HAL
open science

FlexGD : A Flexible Force-directed Model for Graph Drawing

Anne-Marie Kermarrec, Afshin Moin

To cite this version:

Anne-Marie Kermarrec, Afshin Moin. FlexGD : A Flexible Force-directed Model for Graph Drawing. [Research Report] 2012, pp.13. hal-00679574v2

HAL Id: hal-00679574

https://inria.hal.science/hal-00679574v2

Submitted on 29 Nov 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

FlexGD : A Flexible Force-directed Model for Graph Drawing*†

Anne-Marie Kermarrec
INRIA Rennes Bretagne Atlantique
akermarrec@inria.fr

Afshin Moin‡
INRIA Rennes Bretagne Atlantique
afshin2927@yahoo.com

We propose *FlexGD*, a 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 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 [5, 2, 13, 19, 4, 16] are one popular approach to graph drawing. 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.

Attractive and repulsive forces (or equivalently the energy function) are defined in the goal of meeting some aesthetic criteria of drawing like uniform edge length and minimum edge crossing. For example, the Spring-Electrical model [5] enforces uniform edge length while the Stress model [13] estimates the Euclidean distance between vertices on the layout with their graph-theoretic distance. Recently, Noack [19] has investigated the influence of the shape of attraction and repulsion energies[1] on the clustering properties of a model. The author shows that *Lin*ear attraction and *Log*arithmic repulsion energies are better for cluster visualization than previously considered energy functions. The *LinLog* model is consequently proposed, and some of its properties are derived.

*This work is supported by the ERC Starting Grant GOSSPLE number 204742.

†For all references to this report, please cite the conference version to appear in PacificVis2013.

‡Corresponding author. Following the team convention, names are in alphabetical order.

[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.

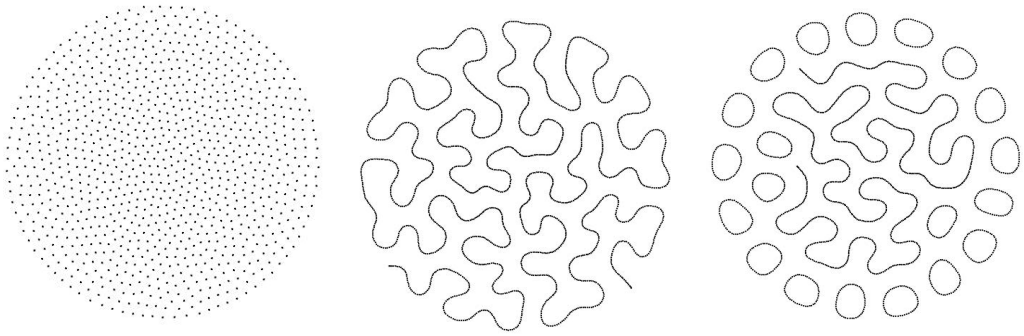
In this paper, we suggest FlexGD, a Flexible force-directed algorithm for Graph Drawing. FlexGD draws graphs according to the two criteria of uniform vertex distribution and structure abstraction. The model is flexible, in that it is parameterized to be biasable towards any of the two drawing criteria, 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, the drawing area is filled optimally and the layout looks pleasing in the frontiers. Second, it upgrades the cluster visualization property of LinLog to *abstractable* cluster visualization, i.e. the user can decide upon the density of the clusters and to what extent they are set apart. FlexGD is also capable of drawing disconnected graphs while most of the previous models have difficulties with their handling.

Existing minimization algorithms are in general designed for energy functions creating a rather uniform edge length distribution. FlexGD (like LinLog) may give rise to layouts with very uneven edge lengths. To overcome this challenge we suggest a sophisticated multilevel algorithm with exact parameterization methods to minimize the FlexGD energy function. A slightly modified version of this algorithm can be used to find LinLog minimum energy layouts. This is particularly important as no algorithm is proposed in [19] for finding the LinLog layouts of large graphs. We present FlexGD layouts of some large real datasets to illustrate that the algorithm can generate quality layouts in a wide range of abstraction, satisfying different user preferences. At the end, some properties of FlexGD minimum energy layouts are analytically derived.

2 Related Work

Force-directed algorithms [5, 2, 13, 19, 4, 16] have been in use for years and many derivations of them are applied in industry and academia. In this section, we mention a number of important works and approaches to graph drawing, although the literature we discuss is by no means comprehensive.

The initial versions of force-directed models suffered from high running time. A survey of these models is available in [23]. The main source of complexity is the computation of pairwise forces. The Barnes and Hut algorithm [1] alleviated this problem by estimating the repulsion force of close vertices which are sufficiently far from the active vertex as a whole. With the advent of the multilevel algorithms [25, 8, 10, 15], force-directed algorithms became rather efficient both from the computational and the quality points of view. Harel and Koren suggest a different approach called High Dimensional Embedding (HDE) [9] for graph drawing. HDE relates the Euclidean distance between the vertices to their graph-theoretic distance in a high-dimensional space. It then projects the vertices back to the 2-dimensional space using Principal Component Analysis (PCA). Another line of work which has recently become popular is spectral graph drawing [15, 14, 20]. The idea is to use the (generalized) eigenvectors of the Laplacian/Adjacency matrix of the graph as the drawing. Algebraic multi-scale Computation of Eigenvectors (ACE) [15] uses the smallest eigenvectors of the Laplacian as the drawing. The algorithm is combined with 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.

multilevel coarsening scheme to obtain a sophisticated initialization of the eigenvectors. Despite very low execution time, the quality of HDE and spectral layouts is usually lower than the quality of layouts computed by force-directed algorithms [7].

We are motivated to suggest FlexGD as a robust model to graph drawing because it can generate a spectrum of layouts encompassing from conventional force-directed layouts to clustered layouts of LinLog. Comparing with each individual model, FlexGD layouts seem to be nicer and more symmetric. Furthermore, in huge irregular sparse graphs, FlexGD reveals the community structure better than other models. This is helpful for visualization of graphs arising from applications like web connectivity and email networks.

