# FlexGD : A Flexible Force-directed Model for Graph Drawing

Anne-Marie Kermarrec, Afshin Moin

# FlexGD : A Flexible Force-directed Model for Graph Drawing

Anne-Marie Kermarrec and Afshin Moin
INRIA Rennes Bretagne Atlantique, France

**Abstract**

We propose *FlexGD*, a novel force-directed algorithm for straightline undirected graph drawing. The algorithm strives to draw graph layouts encompassing from uniform vertex distribution to extreme structure abstraction. It is flexible for it is parameterized so that the emphasis can be put on either of the two drawing criteria. The parameter determines how much the edges are shorter than the average distance between vertices. Extending the clustering property of the LinLog model, FlexGD is very efficient for cluster visualization in an adjustable level. The energy function of FlexGD is minimized through a multilevel approach, particularly designed to work in contexts where edge length distribution is not uniform. Applying FlexGD on several real datasets, we illustrate both the good quality of the layout on various topologies, and the ability of the algorithm to meet the addressed drawing criteria.

## 1 Introduction

Force-directed algorithms [1, 2, 3, 4, 5, 6] are the prevalent approach to graph drawing. Their popularity rises from high layout quality, ease of implementation and code flexibility. They model vertices as a collection of particles and assign them attractive and repulsive forces according to force shapes improvised from physical metaphors like springs and electrical charges. The algorithm lays out the graph from an initial random configuration computing the net force on each vertex and moving the vertices iteratively until an equilibrium state is achieved between all forces. Force-directed algorithms are composed of two components: the energy function and the minimization algorithm. The energy function assigns a scalar energy value to the layout. Attractive and repulsive forces are linked to the energy function as force is the minus gradient of energy. The role of the minimization algorithm is to compute a force equilibrium in the system, being equivalent to a local minimum of the energy function. These algorithms are heuristics finding a local minimum of this function.

Attractive and repulsive forces (or equivalently the energy function) are defined in the goal of meeting some criteria of aesthetic drawing like uniform edge length and minimum edge crossing. The most widely used conventional models, the Spring-Electrical model [1] and the Stress model [3], enforce uniform edge length over the layout. Recently, Noack [4] has investigated the influence of the shape of attraction and repulsion energies[1] on the clus-

[1]Energy is a state representable by a scalar. The terms *repulsion* and *attraction* can only be used for forces which are vectorial quantities. However, with a slight abuse of notation we use them for the corresponding terms in the energy function too.

tering properties of a model. The author shows *Lin*ear attraction and *Log*arithmic repulsion energies are better for cluster visualization than prior energy functions. The *LinLog* model is consequently proposed, and some of its properties are derived. However, no multilevel algorithm [7, 8, 9] is provided for its minimization.

In this paper, we suggest FlexGD, a Flexible force-directed algorithm for Graph Drawing. FlexGD draws graphs according to the two criteria of uniform vertex distribution and structure abstraction. The model is flexible, in that it is parameterized to be biasable towards any of the two drawing criteria, according to user preferences. The core idea is to use both attractive and repulsive forces to distribute the vertices over the drawing area. More specifically, we replace the pairwise logarithmic repulsion energy of the LinLog model with linear-logarithmic energy, while preserving the linear attraction of edges intact. This modification has multiple advantages. First of all, FlexGD draws more pleasing layouts. Namely, the drawing area is filled optimally and the layout is pleasing in the frontiers. Secondly, it upgrades the property of cluster visualization of LinLog to *abstractable* cluster visualization, that is, the user can decide upon the density of the clusters and to what extent they are set apart. Finally, the model is capable of drawing disconnected graphs as most of the previous models have difficulties with their handling.

We derive formally some properties of the FlexGD energy model and compare them with those of LinLog. The energy function is minimized through a multilevel approach overcoming the non-uniform distribution of edge lengths. FlexGD layouts of some large real datasets are presented to illustrate the algorithm can generate quality layouts in a wide range of abstraction, satisfying different user preferences.

## 2 Definitions and Notations

A $d$-dimensional layout $p$ of a graph $G = (V, E)$ is a mapping of vertices into the Euclidean space $M : V \longrightarrow R^d$, where $\forall u \in V$ is assigned with a coordinate vector $p_u$. The Euclidean distance between $u$ and $v$ is denoted by $\|p_u - p_v\|$. We will use some notions from the literature on graph clustering to express the properties of FlexGD. The *cut* and the *density* are two measures widely used as the coupling between two subgraphs. Minimizing the coupling is an established technique in graph clustering [10, 11]. The cut between two disjoint subgraphs $V_1$ and $V_2$ is defined as $cut[V_1, V_2] = \left|E_{V_1 \times V_2}\right|$, where $E_{V_1 \times V_2}$ represents the set of edges between $V_1$ and $V_2$. The cut has the disadvantage of selecting biased clusters, i.e. one huge cluster against a tiny one. This is undesirable as clusters are supposed to contain a reasonably large group of vertices. One way to bypass this problem is to punish small clusters by dividing the cut by the size of clusters. This leads to the definition of density:

$$density(V_1, V_2) = \frac{cut[V_1, V_2]}{|V_1|\,|V_2|}.$$

Arithmetic, geometrical and harmonic mean are the most popular definitions in the literature to measure the mean distance between a set of vertices. For $F \subset V^{(2)}$ and layout $p$,

they are defined as:

$$arith(F,p) = \frac{1}{|F|} \sum_{\{u,v\}\in F} \|p_u - p_v\|.$$

$$geo(F,p) = \sqrt[|F|]{\prod_{\{u,v\}\in F} \|p_u - p_v\|}.$$

$$harm(F,p) = \frac{|F|}{\sum_{\{u,v\}\in F} \frac{1}{\|p_u - p_v\|}}.$$

In FlexGD, attractive forces are reinforced by some abstraction constant. Hence, we found it helpful to generalize the definition of the arithmetic mean in order to write the theorems in a more readable form. For $F \subset V^{(2)}$ the *generalized arithmetic mean* is defined as:

$$arith^{k+}(F,p) = \frac{\sum_{\{u,v\}\in F} \lambda_{uv}\|p_u - p_v\|}{\sum_{\{u,v\}\in F} \lambda_{uv}}, \quad \lambda_{uv} = \begin{cases} k+1 & if\ \{u,v\} \in E_F \\ 1 & otherwise \end{cases},$$

where $E_f$ is the set of edges over $F$. This definition gives more weight to pairs of connected vertices than pairs of disconnected vertices.

## 3 The FlexGD Energy Function

For layout $p$ of a graph $G = (V, E)$, many of the known energy models [4, 1, 2] have the following form:

$$U(p) = \sum_{\{u,v\}\in E} f(\|p_u - p_v\|) + \sum_{\{u,v\}\in V^{(2)}} g(\|p_u - p_v\|), \tag{1}$$

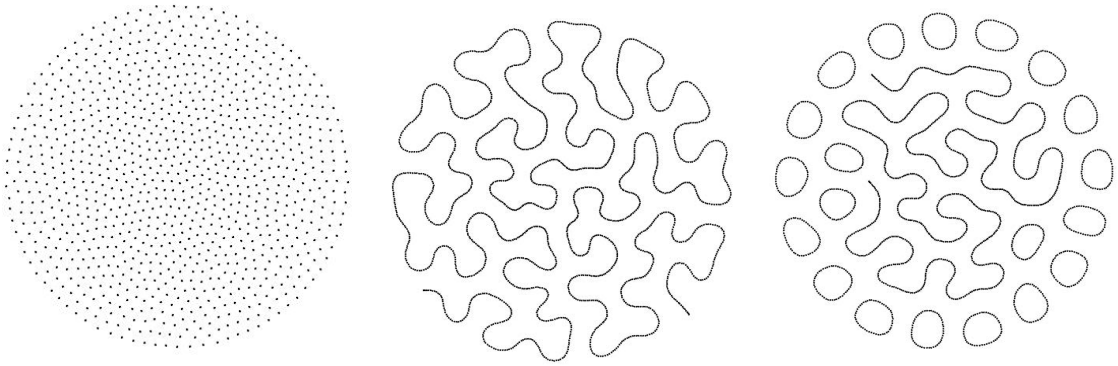
where $f(\|p_u - p_v\|)$ is associated with the attraction of edges, and $g(\|p_u - p_v\|)$ with the repulsion between all pairs of vertices. The minus gradient of $f$ and $g$ determines the attractive and repulsive forces. In the LinLog model, the energy function is:

$$U_{LinLog}(p) = \sum_{\{u,v\}\in E} \|p_u - p_v\| - \sum_{\{u,v\}\in V^{(2)}} \ln \|p_u - p_v\|.$$

The FlexGD energy function is defined as:

$$U(p,k) = \sum_{\{u,v\}\in E} k\|p_u - p_v\| + \sum_{\{u,v\}\in V^{(2)}} \|p_u - p_v\| - \ln \|p_u - p_v\|. \tag{2}$$

The first term captures the graph structure by shortening the edges, while the second term distributes the vertices evenly over the drawing surface. In models like [4, 1, 2], $g$ is monotonously decreasing. As a result, disconnected vertices are likely to repulse each other towards infinity. On the contrary, both attractive and repulsive components are present in $g$ of FlexGD. Hence, disconnected vertices rest in a finite neutral distance from each other, explaining why FlexGD can draw even totally disconnected graphs. Parameter $k$ determines how much the edges must be shorter with respect to the mean neutral distance. Figure 1 shows how a



(a) a disconnected graph of 1000 vertices.

(b) a line of 2000 vertices.

(c) a line of 1000 vertices with 20 circles of 50 vertices.

