Drawing Graphs with Non-Uniform Vertices

David Harel
harel@wisdom.weizmann.ac.il

Yehuda Koren
yehuda@wisdom.weizmann.ac.il

Dept. of Computer Science and Applied Mathematics
The Weizmann Institute of Science, Rehovot, Israel

ABSTRACT

The vertices of most graphs that appear in real applications are non-uniform. They can be circles, ellipses, rectangles, or other geometric elements of varying shapes and sizes. Unfortunately, current force directed methods for laying out graphs are suitable mostly for graphs whose vertices are zero-sized and dimensionless points. It turns out that naively extending these methods to handle non-uniform vertices results in serious de£ciencies in terms of output quality and performance. In this paper we try to remedy this situation by identifying the special characteristics and problematics of such graphs and offering several algorithms for tackling them. The algorithms can be viewed as carefully constructed extensions of force-directed methods, and their output quality and performance are similar.

Categories and Subject Descriptors

H.5 [**Information Systems**]: Information Interfaces and Presentation; I.3 [**Computing Methodologies**]: Computer Graphics

General Terms

Algorithms, Design, Experimentation

Keywords

Graph Drawing, Visualization, Force directed optimization, Vertex overlaps

1. INTRODUCTION

A graph $G(V, E)$ is an abstract structure that is used in the real world to model an entity-relationship structure; the entities are represented by set of the vertices V and the relationship by the edges $E \subseteq V \times V$. Since real graphs are usually intended to be comprehended by humans, the usefulness of a graph depends on the clarity of its layout. Achieving a clear, aesthetic picture of a graph is not an easy task, and for many years now researchers have been motivated to seek automated means for drawing graphs "nicely". The state of the art is surveyed comprehensively in [3].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for pro£t or commercial advantage and that copies bear this notice and the full citation on the £rst page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior speci£c permission and/or a fee.
AVI '2002 Trento, Italy
Copyright 2002 ACM X-XXXXX-XX-X/XX/XX ...$5.00.

A central graph drawing problem is that of laying out an arbitrary undirected graph. The most common approach for drawing such graphs is apparently the force-directed approach [4, 9, 1, 5]. Algorithms based on this approach consist of two components. The £rst is the heuristic force (or energy) model that quanti£es the quality of a drawing, and the second is an optimization algorithm for computing a drawing that is locally optimal with respect to this model.

In practice, the vertices of graphs are non-uniform. There are applications with different types of vertices that are depicted using different geometric shapes. Also, a vertex in a graph often represents an entity that is associated with some information — a name or a parameter – and we would like to provide this information inside the drawing of the vertex. We thus often have varying sized circles, ellipses or rectangles as the vertices. Unfortunately, most of the force directed methods represent vertices by zero-sized dimensionless points in the plane, and naive extensions to handle non-uniform vertices are seriously de£cient in terms of output quality and performance.

In this paper we present several algorithms for drawing undirected graphs with vertices of various shapes and sizes. The algorithms can be viewed as enhancements and generalizations of existing force-directed methods, and unlike the naive extensions their output quality and performance are comparable to the originals.

2. FORCE-DIRECTED GRAPH DRAWING

The force-directed approach is apparently the prevalent attitude for drawing general graphs. Algorithms based on this approach consist of two components. The £rst is the force (or energy) model that quanti£es the quality of a drawing. The second is an optimization algorithm for computing a drawing that is locally optimal with respect to this model. The resulting £nal layout brings the system to equilibrium in which the total force on each vertex is zero, or equivalently, the potential energy is locally minimal with respect to the vertex positions. Regarding the drawing standard, force-directed methods draw the edges as straight-line segments, so the issue reduces to the problem of positioning the vertices.

Here we outline some notable work on force-directed graph drawing. In the following sections we introduce our enhancements to these methods, which enable us to deal with variably sized and shaped vertices.

2.1 The Spring Embedder Method

The spring embedder method is the earliest viable algorithm for drawing general graphs. It was proposed by Eades [4], and was later re£ned by Fruchterman and Reingold [5]. This method likens a graph to a mechanical collection of electrical charged rings (the vertices) and connecting springs (the edges). Every two vertices reject each other by a repulsive force and adjacent vertices (connected by an edge) are pulled together by an attractive force. The method seeks equilibrium of these con¤icting constraints. Spring based methods are very successful with small-sized graphs of up to around 50 vertices.