Beside links to LinLog, our model has also similarities with the Binary Stress Model of Koren and Çivril [16]. The Binary Stress Model bridges the Stress [13] and the Spring-Electrical [5] models. 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 it will be shown in the paper, 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).

3 Definitions and Notations

A d-dimensional layout p of a graph $G = (V, E)$ is a mapping of vertices into the Euclidean space $p : V \longrightarrow R^d$, where every $u \in V$ 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 [17, 22]. The *cut* between two disjoint vertex sets V_1 and V_2 is defined as $cut[V_1, V_2] = |E_{V_1 \times V_2}|$, where $E_{V_1 \times V_2}$ represents the set of edges between V_1 and V_2. Using the cut as a coupling measure 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 penalize 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, geometric and harmonic mean are the most popular definitions in the literature to measure the mean distance between a set of vertices. For layout p and $F \subset V^{(2)}$, where $V^{(2)}$ represents the set of vertex pairs, we represent arithmetic, geometric and harmonic mean of F on p by $arith(F, p)$, $geo(F, p)$ and $harm(F, p)$ respectively. 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 an 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. The *weighted arithmetic mean* is defined as:

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

where E_f is the set of edges over F.

4 FlexGD Energy Function

For layout p of a graph $G = (V, E)$, many of the known energy models [19, 5] have the following form:

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

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 for layout p is defined as:

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

For layout p and the *abstraction constant* k, the FlexGD energy function is defined as:

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

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 [19, 5], g is monotonically 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 graph is uniformly packed within a circular drawing area regardless of its connectivity. The attractive force of the edges and the pairwise force exerted on a vertex u from another vertex v are:

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

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

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

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 (see [10] for example). 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. 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. First, 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, as the clusters get denser for higher k. Second, disconnected vertices are put towards the frontiers. This prevents them from adding visual noise to the connected components of the graph.

5 Minimization of 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 [25, 10, 5] apply a force-directed algorithm with an adaptive step length. The algorithm starts with an 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. This issue can be treated by applying a vertex specific adaptive step length. In each iteration, the 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. Our force algorithm is given in Algorithm 1.

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

Algorithm 1 ForceAlgo($\mathbf{Y}$,G,k,ϵ)

$\mathbf{X} \leftarrow \mathbf{Y}$ ▷ $\mathbf{Y}$ is the initial guess from the previous round of the multilevel algorithm.
$ratio \leftarrow 2.0$ ▷ $(> 1 + \epsilon)$
$\gamma \leftarrow 0.5$
$d_0 \leftarrow 0.00000001$ ▷ small float.
$s_0 \leftarrow \frac{|V|^2}{k(k|E|+|V|^2)}$ ▷ graph-dependent initial step length.
$\forall i \in V : s_i \leftarrow s_0$
$\vec{f_u} \leftarrow random$
while (ratio $> 1 + \epsilon$) **do**
  $BH(\mathbf{X})$ ▷ computing the Barnes and Hut tree on the current layout.
  d $\leftarrow 0.0$
  for $i \in V$ **do**
    $x_i^0 \leftarrow x_i$, $\vec{f_i^0} \leftarrow \vec{f_i}$
    $\forall j \in V, j \leftrightarrow i : \vec{f_i} = \vec{f_i} + \vec{f_a}(i,j)$ ▷ compute attractive forces of edges.
    $\forall j \in V, j \neq i, \vec{f_i} = \vec{f_i} + \vec{f_r}(i,j)$ ▷ compute pairwise forces. This computation is accelerated using the Barnes and Hut scheme.
    $s_i = s_i + s_i * (\frac{\vec{f_i}}{\|\vec{f_i}\|} \cdot \frac{\vec{f_i^0}}{\|\vec{f_i^0}\|}) * \gamma$ ▷ modify the step length proportional to the change of direction of the net force.
    $x_i = x_i + s_i * (\frac{\vec{f_i}}{\|\vec{f_i}\|})$ ▷ move the active vertex.
    $d = d + \left\|x_i - x_i^0\right\|$
  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

Theorem 5.1 *If p^0 is a drawing of a graph $G = (V,E)$ with minimum FlexGD energy then:* $k\sum_{\{u,v\}\in E}\|p_u - p_v\| + \sum_{\{u,v\}\in V^{(2)}}\|p_u - p_v\| = \left|V^{(2)}\right|$.

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:

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

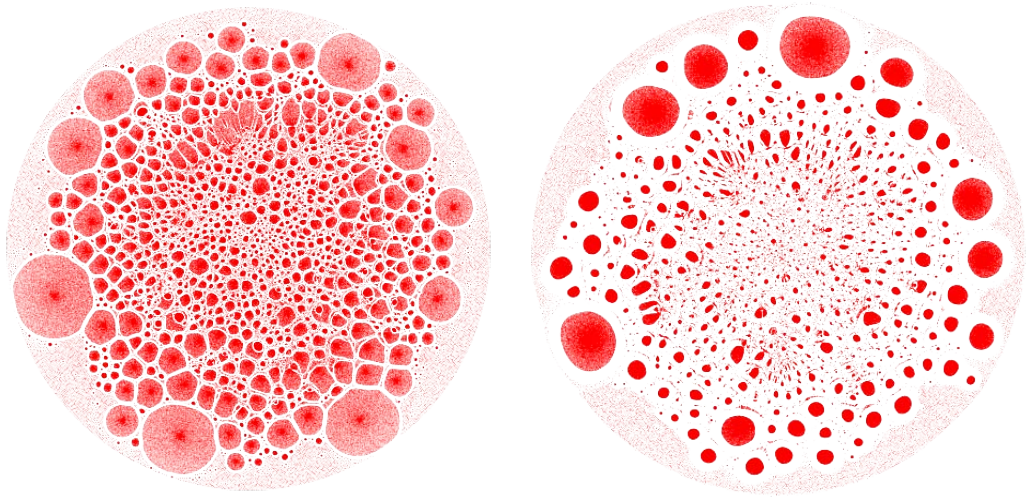
Since p^0 is a drawing with minimum energy, this equation has a minimum at $d = 1$, that is:

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

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

∎

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, $l = \frac{|V|^2}{(k|E|+|V|^2)}$



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

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

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

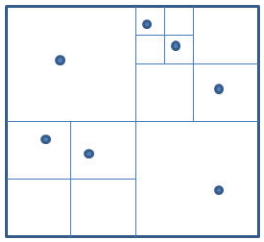


Figure 3: Formation of the quadTree in the Barnes and Hut algorithm.

is a value between $\bar{e}$ and $\bar{d}$. Dividing further by k gives a 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 basis through the adaptive step length scheme.