Figure 1: Even distribution of the graph over the drawing area.

graph is uniformly packed within a circular drawing area regardless of its connectivity. The attractive force of edges and the pairwise force exerted on a vertex $u$ from another vertex $v$ are:

$$\vec{f_a}(u,v) = \frac{k(p_v - p_u)}{\|p_v - p_u\|} \text{ if } \{u,v\} \in E, \tag{3}$$

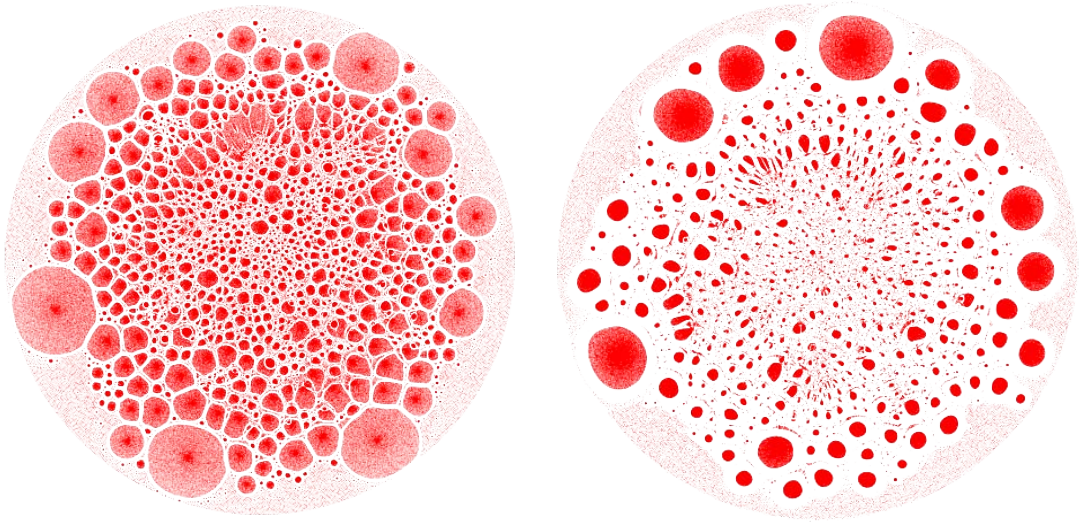
$$\vec{f_r}(u,v) = (1 - \frac{1}{\|p_v - p_u\|}) \cdot \frac{(p_v - p_u)}{\|p_v - p_u\|} \ \{u,v\} \in V^{(2)}. \tag{4}$$

The overall force exerted on $u$ from $v$ is then $\vec{f_a} + \vec{f_r}$.

### 3.1 Properties of the Model

It is proved that adding multiplicative constants to the attractive and repulsive terms of previous energy models does not change the minimum energy layout, but only scales it [4, 9]. However, the minimum energy layout of FlexGD changes with $k$. This gives FlexGD the flexibility of drawing layouts in different levels of abstraction. Figure 2 shows the layout of an email network containing 265214 vertices and 420045 edges. Edges are not represented for more clarity. The abstraction constant is chosen as $k = 4000$ in Figure 2a. The clustered nature of this webmail graph is clear in this figure. Though, one may choose to abstract the layout more at the price of viewing less details. The layout of the same graph is shown in Figure 2b for $k = 20000$.

Figure 2 also demonstrates two further assets of FlexGD layouts. Firstly, there is an empty space around clusters. It is very helpful in distinguishing the frontier between them. These clusters are particularly meaningful in social networks or web graphs, where they represent friendship groups or societies. This effect is due to the intra-distance between the vertices of a cluster being small with respect to their average distance from the rest of the graph. Consequently, they act as supernodes with high mass, exerting strong repulsion to the outside vertices pushing them further from the community. This effect increases with $k$,



(a) email graph, $k = 4000$.

(b) email graph, $k = 20000$.

Figure 2: Email network from a large European research institution.

as the clusters get denser for higher $k$. Secondly, disconnected vertices are put towards the frontiers. This prevents them from being visually noisy to the connected components of the graph.

Here, we derive some properties of FlexGD. The goal is to understand more quantitatively how the model makes a tradeoff between the two drawing criteria, and how it separates the clusters. This analysis elaborates the influence of $k$ on the behavior of FlexGD, and highlights its differences from the LinLog model.

Theorem 3.1 states FlexGD finds the best compromise between maximizing the geometrical mean and minimizing the generalized arithmetic mean of all vertices. It is responsible for shortening the edges and lengthening the non-edges. If the graph contains no edges, the generalized arithmetic mean is equal to the usual arithmetic mean. The maximum of the ratio is then one, as it is a well-known fact from AM-GM inequality that the geometrical mean is always greater than or equal to the arithmetic mean. The maximum is achieved when all distances are equal. Though, in the 2-dimensional space equality is impossible for more than 3 vertices, because of geometrical constraints. Consequently, the model distributes the vertices uniformly in order to maximize the ratio by closing the two means as much as possible. This property explains why the vertices have a perfect even distribution over the drawing area in Figure 1a. When edges reside in the graph, connected vertices are put closer to each other. The reason is they are weighted more in the generalized arithmetic mean. Therefore, their further shortening, up to some extent controlled by $k$, decreases the generalized arithmetic mean more than it increases the geometrical mean.

**Theorem 3.1** *The minimization of the FlexGD energy function is equivalent to the minimization of* $\frac{arith^{k+}(V^{(2)},p)}{geo(V^{(2)},p)}$.

**Proof** Let $p^0$ be a layout with minimum FlexGD energy.
If $\sum_{\{u,v\}\in E} \|p_u^0 - p_v^0\| + \sum_{\{u,v\}\in V^{(2)}} \|p_u^0 - p_v^0\| = c$, then $p^0$ is a solution to:

$$\text{minimize}(-\sum_{\{u,v\}\in V^{(2)}} \ln \|p_u - p_v\|)$$

$$\text{subject to} \sum_{\{u,v\}\in E} \|p_u - p_v\| + \sum_{\{u,v\}\in V^{(2)}} \|p_u - p_v\| = c.$$

The above expression may be reformulated in the form of
$\text{minimize} - \ln(geo^{\left|V^{(2)}\right|}(V^{(2)},p))$.
Since $\sqrt[\left|V^{(2)}\right|]{\exp(x)}$ is an increasing function of $x$, the minimization of this expression is equivalent to $\text{minimize}\exp(\ln \frac{1}{geo(V^{(2)},p)})$. Multiplying the numerator by the constant $arith^{k+}(V^{(2)},p)$, and rewriting the restriction, we obtain:

$$\text{minimize}\frac{arith^{k+}(V^{(2)},p)}{geo(V^{(2)},p)} \quad \text{subject to} \quad arith^{k+}(V^{(2)},p) = \frac{c}{|E| + \left|V^{(2)}\right|}.$$

Suppose there exists a layout $q^0$ of $G$ with minimum FlexGD energy for which $\frac{arith^{k+}(V^{(2)},q^0)}{geo(V^{(2)},q^0)} < \frac{arith^{k+}(V^{(2)},p^0)}{geo(V^{(2)},p^0)}$. We can always define a scaling
$q^1 = \frac{c}{(|E|+\left|V^{(2)}\right|)arith^{k+}(V^{(2)},q^0)} q^0$ for which $arith^{k+}(V^{(2)},q^1) = \frac{c}{|E|+\left|V^{(2)}\right|}$, but
$\frac{arith^{k+}(V^{(2)},q^1)}{geo(V^{(2)},q^1)} = \frac{arith^{k+}(V^{(2)},q^0)}{geo(V^{(2)},q^0)} < \frac{arith^{k+}(V^{(2)},p^0)}{geo(V^{(2)},p^0)}$. This is a contradiction. Hence $q^0$ does not exist and the restriction may always be removed. ∎

Theorem 3.2 posits that in 1-dimensional FlexGD layouts of bipartitions, the distance between the two partitions of a graph decreases with $k$ times their density. This theorem does not generalize to more than one dimension, but remains *approximately* true for 1+ dimensional layouts of *clusterizable bipartitions*. This approximation is discussed in detail in Section 3.2. For graphs containing a higher number of clusters, there is in general no 2D or 3D drawing where the distance between *every* two clusters obey a same equation, without violating the triangle inequality w.r.t. a third cluster. Despite, Theorem 3.2 illustrates the logic behind the separation of clusters in FlexGD layouts.

**Theorem 3.2** *Let* $p^0$ *be a one-dimensional drawing of the graph* $G = (V, E)$ *with minimum FlexGD energy. Let* $(V_1, V_2)$ *be a bipartition of* $V$ *such that the vertices in* $V_1$ *have smaller positions than the vertices in* $V_2$ *(i.e.* $\forall v_1 \in V_1,\ \forall v_2 \in V_2 : p_{v_1} < p_{v_2}$*). Then:*

$$harm(V_1 \times V_2, p^0) = \frac{1}{1 + k * density(V_1, V_2)}.$$

**Proof** Let $p^0$ be a layout with minimum FlexGD energy. If we add $d \in R$ to the coordinates of the vertices of $V_1$ in a way that the largest coordinate of the vertices in $V_1$ remains less than the smallest coordinate of the vertices in $V_2$, the FlexGD energy becomes:

$$\begin{aligned} U(d, p^0) &= \sum_{\{u,v\} \in E_{V_1^{(2)}} \cup E_{V_2^{(2)}}} k\,|p_u - p_v| \\ &+ \sum_{\{u,v\} \in V_1^{(2)} \cup V_2^{(2)}} |p_u - p_v| - \ln |p_u - p_v| \\ &+ \sum_{\{u,v\} \in E_{V_1 \times V_2}} k(|p_u - p_v| + d) \\ &+ \sum_{\{u,v\} \in V_1 \times V_2} |p_u - p_v| + d - \ln(|p_u - p_v| + d). \end{aligned}$$

Since $p^0$ is a layout with minimum energy, the above function has a minimum at $d = 0$, i.e. $U'(d = 0, p^0) = 0$. Then:

$$k \left| E_{V_1 \times V_2} \right| + |V_1 \times V_2| = + \sum_{\{u,v\} \in V_1 \times V_2} \frac{1}{|p_u - p_v|}.$$

