Learning Research*, 14(1):1145–1173, Apr. 2013.

[5] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics (TOG)*, 15(4):301–331, 1996. doi:10.1145/234535.234538.

[6] E. Dengler, M. Friedell, and J. Marks. Constraint-driven diagram layout. In *Visual Languages, 1993., Proceedings 1993 IEEE Symposium on*, pages 330–335. IEEE, 1993. doi:10.1109/VL.1993.269619.

[7] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: An annotated bibliography. *Computational Geometry Theory and Applications*, 4(5):235–282, Oct. 1994. doi:10.1016/0925-7721(94)00014-X.

[8] T. Dwyer. Scalable, versatile and simple constrained graph layout. In *Proceedings of the 11th Eurographics Conference on Visualization*, EuroVis'09, pages 991–1006, Chichester, UK, 2009. The Eurographs Association & John Wiley & Sons, Ltd. doi:10.1111/j.1467-8659.2009.01449.x.

[9] T. Dwyer, K. Marriott, and M. Wybrow. Topology preserving constrained graph layout. In *Graph Drawing*, pages 230–241. Springer, 2008. doi:10.1007/978-3-642-00219-9_22.

[10] P. Eades. A heuristics for graph drawing. *Congressus numerantium*, 42:146–160, 1984.

[11] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164, Nov. 1991. doi:10.1002/spe.4380211102.

[12] E. Haines. Point in polygon strategies. In P. S. Heckbert, editor, *Graphics Gems IV*, pages 24–46. Academic Press Professional, Inc., 1994.

[13] W. He and K. Marriott. Constrained graph layout. *Constraints*, 3(4):289–314, 1998. doi:10.1023/A:1009771921595.

[14] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, Jan 2000. doi:10.1109/2945.841119.

[15] T. Kamada. *Visualizing abstract objects and relations*, volume 5. World Scientific, 1989.

[16] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989. doi:10.1016/0020-0190(89)90102-6.

[17] T. Kamps, J. Kleinz, and J. Read. Constraint-based spring-model algorithm for graph layout. In *Proceedings of the Symposium on Graph Drawing*, GD '95, pages 349–360, London, UK, 1996. Springer-Verlag. doi:10.1007/BFb0021818.

[18] S. Kobourov. Force-directed drawing algorithms. In R. Tamassia, editor, *Handbook of Graph Drawing and Visualization*, pages 383–408. CRC, 2013.

[19] K. Ryall, J. Marks, and S. Shieber. An interactive constraint-based system for drawing graphs. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 97–104. ACM, 1997. doi:10.1145/263407.263521.

[20] P. Simonetto, D. Archambault, D. Auber, and R. Bourqui. ImPrEd: An improved force-directed algorithm that prevents nodes from crossing edges. *Computer Graphics Forum*, 30(3):1071–1080, 2011. doi:10.1111/j.1467-8659.2011.01956.x.

[21] R. Tamassia. Constraints in graph drawing algorithms. *Constraints*, 3(1):87–120, 1998. doi:10.1023/A:1009760732249.

[22] C. Tominski, J. Abello, and H. Schumann. CGV an interactive graph visualization system. *Computers & Graphics*, 33(6):660–678, 2009. doi:10.1016/j.cag.2009.06.002.

[23] C. Tominski, S. Gladisch, U. Kister, R. Dachselt, and H. Schumann. A survey on interactive lenses in visualization. In R. Borgo, R. Maciejewski, and I. Viola, editors, *EuroVis - STARs*. The Eurographics Association, 2014. doi:10.2312/eurovisstar.20141172.

[24] W. T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, 13:743–768, 1963.