5.1 The Barnes and Hut Algorithm

A direct application of Algorithm 1 is not effective for large graphs, because its complexity is $O(|E| + |V|^2)$. A common practice in graph drawing (like [21, 16, 24]) is to decrease the complexity to $O(|E| + |V| \log |V|)$ using the Barnes and Hut scheme [1]. 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*, 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 the center of the parent node are updated as follows:

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

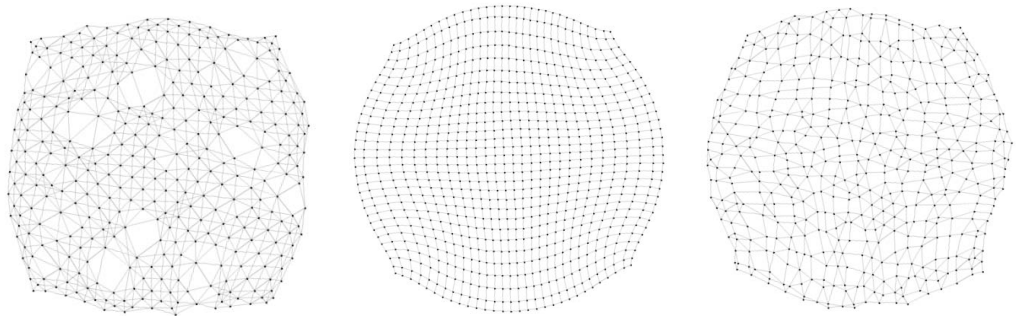
where m and $\vec{x}$ are the mass and the center of the node, and m_i and $\vec{x}_i$ the mass and coordinates of the inserted vertex, respectively. This procedure is iterated until all vertices are inserted, and there is only zero or one vertex in each external node. Figure 3 illustrates the formation of a quadTree. We form the quadTree once in the beginning of each execution cycle. When computing the pairwise forces, all the 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 $\theta = 0.5$.

5.2 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 create 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 promulgated to the finer graph. The finer graph usually needs less modifications as it is already in a rather good shape.

Edge Collapsing (EC) [6, 26, 10] is one 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.

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 [10] by adapting an alternative coarsening strategy based on Multiple Independent Vertex Set



(a) MIVS coarsening (b) grid 30×30 (c) EC coarsening

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

(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 [10], edges are computed using the Galrekin product of the prolongation matrix P and the adjacency matrix of the finer graph A_f (see [11] 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 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 Heavy Edge Collapsing (HEC) [26, 10]. Namely, each active vertex is merged with an unmatched neighbor corresponding to the heaviest incident edge, and the edge between them is collapsed. In few cases where the result of HEC was not good enough, we used an alternative coarsening strategy based on a Multiple Independent Vertex Set (MIVS) [10]. In this strategy, the coarser graph is built by choosing an MIVS from among the vertices of the finer graph. Then, an edge is added between each two vertices if their graph-theoretic distance apart is no more than 3. For implementation details about the coarsening and the prolongation phases of HEC and MIVS refer to the technical report provided in [*anonymized*]. We stop coarsening if one of the following happens. First, the level of coarsening is more than a predefined threshold. In our setting we do not coarse more than 12 levels. Second, 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.

6 Properties of FlexGD Minimum Energy Layouts

In this section, we derive some properties of FlexGD minimum energy layouts. The goal is to understand quantitatively how the model makes a tradeoff between the two drawing criteria, and how it separates the clusters by tweaking k.

Theorem 6.1 states that FlexGD finds the best compromise between maximizing the geometric mean and minimizing the weighted arithmetic mean distance between all vertices. This property is responsible for shortening the edges and lengthening the non-edges. If the graph contains no edges, the weighted 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 geometric 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 geometric 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 perfectly 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 weighted arithmetic mean. Therefore, their further shortening, up to some extent controlled by k, decreases the weighted arithmetic mean more than it increases the geometric mean.

Theorem 6.1 *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:

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

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

Model	Minimization equivalence	One-dimensional bipartition	abstractable
LinLog [19]	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.

The above expression may be reformulated in the form of $\text{minimize} - \ln(geo^{|V^{(2)}|}(V^{(2)}, p))$.
Since $\sqrt[|V^{(2)}|]{\exp(x)}$ 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:

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

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 $q^1 = \frac{c}{(|E|+|V^{(2)}|)arith^{k+}(V^{(2)},q^0)} q^0$ for which $arith^{k+}(V^{(2)}, q^1) = \frac{c}{|E|+|V^{(2)}|}$, 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 6.2 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*. Refer to Appendix A for more details of the approximation. For graphs containing a higher number of clusters, there is in general no 2D or 3D drawing where distance between *every* two clusters obeys the same equation, without violating the triangle inequality w.r.t. a third cluster. Despite this, Theorem 6.2 illustrates the logic behind the separation of clusters in FlexGD layouts.

Theorem 6.2 *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:

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

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 6.1 explains how convex subgraphs are clustered in the FlexGD layouts, Theorem 6.2 is responsible for the separation of clusters as a function of their coupling. This suggests 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 6.3 formalizes this finding for a set of abstraction constants $\{k_1, k_2, k_3\}$:

Theorem 6.3 *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 $\frac{k_3}{k_2}$.*

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

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

This can be rewritten as:

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

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 two properties of FlexGD expressed by Theorem 6.2 and Theorem 6.1 with analogous results about LinLog taken from [19].