Replacing the right side with $\frac{|V_1||V_2|}{harm(V_1 \times V_2, p^0)}$ and $|V_1 \times V_2|$ with $|V_1|\,|V_2|$, the result is directly obtained. ∎

While Theorem 3.1 explains how convex subgraphs are clustered in the FlexGD layouts, Theorem 3.2 is responsible for separation of clusters in function of their coupling. This reminds the definition of clustering, where vertices inside a cluster must be as similar as possible, while being dissimilar from vertices of the other clusters. At this point, we would like to add that extra parameters do not give more features to the model. Theorem 3.3 formalizes this finding:

**Theorem 3.3** *The minimum of*

$$U = \sum_{\{u,v\} \in E} k_1 \|p_u - p_v\| + \sum_{\{u,v\} \in V^{(2)}} k_2 \|p_u - p_v\| - k_3 \ln \|p_u - p_v\|$$

*is equal to the minimum of*

$$U' = \sum_{\{u,v\} \in E} \frac{k_1}{k_2} \|p_u - p_v\| + \sum_{\{u,v\} \in V^{(2)}} \|p_u - p_v\| - \ln \|p_u - p_v\|,$$

*up to scaling by* $\frac{k_3}{k_2}$.

| Model | Minimization equivalence | One-dimensional bipartition | abstractable |
|---|---|---|---|
| LinLog [4] | minimize $\frac{arith(E,p)}{geo(V^{(2)},p)}$ | $harm(V_1 \times V_2, p^0) = \frac{1}{density(V_1,V_2)}$ | NO |
| FlexGD | minimize $\frac{arith^{k+}(V^{(2)},p)}{geo(V^{(2)},p)}$ | $harm(V_1 \times V_2, p^0) = \frac{1}{1+k*density(V_1,V_2)}$ | YES |

Table 1: Summary of some properties of FlexGD and LinLog.

**Proof** If we scale the layout by $\frac{k_3}{k_2}$, the energy of $U_{new}$ is:

$$U_{new} = \sum_{\{u,v\}\in E} \frac{k_1 k_3}{k_2}\|p_u - p_v\| + \sum_{\{u,v\}\in V^{(2)}} \frac{k_3 k_2}{k_2}\|p_u - p_v\| - k_3 \ln \frac{k_3}{k_2}\|p_u - p_v\|.$$

This can be rewritten as:

$$U_{new} = k_3(\frac{k_1}{k_2} \sum_{\{u,v\}\in E} \|p_u - p_v\| + \sum_{\{u,v\}\in V^{(2)}} \|p_u - p_v\| - \ln\|p_u - p_v\|) - \sum_{V^{(2)}} k_3 \ln \frac{k_3}{k_2}.$$

Since $k_3$ is positive and $\sum_{V^{(2)}} k_3 \ln \frac{k_3}{k_2}$ is a constant, the minimum of this function is the same as the minimum of

$$\frac{k_1}{k_2} \sum_{\{u,v\}\in E} \|p_u - p_v\| + \sum_{\{u,v\}\in V^{(2)}} \|p_u - p_v\| - \ln\|p_u - p_v\|.$$

∎

This theorem states that the effect of $k_3$ is limited to scaling, having merely a zooming effect. Furthermore, apart from its scaling effect, $k_2$ only changes the abstraction constant to $\frac{k_1}{k_2}$. Since every positive real value can be chosen directly as the abstraction constant, adding $k_2$ has no mathematical advantage. Table 1 compares the properties of FlexGD with those of LinLog.

### 3.2 Distance Interpretability in 1+ Dimensional FlexGD Bipartition Layouts

In this section we explain the approximate generalizability of Theorem 3.2 to 1+ dimensions. The following theorem holds exactly for layouts with any number of dimensions:

**Theorem 3.4** *Let $p$ be a $D$-dimensional drawing of $G = (V, E)$ with minimum FlexGD energy. Let $(S_1, S_2)$ be a bipartition of the drawing space by any hyperplane $H$ defined by $\sum_{i\in I} a_i x_i = b$, $I \in \binom{\{1,\cdots,D\}}{D-1}$. If $(V_1, V_2)$ is a bipartition of vertices in a way that $\forall u \in V_1 : p_u \in S_1$, and $\forall v \in V_2 : p_v \in S_2$, that is $\forall u \in V_1 : \sum_i a_i x_i^u < b$ and $\forall v \in V_2 : \sum_i a_i x_i^v > b$, then:*

$$\sum_{\{u,v\}\in E_{V_1\times V_2}} k\frac{\sum_{i=1}^{D}(x_i^u - x_i^v)}{\|p_u - p_v\|} + \sum_{\{u,v\}\in V_1\times V_2} \frac{\sum_{i=1}^{D}(x_i^u - x_i^v)}{\|p_u - p_v\|} =$$

$$\sum_{\{u,v\}\in V_1\times V_2} \frac{\sum_{i=1}^{D}(x_i^u - x_i^v)}{\|p_u - p_v\|^2}.$$

**Proof** If we add some distance vector $\vec{d} = (d_1, \ldots, d_D)$ to all vertices in $V_1$ in a way that none of them enter $S_2$, i.e. $\forall u \in V_1 : \sum_i a_i(x_i^u + d_i) < b$, the energy of the new drawing is:

$$\begin{aligned} U^{new} &= \sum_{\{u,v\} \in E_{V_1^{(2)}} \cup E_{V_2^{(2)}}} k\|p_u - p_v\| + \\ &\sum_{\{u,v\} \in V_1^2 \cup V_2^2} (\|p_u - p_v\| - \ln \|p_u - p_v\|) + \sum_{\{u,v\} \in E_{V_1 \times V_2}} k\sqrt{\sum_{i=1}^{D} (x_i^u - x_i^v + d_i)^2} \\ &+ \sum_{\{u,v\} \in V_1 \times V_2} (\sqrt{\sum_{i=1}^{D} (x_i^u - x_i^v + d_i)^2} - \ln \sqrt{\sum_{i=1}^{D} (x_i^u - x_i^v + d_i)^2}). \end{aligned}$$

The partial derivative of this function with respect to $d_i$ is:

$$\begin{aligned} \frac{\partial U_{new}}{\partial d_i} &= \sum_{\{u,v\} \in E_{V_1 \times V_2}} k \frac{x_i^u - x_i^v + d_i}{\sqrt{\sum_{i=1}^{D} (x_i^u - x_i^v + d_i)^2}} \\ &+ \sum_{\{u,v\} \in V_1 \times V_2} (\frac{x_i^u - x_i^v + d_i}{\sqrt{\sum_{i=1}^{D} (x_i^u - x_i^v + d_i)^2}} - \frac{x_i^u - x_i^v + d_i}{\sum_{i=1}^{D} (x_i^u - x_i^v + d_i)^2}). \end{aligned}$$

Since $p$ is a layout with minimum FlexGD energy, the application of any non zero vector $\vec{d}$ must result in increase of energy. Then, the gradient vector of $U_{new}$ must be zero when $\vec{d} = 0$, that is $\forall d_i : \frac{\partial U_{new}}{\partial d_i} = 0$. Hence $\sum_{i=1}^{D} \frac{\partial U_{new}}{\partial d_i} = 0$.

$$\begin{aligned} \sum_{i=1}^{D} \frac{\partial U_{new}}{\partial d_i} &= \sum_{\{u,v\} \in E_{V_1 \times V_2}} k \frac{\sum_{i=1}^{D} (x_i^u - x_i^v)}{\sqrt{\sum_{i=1}^{D} (x_i^u - x_i^v)^2}} \\ &+ \sum_{\{u,v\} \in V_1 \times V_2} (\frac{\sum_{i=1}^{D} (x_i^u - x_i^v)}{\sqrt{\sum_{i=1}^{D} (x_i^u - x_i^v)^2}} - \frac{\sum_{i=1}^{D} (x_i^u - x_i^v)}{(x_i^u - x_i^v)^2}) = 0. \end{aligned}$$

∎

For D-dimensional layouts of graphs, clusterizable to some extent, Theorem 3.4 leads to the following useful corollary:

**Corollary 3.1** *Let $p$ be a D-dimensional drawing of $G = (V, E)$ with minimum FlexGD energy. For any non-scaling linear transformation of the coordinate system*[2] *that partitions the vertices into $(V_1, V_2)$ in a way that in the new coordinate system $\forall u \in V_1, v \in V_2, 1 \leq i \leq D : x_i^u < x_i^v$:* [3]

$$\begin{aligned} \sum_{\{u,v\} \in E_{V_1 \times V_2}} k \frac{\|p_u - p_v\|_{Man}}{\|p_u - p_v\|} + \sum_{\{u,v\} \in V_1 \times V_2} \frac{\|p_u - p_v\|_{Man}}{\|p_u - p_v\|} = \\ \sum_{\{u,v\} \in V_1 \times V_2} \frac{\|p_u - p_v\|_{Man}}{\|p_u - p_v\|^2}, \end{aligned}$$

*where $\|p_u - p_v\|_{Man} = \sum_{i=1}^{D} |x_i^u - x_i^v|$ is the* Manhattan distance *between $p_u$ and $p_v$.*

[2] This causes no change to the energy of the system.

[3] Notice such transformation does not exist for the layouts of all graphs.