2.2 Kamada and Kawai's Method

Kamada and Kawai [9] modelled a graph as a system of springs that act in accordance with Hooke's Law: Every two vertices are connected by a spring, whose rest length is proportional to the graph-theoretic distance between its two endpoints, and its stiffness is inversely proportional to the square of its rest length. The optimization procedure tries to minimize the total energy of the system, that is:

$$E = \sum_{v,u \in V} \frac{1}{d_{uv}^2} (l(u,v) - L d_{uv})^2$$

where $l(u, v)$ is the length of the spring between u and v, L is the length of a single edge, and d_{uv} is the graph-theoretic distance between u and v.

Kamada and Kawai's method treats all the aesthetic criteria that the spring-embedder method addresses, and produces drawings with a similar quality. An advantage of this method is that it can be applied straightforwardly to drawing weighted graphs, assuming that edge lengths have to re¤ect their weights.

2.3 Multi-scale Graph Drawing

The *multi-scale* approach is a vast improvement of the force-directed technique, which facilitates the drawing of much larger graphs (containing over 10,000 vertices), see e.g., [7, 8]. These methods consider a series of abstractions of the graph called *coarse graphs*, in which the combinatorial structure is signi£cantly simpli£ed, but important topological features are well preserved. The energy minimization is divided between these coarse graphs, in such a way that globally related properties are optimized on coarser graphs, while locally related properties are optimized on £ner graphs. As a result, the energy minimization process considers only small neighborhoods at once, yielding a quick running time.

3. DEFINING THE PROBLEM

Seeking a nice layout of a graph, we £rst have to de£ne what we mean by "nice". There are several known aesthetic criteria for graphs, whose relevance has been demonstrated in the literature, such as uniform and small edge lengths. When dealing with vertices that are no longer zero-sized points, all the regular criteria still hold, but two additional *constraints* are needed, to prevent overlaps that involve vertices:

C1 Vertices are not to overlap.

C2 Edges are not to cross vertices.

What kind of vertices do we allow? Well, most of the methods we describe can be adapted easily to take care of any common shape, but we have implemented them for ellipses or rectangles. We also assume that the user has oriented the vertices, so they are not to be rotated or re¤ected. This means that laying out a vertex entails only deciding on the location of its center.

As for the edges, we have chosen to use the straight-line standard, whereby edges are drawn as straight-line segments connecting the boundaries of the vertices. For the actual choice of this segment our implementation offers two possibilities: (i) the boundary-connecting segment that lies on the line connecting the centers of the vertices, or (ii) the shortest straight-line segment connecting the two boundaries. Having adopted one of these possibilities, the problem of drawing a graph reduces to the problem of deciding on locations for its vertices. Hence, a layout is de£ned as follows:

DEFINITION 3.1. *A* layout *of a graph $G(V, E)$ is a mapping of the graph's vertices to the two dimensional Euclidean space: $X : V \longrightarrow \mathbb{R}^2$. When the vertices are not points, $X(v)$ is taken to be the location of the center of v.*

Trivial solutions

A couple of naive methods for drawing such graphs come to mind. The easiest one is to consider the graph as having conventional zero-sized vertices, to construct a nice layout of it using some known method, and then to scale-up the entire resulting drawing until there is enough space for all the vertices, without violating constraints C1 and C2 above. This method will always work. However, some important aesthetic criteria are not achieved. Because the scaling-up is carried out globally, the edges will not be of uniform length, and many might be very long. Except for very simple cases, the drawing will not be compact and will look extremely distorted.

We should stress that the ability to achieve a compact picture is of great importance, since area-ef£cient drawings are essential in practical visualization applications where screen space is one of the most valuable commodities.

A better approach is to £rst bound the vertices from the outside by circles. Drawing a graph with circular shaped vertices can be carried out easily using a simple variant of virtually any known force-directed method, by increasing the length of each edge by the lengths of the radii of its incident vertices. For more details see Section 6, where we call such graphs "wt-graphs". This method will quickly produce drawings without vertex overlaps. However, the edge lengths would still be non-uniform, and the resulting drawing would not be satisfactorily compact because of the wasted space between enclosing circles and the original vertices. For example, many applications have rectangular vertices that are long and narrow.