7 Implementation and Results

We have implemented a multi-threaded simulator based on Java, and used the visualization capabilities of the JUNG library [12]. Most of the graphs in this section are taken from the University of Florida Sparse Matrix Collection [3]. FlexGD is very sensitive to the correct calibration of the initial step length according to Theorem 5.1 and the value of the abstraction constant k. It reveals

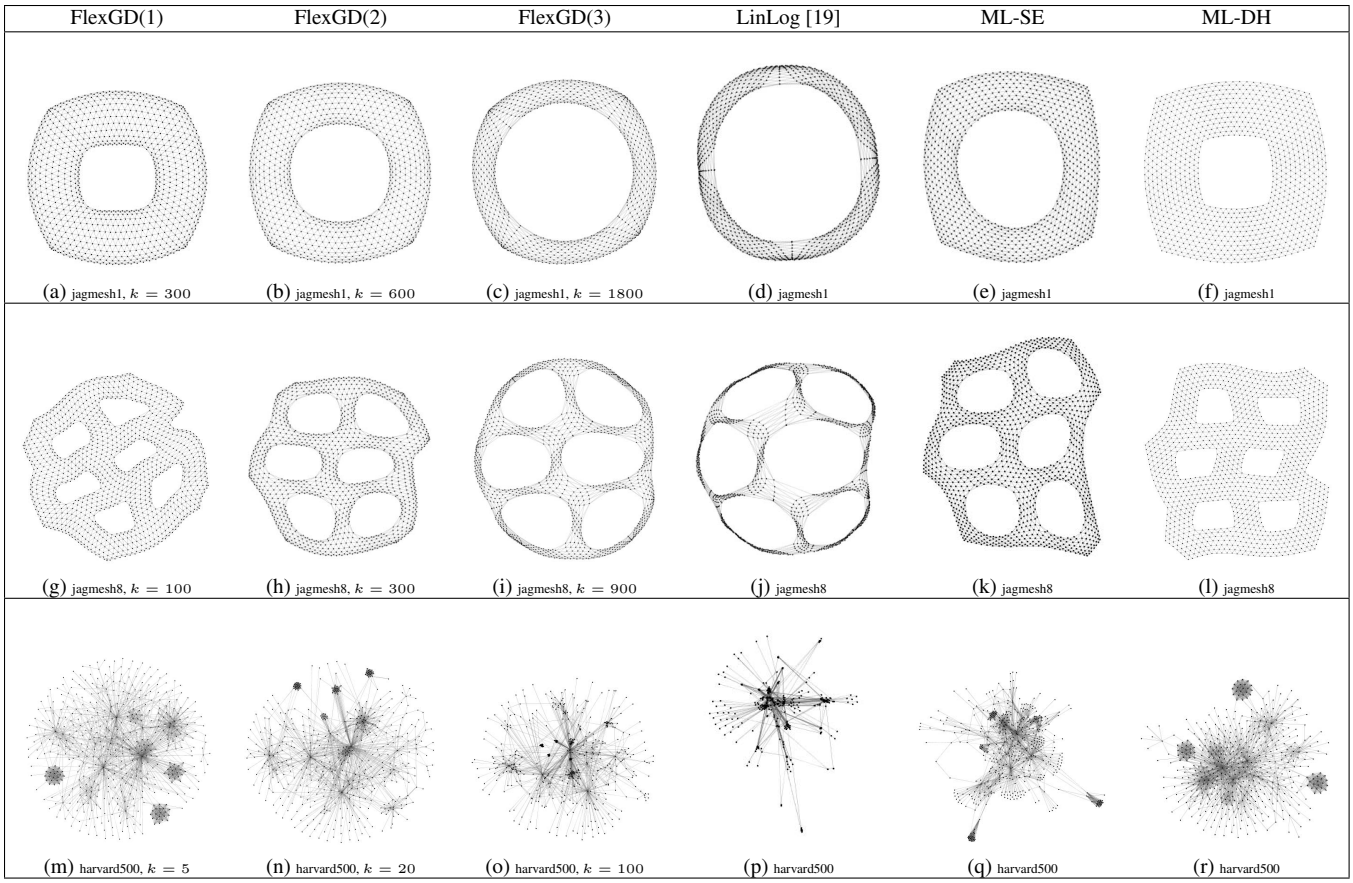


Figure 5: Comparison of FlexGD with force-directed models.

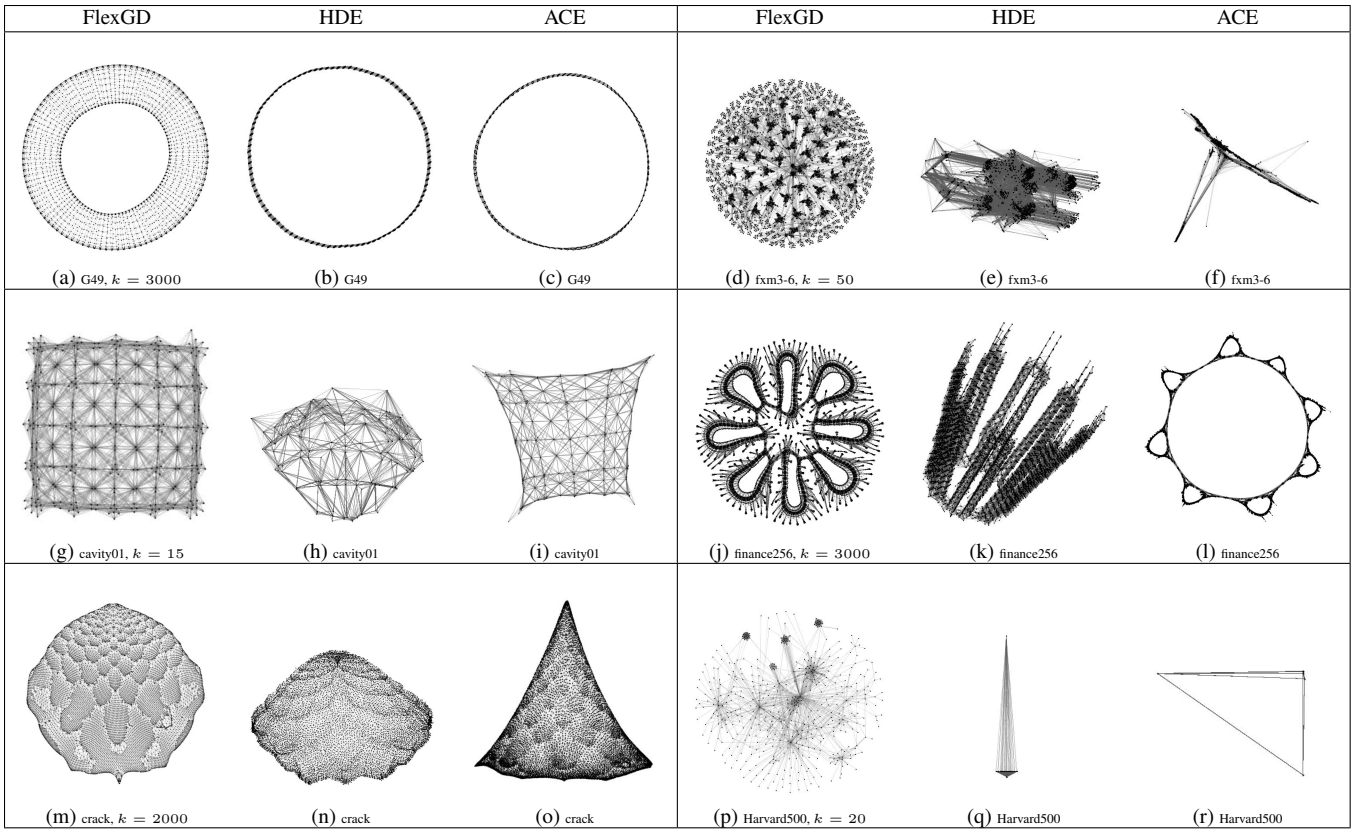
the structure of a graph provided k is large enough. We observed choosing k as $o(\frac{|V^2|}{|E|})$ is a proper value, depending on the level of abstraction the user prefers. If the graph is disconnected, the biggest component may be considered. For very sparse graphs, where vertex degree distribution is very uneven, smaller values of k can also be used. Such graphs generally result from applications like social networking or web connectivity networks.