We know from Theorem 3.1 that abstraction constant may be increased to shorten edges as much as necessary. Hence, *provided the graph is clusterizable into two convex subgraphs*, we can increase $k$ to decrease the diameter of clusters (i.e. the maximum Euclidean distance between pairs of a cluster) compared to their distance as much as desired. If the clusters are concentrated and far from each other, the Euclidean and Manhattan distance become almost equal. In this case we can state:

**Corollary 3.2** *Let $p$ be a $D$-dimensional drawing of $G = (V, E)$ with minimum FlexGD energy. If a bipartition of vertices $(V_1, V_2)$ exists in a way that the diameter of $V_1$ and $V_2$ is small compared to their distance, then:*

$$harm(V_1 \times V_2, p^0) \approx \frac{1}{1 + k * density(V_1, V_2)}.$$

**Proof** Putting $\|p_u - p_v\| \approx \|p_u - p_v\|_{Man}$ into Corollary 3.1, we obtain:

$$k \left|E_{V_1 \times V_2}\right| + |V_1 \times V_2| \approx \sum_{\{u,v\} \in V_1 \times V_2} \frac{1}{\|p_u - p_v\|}.$$

Replacing the right side by $\frac{|V_1 \times V_2|}{harm(V_1 \times V_2, p^0)}$, the result is derived. ∎

# 4 Minimization of the FlexGD Energy Function

The minimum of the FlexGD energy function can be found using an iterative algorithm. In each iteration, the net force exerted on each vertex is computed. The vertices are then moved in the direction of this force by some step length until the layout change is less than some tolerance. Previous works [7, 9, 1] apply a force-directed algorithm with an adaptive step length. The algorithm starts with some initial global step length and decreases it per cycle. This scheme works well if the edge length distribution is not very uneven. In FlexGD this assumption is violated, specifically if a large level of abstraction is applied, i.e. $k$ is set to a large value. As a result, no same value of step length is suitable for all vertices. We treat this issue by applying vertex specific adaptive step length. Our force algorithm is given in Algorithm 1. In each iteration, current direction of the net force exerted on a vertex is compared with its previous direction. The step length is then increased or decreased proportional to the change of direction. This scheme copes well with the non-uniform distribution of edge lengths, but has occasionally higher running time than previous simpler algorithms.

Since the force algorithm works on top of a multilevel coarsening scheme (see Section 4.2), it is important that the initial step length is small enough, otherwise the usefulness of the layout resulted from the previous coarsening level is destroyed. We compute a graph-specific initial step length with an empirical equation improvised from the following theorem:

**Theorem 4.1** *If $p^0$ is a drawing of a graph $G = (V, E)$ with minimum FlexGD energy then:*

$$k \sum_{\{u,v\} \in E} \|p_u - p_v\| + \sum_{\{u,v\} \in V^{(2)}} \|p_u - p_v\| = \left|V^{(2)}\right|.$$

**Algorithm 1** ForceAlgorithm($G$,$k$,$\epsilon$)

$ratio \leftarrow 2.0\ (> 1+\epsilon)$, $\gamma \leftarrow 0.5$, $d_0 \leftarrow 0.00000001$ (small float), $s_0 \leftarrow \frac{|V|^2}{k(k|E|+|V|^2)}$
$\forall v \in V : s_u \leftarrow s_0, \vec{f_u} \leftarrow random$
**while** (ratio $> 1+\epsilon$) **do**
  d $\leftarrow 0.0$
  **for** $u \in V$ **do**
    $\vec{p_u^0} \leftarrow \vec{p}_u$, $\vec{f_u^0} \leftarrow \vec{f_u}$
    $\forall v \in V, v \leftrightarrow u : \vec{f_u} = \vec{f_u} + \vec{f_a}(u,v)$ % compute attractive forces of edges.
    $\forall v \in V, v \neq u, \vec{f_u} = \vec{f_u} + \vec{f_r}(u,v)$ % compute pairwise forces.
    $s_u = s_u + s_u * (\frac{\vec{f_u}}{\|\vec{f_u}\|} \cdot \frac{\vec{f_u^0}}{\|\vec{f_u^0}\|}) * \gamma$ % modify the step length proportional to the change of direction of the net force.
    $\vec{p}_u = \vec{p}_u + s_u * (\frac{\vec{f_u}}{\|\vec{f_u}\|})$ % move the active vertex.
    $d = d + \left\|\vec{p}_u - \vec{p_u^0}\right\|$
  **end for**
  $ratio \leftarrow \max(\frac{d}{d_0}, \frac{d_0}{d})$ % halt when the change of layout is negligible.
  $d_0 \leftarrow d$
**end while**

**Proof** Suppose $p^0$ is a drawing with minimum FlexGD energy. If we multiply all coordinates in $p^0$ by $d \in R$, the energy of the system is:

$$U(d,p^0) = \sum_{\{u,v\}\in E} d\|p_u - p_v\| + \sum_{\{u,v\}\in V^{(2)}} d\|p_u - p_v\| - \ln d\|p_u - p_v\|.$$

Since $p^0$ is a drawing with minimum energy, this equation has a minimum at $d = 1$, that is:

$$\begin{aligned} U'(d,p^0) &= \sum_{\{u,v\}\in E} \|p_u - p_v\| + \sum_{\{u,v\}\in V^{(2)}} \|p_u - p_v\| - \frac{\left|V^{(2)}\right|}{d}, \\ U'(d=1,p^0) &= \sum_{\{u,v\}\in E} \|p_u - p_v\| + \sum_{\{u,v\}\in V^{(2)}} \|p_u - p_v\| - \left|V^{(2)}\right| = 0. \end{aligned}$$

∎

We can rewrite the left side of this theorem in the form of $k\,|E|\,\bar{e} + |V|^2\,\bar{d} = (k\,|E| + |V|^2)l$, where $\bar{e}$ is the mean edge length and $\bar{d}$ the mean distance between every two vertices. Hence, $l = \frac{|V|^2}{(k|E|+|V|^2)}$ is a value between $\bar{e}$ and $\bar{d}$. Dividing further by $k$ gives some value of the order of the mean edge length. Setting the initial step length to $l/k$ always led to satisfactory results for the graphs we tried. Further modification of the step length is done on a per vertex base through the adaptive step length scheme.

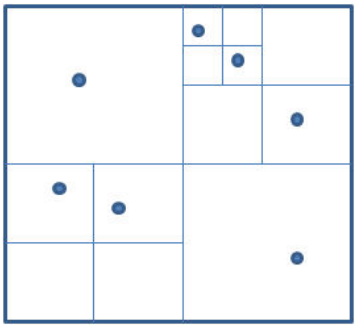


Figure 3: Formation of the quadTree in the Barnes and Hut algorithm.

### 4.1 The Barnes and Hut Algorithm

A direct application of Algorithm 1 is not effective for large graphs, because it is complex as $O(|E| + |V|^2)$. The complexity may be decreased to $O(|E| + |V| \log |V|)$ using the Barnes and Hut scheme [12]. The idea is to speed up the calculation of pairwise forces by regrouping the nearby vertices and computing their force as a whole, provided their center of mass is far enough from the active vertex. This is done through recursively assigning the vertices to the nodes of a *quadTree*. A quadTree is a tree where each node has at most four children. There is some mass, a center and a square area associated with each node. Vertices are inserted one by one into the tree, starting from the root node. If the current node already contains a vertex, the corresponding area is divided into four squares known as *quads*. The new vertex is consequently inserted into the right quad, and the mass and center of the parent node are updated as follows:

$$\begin{aligned} \vec{x} &= (m \cdot \vec{x} + m_i \cdot \vec{x}_i)/(m + m_i), \\ m &= m + m_i, \end{aligned}$$

where $m$ and $\vec{x}$ are the mass and the center of the node, and $m_i$ and $\vec{x}_i$ the mass and coordinates of the inserted vertex, respectively. This procedure is iterated until all vertices are inserted, and there is only zero or one vertex in each external node. Figure 3 illustrates the formation of a quadTree. We form the quadTree once in the beginning of each execution cycle. When computing the pairwise forces, all vertices of a node are approximated as a single vertex if $s/d < \theta$, where $s$ is the width of the area represented by the quad of the corresponding node, and $d$ the distance of the active vertex from the quad center. In our setting we set $\theta = 0.5$. The same scheme can be used for 3D drawings replacing the quadTree by an *ocTree*. An ocTree is a tree where each node has at most 8 children.

### 4.2 A Multilevel Algorithm

The force algorithm (Algorithm 1) finds a *local* minimum of the energy function. Consequently, it is not very probable that it results in satisfactory drawings of large graphs as their energy functions have many local minima. In addition, too many cycles are needed to lay

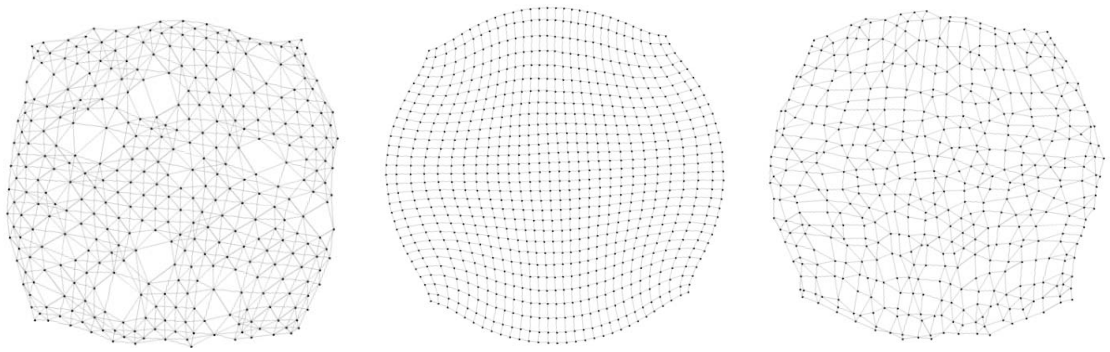
a stable drawing out of the initial configuration. Multilevel algorithms can greatly alleviate these problems by consecutively coarsening a graph $G_0$ into coarser graphs $G_1, ..., G_n$. The layout of the coarsest graph is computed cheaply as it is very small. The computed coordinates are then prolongated to the finer graph. The finer graph usually needs less modifications as it is already in a rather good shape.