Figure 5 compares the FlexGD layouts of a few graphs with the layouts of some other force-directed algorithms. Davidson and Harel [2] suggest an energy function resulting in more uniform vertex distribution. Spring-Electrical [5] is one of the most popular energy *functions* enforcing rather uniform edge length. Although these models are popular, the minimization algorithms suggested in the original papers are no more applied as more advanced algorithms have been proposed. We applied the multilevel algorithm suggested by Walshaw [25] to find the minimum energy layouts of these models. Hence, we call them ML-SE and ML-DH standing for Multilevel Spring-Electrical and Multilevel Davidson and Harel. LinLog layouts are drawn with a variant of Algorithm [1] where FlexGD attraction and repulsion forces are replaced with those of LinLog. Furthermore, an initial step length proper to LinLog is used. This step length is estimated through the analogue of Theorem 5.1 for LinLog. 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 the LinLog layouts. For smaller values of k, the FlexGD layouts are more similar to the layouts of the conventional models, while for larger values of k, they are closer to the LinLog layouts. This property is pretty interesting as one can draw a spectrum of layouts with different properties without changing the drawing model.

Figure 6 compares some other FlexGD sample layouts with the layouts of HDE and ACE. The running time of HDE and ACE is a few seconds. Though, their quality is generally much inferior to FlexGD layouts. As it is seen in Figures 5 and 6, FlexGD has a satisfactory performance on regular grid-like graphs. Though its main usefulness is for the representation of huge sparse graphs with non-uniform vertex degrees distribution. Many conventional models have difficulties with giving useful insight into the community structure of such graphs, i.e. they usually result a little informative clutter of interconnected vertices. One example of such graphs was shown in Figure 2. Other examples are provided in Figure 7. We believe for such graphs, drawing in the goal of visualizing the community structure is more indicative than using the conventional drawing criteria.

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.

The execution time of the FlexGD algorithm is given in Table 2 for some sample layouts. For graphs in the first part of Table 2, k has been chosen as $o(\frac{|V|^2}{|E|})$. The value $\frac{|V|^2}{|E|}$ is suggested by the algorithm to the users as one proper value for k. Graphs in the second part of Table 2 are sparse irregular graphs for which k is set to values smaller than $o(\frac{|V|^2}{|E|})$. For graphs containing up to



(a) G49, $k = 3000$ (b) G49 (c) G49 (d) fxm3-6, $k = 50$ (e) fxm3-6 (f) fxm3-6

(g) cavity01, $k = 15$ (h) cavity01 (i) cavity01 (j) finance256, $k = 3000$ (k) finance256 (l) finance256

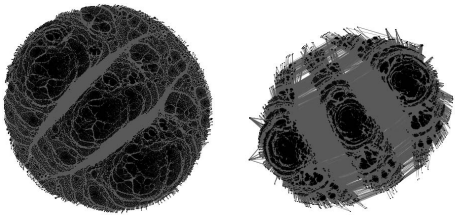
(m) crack, $k = 2000$ (n) crack (o) crack (p) Harvard500, $k = 20$ (q) Harvard500 (r) Harvard500

Figure 6: Comparison of FlexGD with HDE and ACE.

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. The Barnes and Hut scheme and the multilevel strategy improve significantly the quality and the running time of the algorithm. The adaptive step length scheme increases occasionally the running time as for some topologies like hollow ring-like graphs the layout converges late. Though, this scheme is essential to capture the non-uniform distribution of the edge length. An example is the G49 graph with 3000 vertices for which the running time is almost the same as that of the 100 by 100 grid which is 3 times larger in size but has a regular grid structure. In the same way, cegb2919 has about the same number of vertices as G49 and even contains more edges, but the algorithm draws it in almost a quarter of the time it takes to draw G49. In our experiments, our objective was to obtain the highest layout quality possible. Hence, we set $\theta = 0.5$ for the Barnes and Hut algorithm, and chose small values of tolerance. Of course, the running time of the algorithm depends on these settings. One can decrease the running time by choosing larger values of θ and tolerance if little distortion is tolerable in the underlying application. A collection of FlexGD layouts is provided in Figure 8. The properties of the graphs are given in Table 2. Interested readers may find more results and comparisons in Chapter 3 of [18].

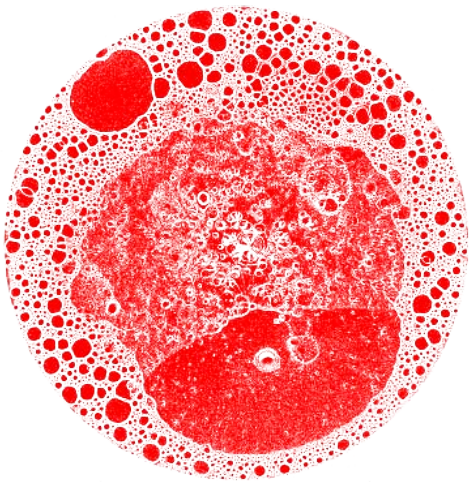
8 Conclusion

FlexGD allows the user to abstract the graph structure to a desired level, optimally filling a circular drawing. Consequently, tweaking the abstraction constant, a user has more chance to obtain her favorite drawing. It is suitable for cluster visualization and extends this property of the LinLog model. FlexGD is indeed an extension to LinLog which in behavior acts similar to the Binary Stress model. However, it enjoys the advantages of both models. On the one hand, it has the abstractability property of the Binary Stress. On the other hand, it extends the clustering property of LinLog. Hence, the clusters are separated better, and the behavior of the model is quantitatively describable. In general, FlexGD layouts are 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 for regular grid-like graphs FlexGD layouts are pleasing as much as, sometimes even more than, the layouts of previous models. FlexGD gives a helpful perspective into the community (cluster) structure of huge sparse graphs arising from domains like web applications, being different from the insight pro-

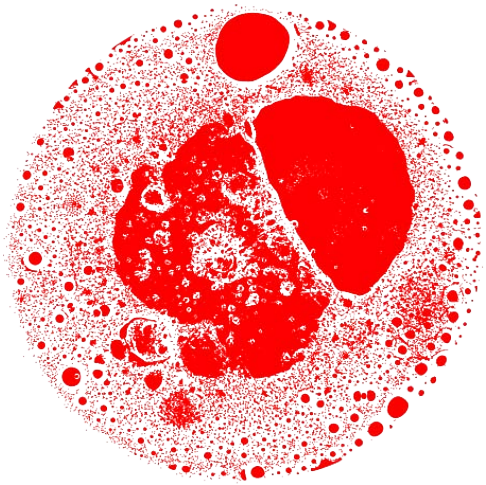


(a) FlexGD model, $k = 200$ (b) FlexGD model, $k = 1000$

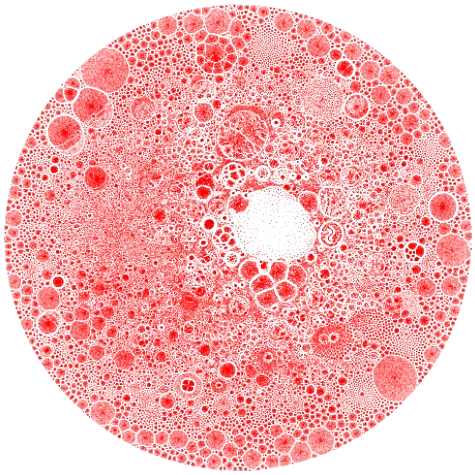
Figure 9: gupta1 graph



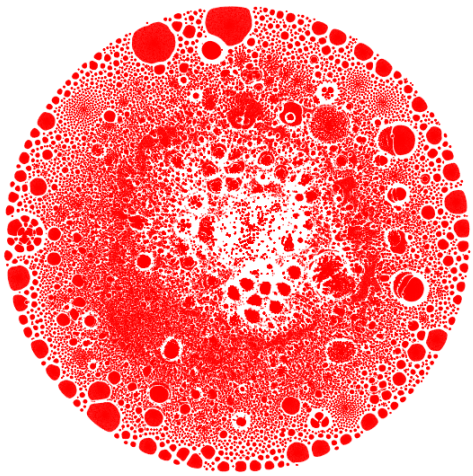
(a) Web connectivity matrix (webbase-1M), $|V| = 1000005$, $|E| = 3105536$, $k = 10000$



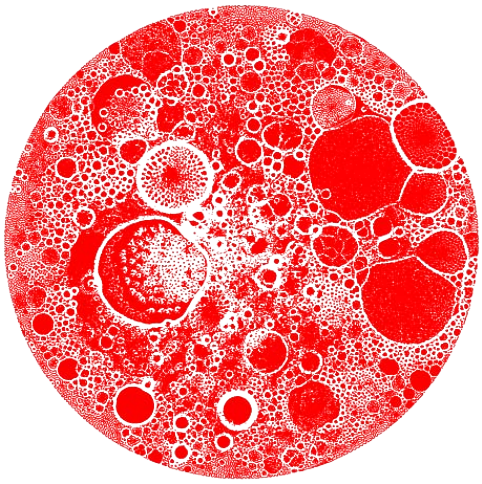
(b) Web connectivity matrix (webbase-1M), $|V| = 1000005$, $|E| = 3105536$, $k = 50000$



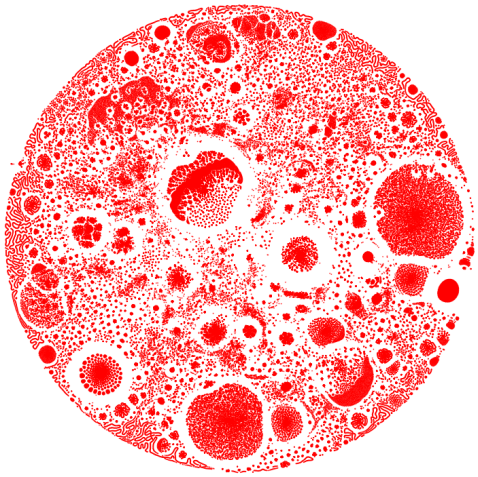
(c) Web of NotreDame University, $|V| = 325729$, $|E| = 1497134$, $k = 1000$