Edge Collapsing (EC) [13, 14, 9] is the most widely-used coarsening strategy in graph drawing. This method works based on a Multiple Independent Edge Set (MIES). An independent edge set is a set of edges no two of which are adjacent to the same vertex. It is maximal, if adding any new edge to the set destroys the independence. MIES can be computed through a greedy algorithm. Namely, all vertices are unmatched in the beginning. An unmatched vertex is picked up at random, and is merged with one of its unmatched neighbors. As a result of this merging, the edge between them is collapsed. Both merged vertices are then marked as matched. If the vertex has no neighbors (i.e. it is disconnected from the rest of the graph), it is marked as matched without being merged. The algorithm iterates until no vertex remains unmatched.

Each vertex of the coarser graph has a weight equal to the sum of the weights of the corresponding vertices in the finer graph. The edge weights of the coarser graph are initially the same as in the finer graph. If there are more than one edge between the corresponding vertices in the finer graph (as there can be up to four of them), the edge weight in the coarser graph will be the sum of the weights of the corresponding edges. Walshaw [7] matches each vertex with the neighbor having the smallest vertex weight, keeping the weight of vertices in the coarser graph as uniform as possible (Uniform Edge Collapsing). In [9], Heavy Edge Collapsing (HEC) is used, that is, the neighbor corresponding to the heaviest edge is merged. We implemented both Heavy Edge Collapsing (HEC) and Uniform Edge Collapsing (UEC). For our model they had almost the same results.

The success of a coarsening scheme depends on the graph topology. For example, we observed that for graphs with hollow topology, EC has difficulties escaping the local minima. In addition, for some graphs the coarsening is very slow, i.e. the number of vertices in the coarser graph is close to the number of vertices in the finer graph. Consequently, we followed [9] by adapting an alternative coarsening strategy based on Multiple Independent Vertex Set (MIVS). A multiple independent vertex set is a set of vertices no two of which are directly connected by an edge. It is maximal, if adding any new vertex to the set leads to violation of the independence condition. When coarsening with MIVS, an edge is added between two vertices of the coarser graph, if the distance apart between the corresponding vertices of the finer graph is no more than three. MIVS coarsens more aggressively than EC, that is, the number of vertices in the coarser graph is much smaller (usually less than 50%) than the number of vertices in the finer graph. The drawback of MIVS is that the number of edges in the coarser graph is sometimes very high. This issue increases the memory and time complexity of the algorithm. Figure 4 shows the result of coarsening a 30 by 30 grid through MIVS and EC coarsening strategies.

In [9], edges are computed using the Galrekin product of the prolongation matrix $P$ and the adjacency matrix of the finer graph $A_f$ (see [15] for more details). The Galerkin product is defined as $P^T A_f P$. We found that computing this product is expensive for large graphs. In



(a) MIVS coarsening (b) grid $30 \times 30$ (c) EC coarsening

Figure 4: A 30 by 30 grid coarsened by different strategies.

our implementation, we first connect all the vertices in the MIVS, if their graph theoretical distance is 2. For distances equal to 3, we iterate on the edges, and for those whose neither of their end points is in the MIVS, check the neighbors of their end points. An edge is added between the two MIVS neighbors of the end points, if there is already no edge between them. This strategy works faster than computing the Galerkin product.

The prolongation of coordinates from the coarser graph to the finer graph is done as follows. If the coarser graph is issue of EC, each vertex of the finer graph corresponds to exactly one vertex of the coarser graph. In this case, the coordinates of the coarser graph are attributed directly to the corresponding vertices of the finer graph. If the coarser graph rises from MIVS, each vertex of the finer graph is either in the multiple independent vertex set or has at least one neighbor in that. In the former case, the coordinates are taken directly from the corresponding vertex of the coarser graph. In the latter, the coordinates are computed as the mean of the coordinates of the neighbors in the multiple independent vertex set. Since some vertices may have the same coordinates in the fine graph, we add some small random displacement to set them apart.

We implemented a hybrid coarsening algorithm. Our default strategy is HEC. In few cases where the result of HEC was not good enough, MIVS was used. We stop coarsening if one of the followings happens. Firstly, the level of coarsening is more than a predefined threshold. In our setting we do not coarse more than 12 levels. Secondly, the coarsening ratio is too high. This ratio is defined as the number of connected vertices in the coarser graph by the number of connected vertices in the finer graph. We set this threshold to 0.9. Finally, the number of the remaining connected vertices is less than a minimum. We set this to 10.

The execution time of the FlexGD algorithm is given in Table 2. Most graphs are taken from the University of Florida Sparse Matrix Collection [16]. FlexGD reveals the structure of a graph provided $k$ is large enough. We observed choosing $k$ as $o(\frac{\left|V^2\right|}{|E|})$ is a proper value, depending on the level of abstraction the user prefers. For graphs in the first part of Table 2, $k$ has been chosen as $o(\frac{|V|^2}{|E|})$. If the graph is disconnected, the biggest component may be

Table 2: Execution time of the FlexGD algorithm

| Graph | $\|V\|$ | $\|E\|$ | $k$ | CPU time in seconds |
|---|---|---|---|---|
| grid30by30 | 900 | 1740 | 30 | 11[a] |
| grid50by50 | 2500 | 4900 | 50 | 54 |
| grid100by100 | 10000 | 19800 | 100 | 163 |
| jagmesh1 | 936 | 3600 | 300 | 8 |
| jagmesh8 | 1141 | 4303 | 300 | 5 |
| plskz362 | 362 | 880 | 100 | 3 |
| bcsstm07 | 420 | 3836 | 50 | 3 |
| bcsstk24 | 3652 | 81736 | 200 | 54 |
| G12 | 800 | 1600 | 500 | 23 |
| G49 | 3000 | 6000 | 3000 | 160 |
| utm1700b | 1700 | 21509 | 200 | 46 |
| utm3069 | 3060 | 42211 | 300 | 87 |
| 3D28984Tetra | 29984 | 599170 | 1500 | 390 |
| can838 | 838 | 5424 | 150 | 9 |
| cegb2919 | 2919 | 162201 | 50 | 44 |
| nasa1824 | 1824 | 20516 | 200 | 22 |
| Alemder | 6245 | 24413 | 1500 | 139 |
| rdist3a | 2398 | 61896 | 100 | 33 |
| web-NotreDame | 325729 | 1497134 | 1000 | 3142[b] |
| webbase-1M | 1000005 | 3105536 | 10000 | 22352 |

[a] Times are measured on a 2.4GHz Core2 Duo.
[b] Times are measured on a 2.5GHz Xeon E5420.

considered. For very sparse graphs, where vertex degree distribution is very uneven, smaller values of $k$ can also be used. Such graphs are generally issue of applications like social networks or web graphs. They are in the second part of Table 2. For graphs containing up to some tens of thousands of vertices, the execution time is a few minutes. This is a reasonable time considering the high quality of the layouts.

It is also worth mentioning the running time of the algorithm increases with $k$. The reason is that higher values of $k$, put connected vertices closer to each other. Consequently, the Barnes and Hut algorithm divides the space into smaller quads, meaning the quadTree becomes bigger. In addition, the force algorithm needs more iterations, because the layout must be refined in smaller distances necessitating smaller values of tolerance. Figure 5 compares the FlexGD layouts of a few graphs with the layouts of some other force-directed algorithms. The symmetries are shown well in the FlexGD layouts and the frontiers are decent. The distribution of vertices in FlexGD layouts is more uniform than LinLog layouts. For example, the frontiers of the LinLog grid layout are denser than its center. For smaller values of $k$, the FlexGD layouts are more similar to the layouts of conventional models, while for larger values of $k$, they are closer to the LinLog layouts.

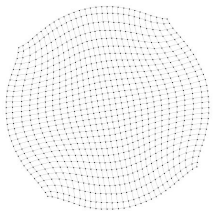
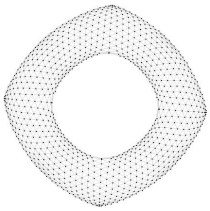
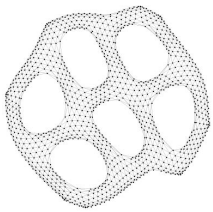
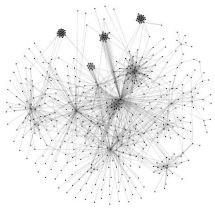
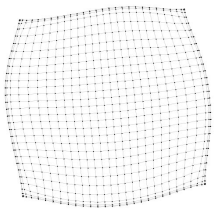
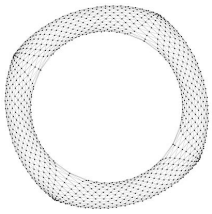
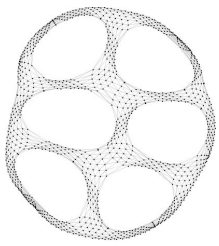
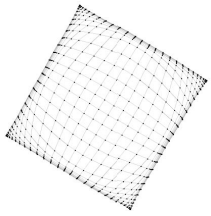
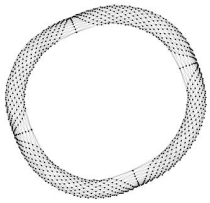
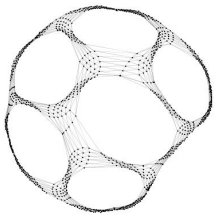
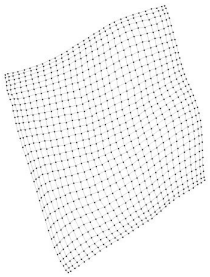
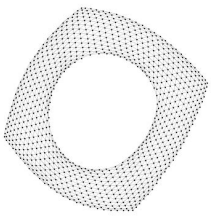
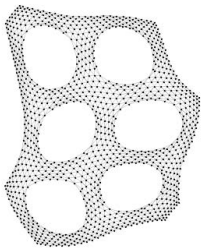
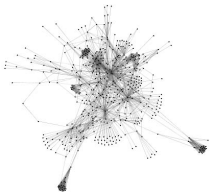
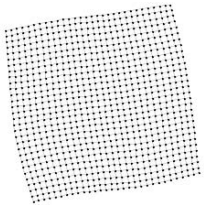
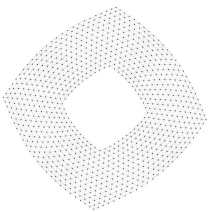
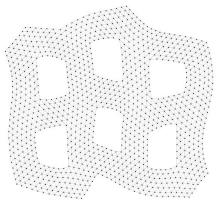
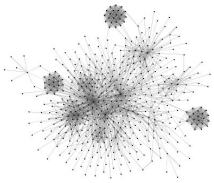
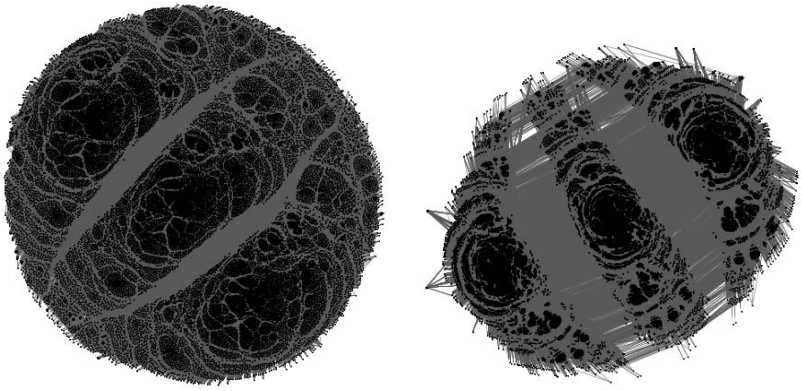
| Model | grid 30by30 | jagmesh1 | jagmesh8 | Harvard500 |
|---|---|---|---|---|
| FlexGD (1) |  (a) $k = 30$ |  (b) $k = 600$ |  (c) $k = 300$ |  (d) $k = 20$ |
| FlexGD (2) |  (e) $k = 500$ |  (f) $k = 1800$ |  (g) $k = 900$ |  (h) $k = 100$ |
| LinLog [4] |  (i) |  (j) |  (k) |  (l) |
| Spring-Electrical [1] |  (m) |  (n) |  (o) |  (p) |
| Davidson&Harel [2] |  (q) |  (r) |  (s) |  (t) |

Figure 5: Comparison of the FlexGD layouts with other models.



(a) FlexGD model, $k = 200$ (b) FlexGD model, $k = 1000$

Figure 6: gupta1 graph

## 5 Related Work

Energy models [1, 5, 3, 4, 2] are the most popular technique in graph drawing. The initial versions suffered from high running time. The Barnes and Hut algorithm [12], initiated to graph drawing by Tunkelang [17], alleviated this problem. With the advent of multilevel algorithms [7, 8, 9], force-directed algorithms became rather efficient both from the computational and quality points of view. Another line of work which has recently become popular is spectral graph drawing [18, 19, 20]. The idea is to use the smallest/largest eigenvectors of the Laplacian/Adjacency matrix of the graph as the drawing. Despite very low execution time, their layout quality is usually lower than force-directed algorithms.

Apart from its direct links to the LinLog model, our model has also similarities with the Binary Stress Model of Koren and Çivril [6]. The Binary Stress Model bridges the Stress [3] and the Spring-Electrical [1] models. To the best of our knowledge, this is the only force model capable of drawing disconnected graphs without any extra processing steps. It has also the property of abstractability. However, apart from its major differences from FlexGD both in the energy function and in the optimization technique, we believe that, as posited by Theorem 3.1 and Theorem 3.2, linear attraction and linear-logarithmic pairwise energy functions of FlexGD makes it more efficient in uniform vertex distribution and in cluster visualization. For example, the frontiers of the Binary Stress layouts are denser than the center. The authors add a random perturbation improving occasionally this drawback. Such a problem does not arise with FlexGD, as the linear-logarithmic shape of the pairwise forces result in perfectly even distribution of the vertices over the drawing area (see Figure 1).

## 6 Discussion

FlexGD abstracts the graph structure in a desired level, filling optimally a circular drawing area. Consequently, tweaking the parameter, a user has more chance to obtain her favorite drawing. FlexGD is suitable for cluster visualization and extends this property of the LinLog

model. In general, the layout is decent in the frontiers, and the symmetries are shown well. From an applicative point of view, we examined the model on graphs arising from a wide variety of real world applications like web graphs, 2D/3D problems, structural problems, electromagnetic problems, social networks, etc. Although no single model can be claimed to have better performance on all graphs, as the suitability of a visualization model depends on the graph topology and the visualization requirements, it seems that in many cases FlexGD layouts are pleasing as much as, sometimes even more than, the layouts of previous models.

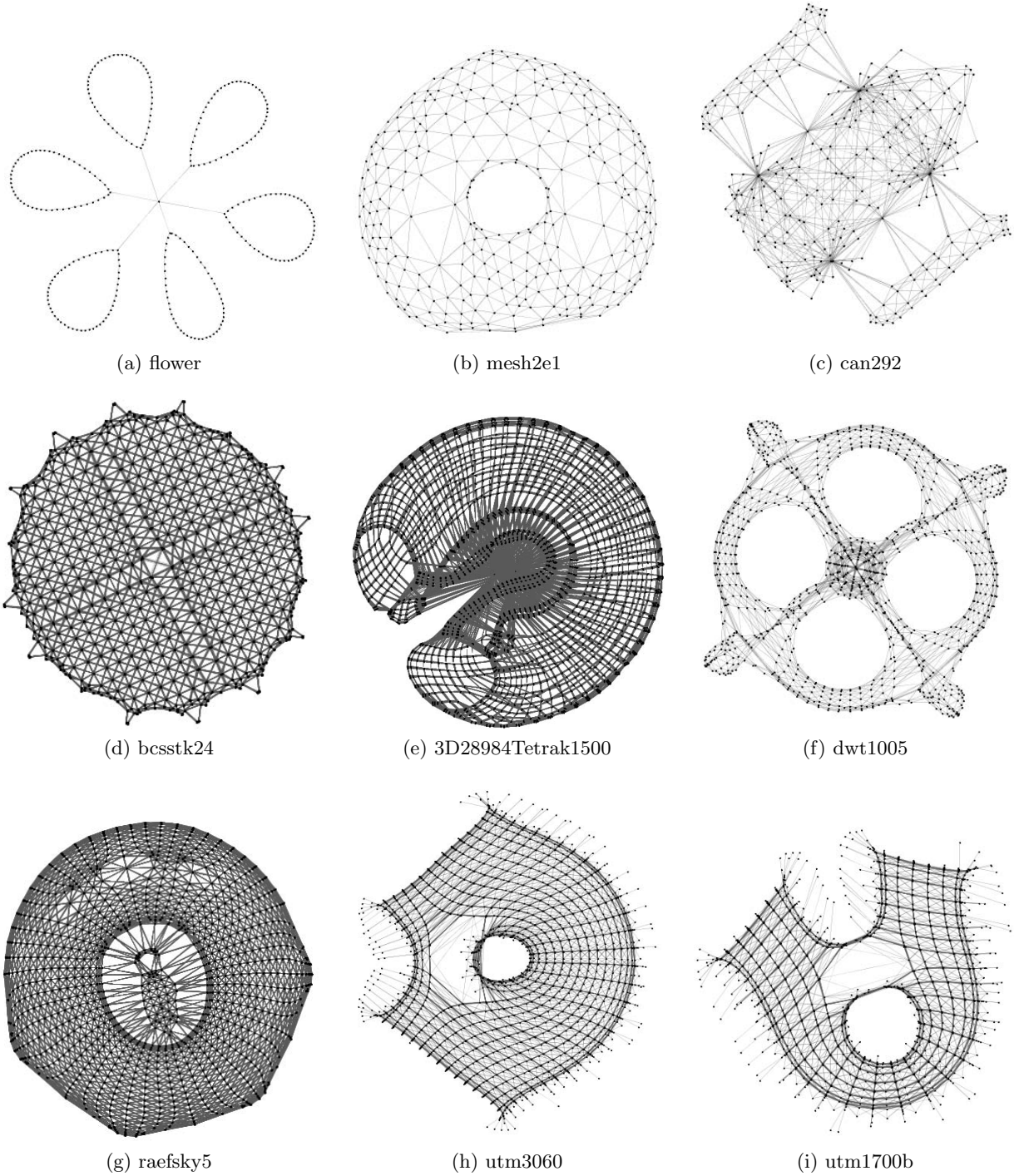
Most previous works adopt an empirical approach to validate their model. A distinguishing point of this work is to adopt a more formal approach initiated by Noack [4]. This was pretty helpful, as guided us through sophisticated parameterization of the force algorithm such that it copes well with the multilevel algorithm acting in its bottom. Furthermore, it gives a more quantitative understanding of the behavior of the model and clarifies its differences from the LinLog model. We used some vertex specific adaptive step length in the iterative force algorithm to overcome the uneven distribution of edge length in FlexGD layouts.

The FlexGD model gives a unique perspective into the community (cluster) structure of huge sparse graphs arising from domains like web applications. We believe for such graphs, drawing in the goal of visualizing the community structure is more indicative than using the conventional drawing criteria, as this latter usually results in a less informative clutter of interconnected vertices.

For some topologies, specially those containing star-shaped components, the beforementioned coarsening strategies are ineffective as the number of vertices of the coarser graph remains very close to that of the finer of graph. Hence, developing more robust coarsening schemes may be considered as a direction for future work. This problem is not particular to FlexGD, and has been reported by previous authors too [9]. Nevertheless, FlexGD can reveal the structure of some clustered graphs much better than other models without needing any more advanced multilevel scheme. An example is the Gupta/gupta1 graph from the University of Florida Sparse Matrix Collection. The authors were forced to design a new coarsening scheme in [16] to reveal the three groups of vertices in the graphs. Interestingly, FlexGD reveals these clusters easily without any coarsening phase. Figure 6 shows the FlexGD layout of this graph in two levels of abstraction.