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

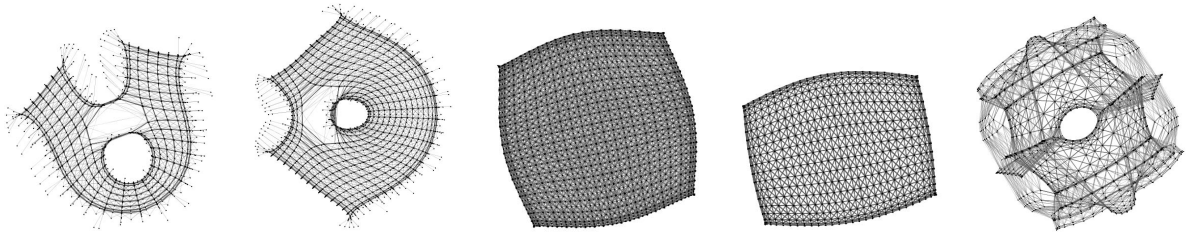


(e) Web of Stanford University, $|V| = 281903$, $|E| = 2312497$, $k = 1000$

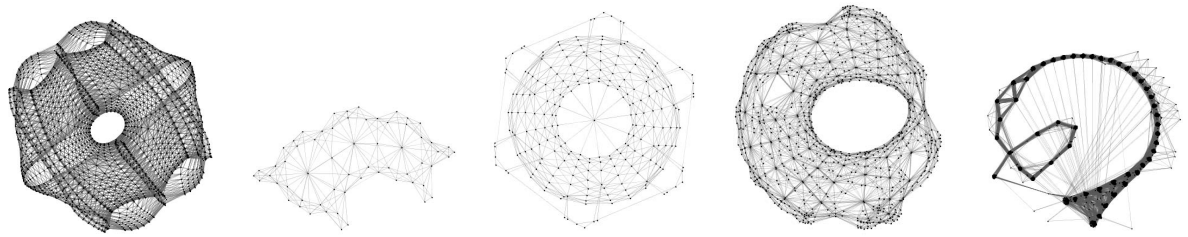


(f) Web of Stanford University, $|V| = 281903$, $|E| = 2312497$, $k = 5000$

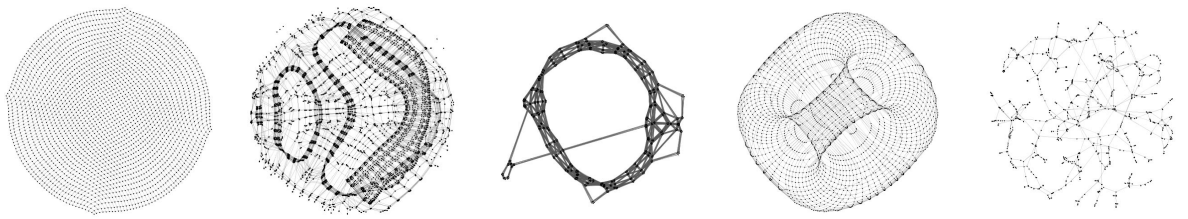
Figure 7: Sample Layouts of the FlexGD model.



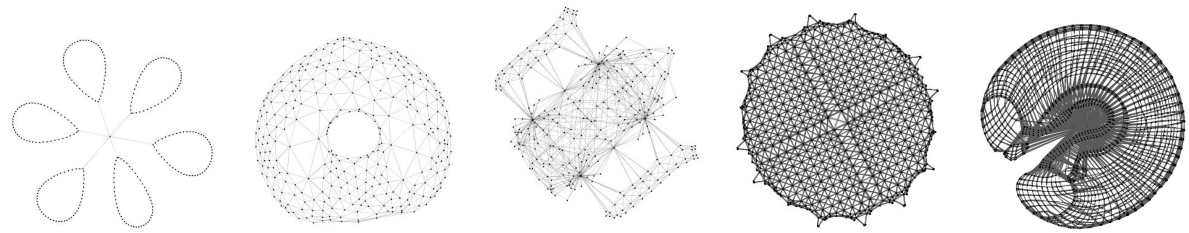
(a) utm1700b, $k = 200$, $46s$ (b) utm3060, $k = 300$, $87s$ (c) cavity24, $k = 200$, $62s$ (d) nasa2146, $k = 150$, $31s$ (e) nasa1824, $k = 200$, $22s$



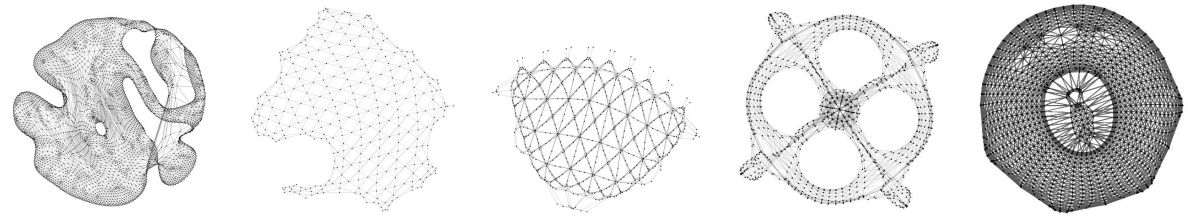
(f) nasa4704, $k = 400$, $135s$ (g) can61, $k = 10$, $3s$ (h) can229, $k = 60$, $12s$ (i) can838, $k = 150$, $9s$ (j) rdist3a, $k = 100$, $33s$



(k) grid50by50, $k = 50$, $54s$ (l) alemder, $k = 1500$, $139s$ (m) lock1074, $k = 50$, $25s$ (n) G34, $k = 1000$, $47s$ (o) 1138bus, $k = 500$, $60s$



(p) flower, $k = 200$, $6s$ (q) mesh2e1, $k = 100$, $12s$ (r) can292, $k = 60$, $3s$ (s) bcsstk24, $k = 200$, $54s$ (t) 3D28984Tetrak1500, $k = 1500$, $390s$



(u) 3elt, $k = 1000$, $556s$ (v) plskz362, $k = 100$, $3s$ (w) nnc666, $k = 150$, $6s$ (x) dwt1005, $k = 250$, $17s$ (y) raefsky5, $k = 250$, $209s$

Figure 8: FlexGD Layouts of sample graphs taken from the University of Florida Sparse Matrix Collection. The abstraction constant and the CPU time in seconds are given in the caption of each figure. Zooming on the layouts reveals more details.