# 7 Collection of Results

Some sample layouts of FlexGD are shown in this appendix. Most graphs are taken from the University of Florida Sparse Matrix Collection [16]. Edges and vertices of all graphs are plot with the same width and size. Further darkness of some layouts is due to the closeness of edges or vertices.

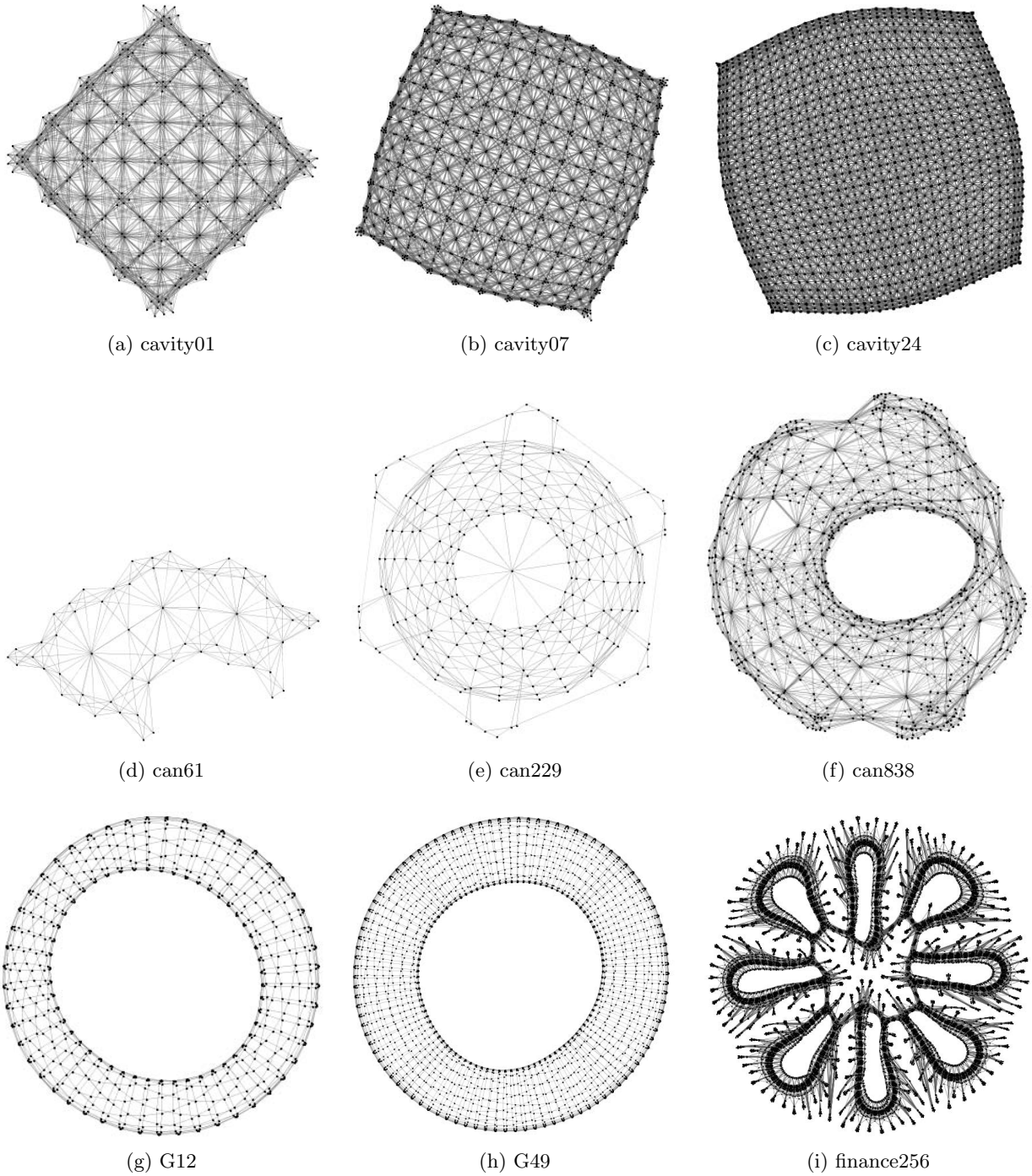


(a) flower (b) mesh2e1 (c) can292

(d) bcsstk24 (e) 3D28984Tetrak1500 (f) dwt1005

(g) raefsky5 (h) utm3060 (i) utm1700b

Figure 7: Sample Layouts of the FlexGD model.

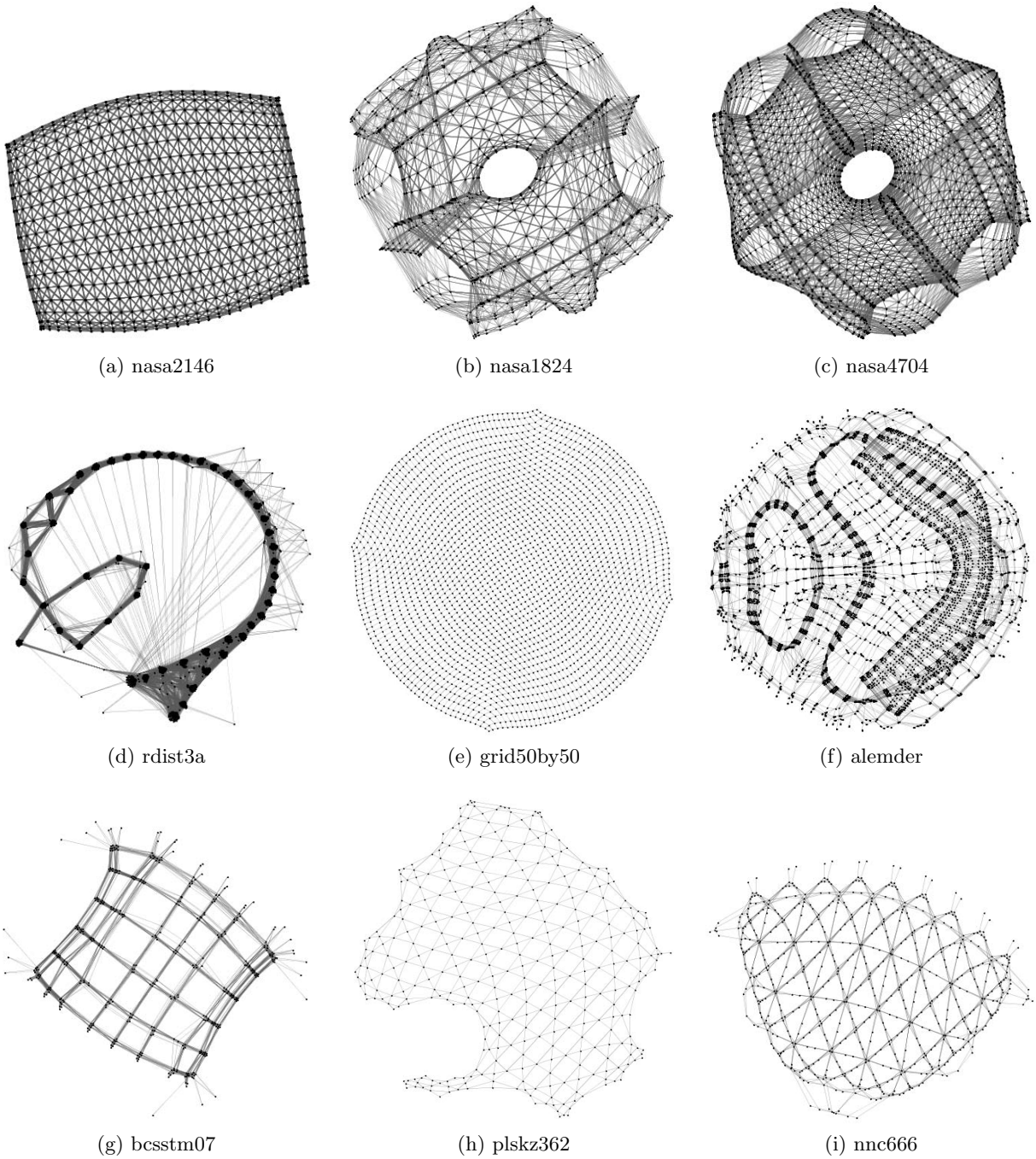


(a) cavity01 (b) cavity07 (c) cavity24

(d) can61 (e) can229 (f) can838

(g) G12 (h) G49 (i) finance256

Figure 8: Sample Layouts of the FlexGD model.

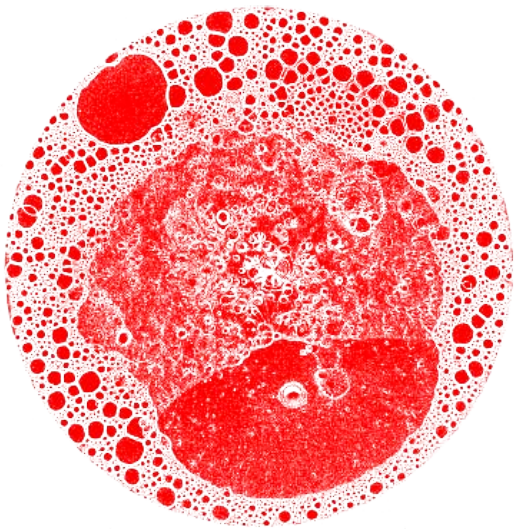


(a) nasa2146 (b) nasa1824 (c) nasa4704

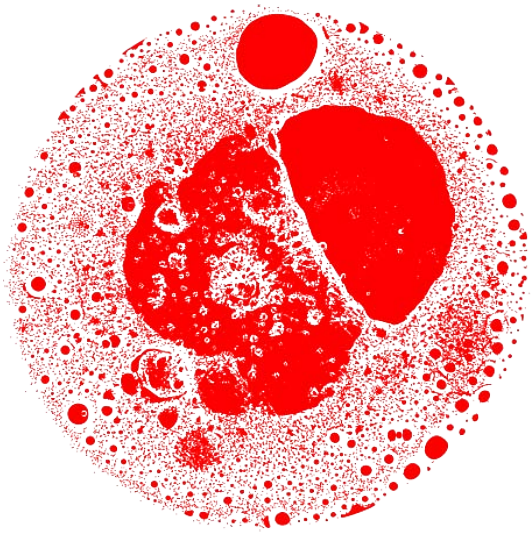
(d) rdist3a (e) grid50by50 (f) alemder

(g) bcsstm07 (h) plskz362 (i) nnc666

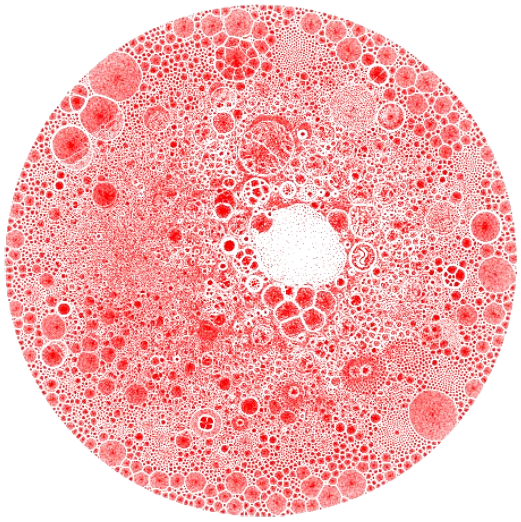
Figure 9: Sample Layouts of the FlexGD model.



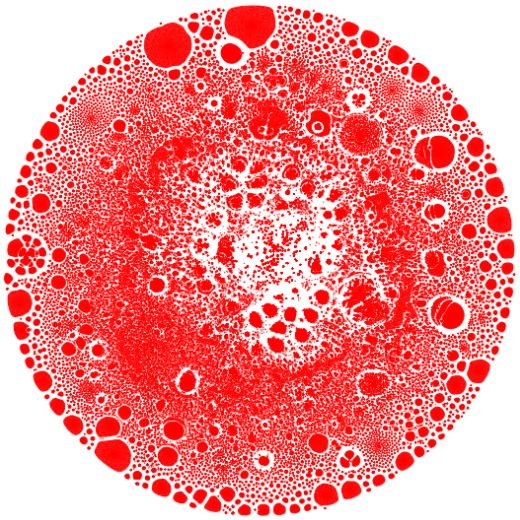
(a) Web connectivity matrix, $|V| = 1000005$, $|E| = 3105536$, $k = 10000$



(b) Web connectivity matrix, $|V| = 1000005$, $|E| = 3105536$, $k = 50000$

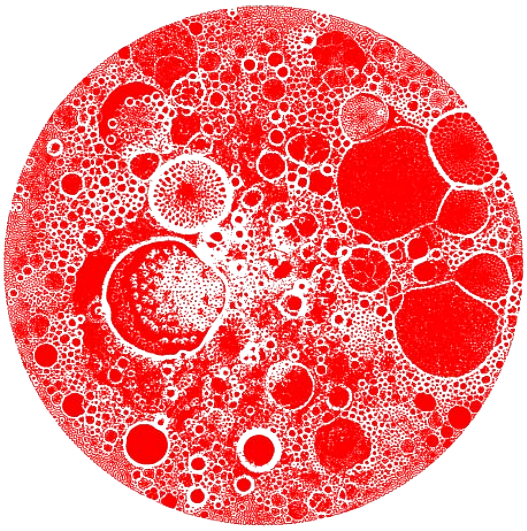


(c) Web of NotreDame University, $|V| = 325729$, $|E| = 1497134$, $k = 1000$

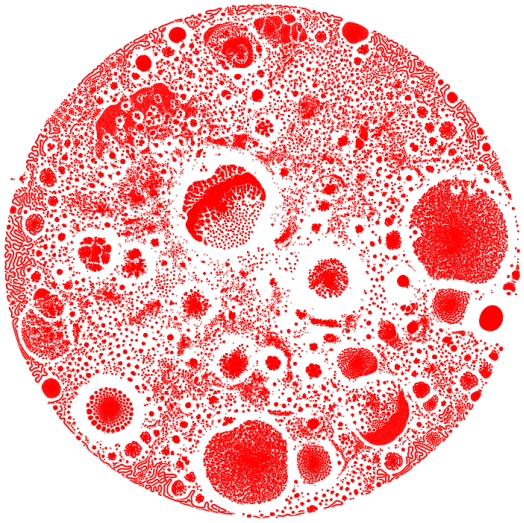


(d) Web of NotreDame University, $|V| = 325729$, $|E| = 1497134$, $k = 5000$

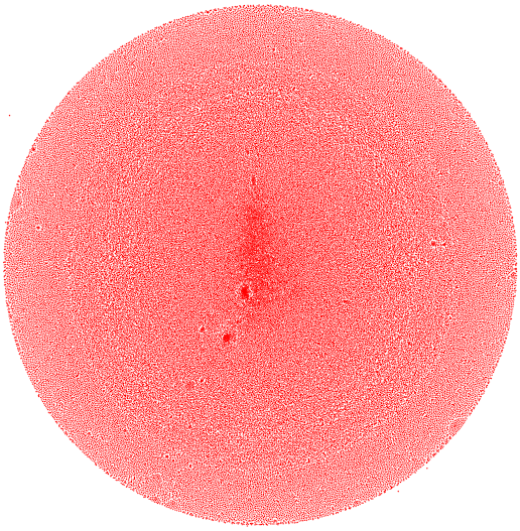
Figure 10: Sample Layouts of the FlexGD model.



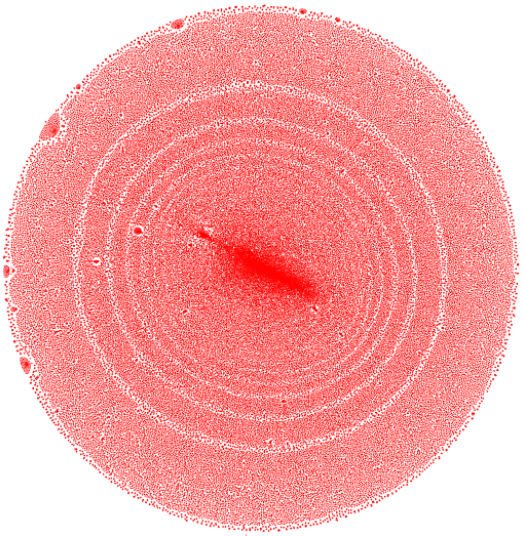
(a) Web of Stanford University, $|V| = 281903$, $|E| = 2312497$, $k = 1000$



(b) Web of Stanford University, $|V| = 281903$, $|E| = 2312497$, $k = 5000$



(c) Web of Stanford University, $|V| = 281903$, $|E| = 2312497$, $k = 200$



(d) Web of Stanford University, $|V| = 281903$, $|E| = 2312497$, $k = 1000$

Figure 11: Sample Layouts of the FlexGD model.

## References


[1] Fruchterman, T.M., Reingold, E.M.: Graph drawing by force-directed placement. In: Software-Practice and Experience. (1991) 1129–1164

[2] Davidson, R., Harel, D.: Drawing graphs nicely using simulated annealing. In: ACM Transactions on Graphics. (1996) 301–331

[3] Kamada, T., Kawai, S.: An algorithm for drawing general undirected graphs. Inf. Process. Lett. (1989) 7–15

[4] Noack, A.: Energy models for graph clustering. Journal of Graph Algorithms and Applications (2007) 453–480

[5] Eades, P.: A heuristic for graph drawing. In: Congressus Numerantium. (1984) 149–160

[6] Koren, Y., Çivril, A.: The binary stress model for graph drawing. In: Graph Drawing. (2009) 193–205

[7] Walshaw, C.: A multilevel algorithm for force-directed graph drawing. In: Graph Drawing. (2001) 31–55

[8] Harel, D., Koren, Y.: A fast multi-scale method for drawing large graphs. In: Journal of Graph Algorithms and Applications. (2002)

[9] Hu, Y.: Efficient, high-quality force-directed graph drawing. ACM Trans. Math. Softw. (2011) 1:1–1:25

[10] Luxburg, U.: A tutorial on spectral clustering. Statistics and Computing (2007) 395–416

[11] Schaeffer, S.E.: Graph clustering. Computer Science Review (2007) 27 – 64

[12] Barnes, J., Hut, P.: A hierarchical $o(nlogn)$ force-calculation algorithm. In: Nature. (1986) 446–449

[13] Gupta, A., Karypis, G., Kumar, V.: Highly scalable parallel algorithms for sparse matrix factorization. Technical report, IEEE Transactions on Parallel and Distributed Systems (1994)

[14] Walshaw, C., Everett, M.G., Cross, M.: Parallel dynamic graph partitioning for adaptive unstructured meshes. J. Parallel Distrib. Comput. (1997) 102–108

[15] Hu, Y.F., Scott, J.A.: A multilevel algorithm for wavefront reduction. SIAM J. Sci. Comput. (2001) 1352–1375

[16] Davis, T.A., Hu, Y.: The university of florida sparse matrix collection. ACM Trans. Math. Softw. (2011) 1:1–1:25

[17] Tunkelang, D.: Jiggle: Java interactive graph layout environment. In: Graph Drawing (GD). (1998) 413–422

[18] Koren, Y.: Drawing graphs by eigenvectors: theory and practice. Comput. Math. Appl. (2005) 1867–1888

[19] Çivril, A., Magdon-Ismail, M., Bocek-Rivele, E.: Ssde: fast graph drawing using sampled spectral distance embedding. In: Proceedings of the 14th international conference on Graph drawing. 30–41

[20] Puppe, T.: Spectral Graph Drawing: A Survey. VDM Verlag (2008)