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
harvard500	500	2636	20	7
1138bus	1138	2596	500	60
cavity01	317	7327	15	5
cavity24	4562	138187	200	62
fxm3-6	5026	49526	50	11
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
G34	2000	4000	1000	47
utm1700b	1700	21509	200	46
utm3060	3060	42211	300	87
3D28984Tetra	29984	599170	1500	390
mesh2e1	360	1162	100	12
can61	61	309	10	3
can229	229	1003	60	12
can292	292	1416	60	3
can838	838	5424	150	9
cegb2919	2919	162201	50	44
nasa1824	1824	20516	200	22
nasa2146	2146	37198	150	31
nasa4704	4704	54730	400	135
Alemder	6245	24413	1500	139
raefsky5	6316	168658	250	209
nnc666	666	4044	150	6
flower	300	306	200	6
lock1074	1074	26313	50	25
dwt1005	1005	4813	250	17
rdist3a	2398	61896	100	33
3elt	4720	13722	1000	556[b]
crack	10240	30380	2000	1449
web-NotreDame	325729	1497134	1000	3142
web-Stanford	281903	2312497	5000	9669
email-EuAll	265214	420045	4000	20455
webbase-1M	1000005	3105536	10000	22352

[a] Times are measured on a 2.4GHz Core2 Duo with 1G of RAM.
[b] Times are measured on a 2.5GHz Xeon E5420 with 4G of RAM.

vided by the conventional models.

FlexGD has the tendency to give a circular shape to the layouts. The circular layout is the natural legacy of using both attractive and repulsive forces to distribute the vertices over the layout. This design choice was originally made to give the abstraction capability to the model. If the width and the length of the graph structure are proportional, this is not a severe constraint, specifically because the circular shape of the layout decreases with k. Fortunately, the majority of real-world applications give rise to such graphs. However, in fewer cases where the length and the width of the graph are very disproportional (like the line graph in Figure 1b), the circular shape of the layout may disturb the graph structure by enforcing it to be packed into a disc. Though, one should note that very long-shaped structures (like a 20×400 grid) are known to be one of the most challenging types of topology for all models of graph drawing.

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 [19]. Its other asset is the multilevel algorithm coping with the non-uniform distribution of the edge length and its model-dependent parameterization. Variant of this algorithm can be considered as a complement to [19], as to date we are unaware of works reporting LinLog layouts of large graphs.

We only treated the case of unweighted graphs. Though, the model can be easily generalized for weighted graphs by integrating the edge weights into the attraction term of the energy function. This gives the following equation for FlexGD energy function of weighted graphs:

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

where ω_{uv} is the edge weight between u and v. All the previous theorems remain correct, but the edge weights are added to the terms corresponding to edges. 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 [10]. 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 [3] to reveal the three groups of vertices in the graphs. Interestingly, FlexGD reveals these clusters easily without any coarsening phase. Figure 9 shows the FlexGD layout of this graph in two levels of abstraction.

References

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

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

[3] T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Softw.*, pages 1:1–1:25, 2011.

[4] P. Eades. A heuristic for graph drawing. In *Congressus Numerantium*, pages 149–160, 1984.

[5] T. M. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. In *Software-Practice and Experience*, pages 1129–1164, 1991.

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

[7] S. Hachul and M. Jünger. An experimental comparison of fast algorithms for drawing general large graphs. In *Graph Drawing*, pages 235–250, 2005.

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

[9] D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. In *Revised Papers from the 10th International Symp. on Graph Drawing*, pages 207–219, 2002.

[10] Y. Hu. Efficient, high-quality force-directed graph drawing. *ACM Trans. Math. Softw.*, pages 1:1–1:25, 2011.

[11] Y. F. Hu and J. A. Scott. A multilevel algorithm for wavefront reduction. *SIAM J. Sci. Comput.*, pages 1352–1375, 2001.

[12] *JUNG*, 2012. `http://jung.sourceforge.net`.

[13] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, pages 7–15, 1989.

[14] Y. Koren. Drawing graphs by eigenvectors: theory and practice. *Comput. Math. Appl.*, pages 1867–1888, 2005.

[15] Y. Koren, L. Carmel, and D. Harel. Ace: a fast multiscale eigenvectors computation for drawing huge graphs. In *INFOVIS*, pages 137–144, 2002.

[16] Y. Koren and A. Çivril. The binary stress model for graph drawing. In *Graph Drawing*, pages 193–205, 2009.

[17] U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, pages 395–416, 2007.

[18] A. Moin. *Recommendation And Visualization Techniques For large Scale Data*. PhD thesis, Université Rennes 1, 2012.

[19] A. Noack. Energy models for graph clustering. *Journal of Graph Algorithms and Applications*, pages 453–480, 2007.

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

[21] A. Quigley and P. Eades. Fade: Graph drawing, clustering, and visual abstraction. In *Graph Drawing*, pages 197–210, 2001.

[22] S. E. Schaeffer. Graph clustering. *Computer Science Review*, pages 27 – 64, 2007.

[23] I. G. Tollis, G. Di Battista, P. Eades, and R. Tamassia. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

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

[25] C. Walshaw. A multilevel algorithm for force-directed graph drawing. In *Graph Drawing*, pages 31–55, 2001.

[26] C. Walshaw, M. G. Everett, and M. Cross. Parallel dynamic graph partitioning for adaptive unstructured meshes. *J. Parallel Distrib. Comput.*, pages 102–108, 1997.

A Distance Interpretability in 1+ Dimensional FlexGD Bipartition Layouts

In this appendix we explain the approximate generalizability of Theorem 6.2 to 1+ dimensions. The following theorem holds exactly for layouts with any number of dimensions:

Theorem A.1 *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 A.1 leads to the following useful corollary:

Corollary A.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]

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

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.*

We know from Theorem 6.1 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 A.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 A.1, we obtain:

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

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

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

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