Our intention when describing these two trivial solutions is to convince the reader that we can handle large sized vertices quite easily when allowing long and non-uniform edges. In the rest of this paper we will show methods that deal with large vertices, while producing pleasingly compact drawings.

4. SPRING BASED METHODS

The simplicity of the underlying principle of the spring method makes it ¤exible enough to be adapted for non-uniform vertices by rede£ning the forces. Our modi£cations will be seen to prevent vertex overlaps, while achieving the aesthetic criteria that the original method addressed. When edge lengths are short and uniform edge-vertex overlaps will be eliminated too (see Section 8). We strongly believe in methods that tightly integrate the achievement of aesthetic issues with trying to impose the constraints, because this way we do not have to separately address the aesthetics and the constraints and to trade-off between the two. As a result, our methods will not move two vertices away from each other without considering the aesthetic implications of this action.

The Elliptic Spring Method

Ellipses are better than circles in bounding the kinds of shapes that appear in real applications, like rectangles and parallelograms, so it makes sense to deal with elliptical vertices. Hence, we propose a generalization of standard spring based methods for elliptic-shaped vertices. We restrict ourselves to ellipses whose radii are parallel to the axes.

The method uses forces with elliptic and inverse-elliptic force £elds, which are related to the shapes of the vertices. The forces are de£ned as follows:

For any two elliptical vertices v_i and v_j with centers (x_i, y_i) and (x_j, y_j) and radii (r_i^x, r_i^y) and (r_j^x, r_j^y), respectively, we de£ne the amount of repulsive force between their centers to be:

$$f_r = w \cdot \left(\frac{(x_i - x_j)^2}{(r_i^x + r_j^x + Len)^2} + \frac{(y_i - y_j)^2}{(r_i^y + r_j^y + Len)^2} \right)^{-1}$$

If the vertices are adjacent, they attract each other by an attractive force of strength:

$$f_a = \frac{(x_i - x_j)^2}{(r_i^x + r_j^x + Len)^2} + \frac{(y_i - y_j)^2}{(r_i^y + r_j^y + Len)^2}$$

The intuition is that we do not want the center of v_i to lie inside the ellipse whose center is (x_j, y_j) and its radii are $(r_i^x + r_j^x + Len, r_i^y + r_j^y + Len)$.

When two ellipses overlap, the repulsive force is very large and the attractive force is small. When two adjacent ellipses are far away we have the opposite behavior. Here, w is a weighting constant typically set to 1, in which case at adjacent vertices the two forces cancel each other out when the two ellipses are approximately at distance Len. If the resulting layout still contains vertex overlaps, one should increase the value of w (or of Len) so that the vertices are placed further apart. This may be done automatically, and focused locally on the overlapping vertices.

To £nd the layout itself, we use the optimization method of Fruchterman and Reingold [5]: In the initial con£guration, all the vertices are placed randomly in the frame, and the £nal con£guration is achieved by a prede£ned number of *sweeps*. In each sweep every vertex moves in the direction that is determined by the total effect of all the forces exerted on it. The amount of the displacement is decreased in each iteration, which is like decreasing a *temperature*.

The method can be applied to graphs with zero-sized vertices by setting all the radii to 0, and it achieves results comparable to other known spring based methods. We thus call the algorithm the *Elliptic Spring Method*, since it generalizes the traditional spring based approach to elliptically shaped vertices.

Figure 1 exhibits results of the Elliptic Spring Method, and demonstrates the ¤exibility of the algorithm in allocating the necessary area for the vertices while achieving an aesthetic layout. Graphs A and A1 have the same structure, but different vertex sizes. Various statistics regarding these results are given in Table 1.

The Modi£ed Spring Method

An alternative approach to adopting the spring-embedder for our needs, is to take the distance between vertices to be the shortest distance between their boundaries (and not between their centers). As a consequence, we rede£ne the strengths of the forces in the Fruchterman-Reingold method [5] to be:

$$f_a(v, u) = \frac{l_b(v, u)^2}{Len} \qquad f_r(v, u) = w \cdot \frac{Len^2}{\max(l_b(v, u), \epsilon)}$$

Here $l_b(u, v)$ is the shortest distance between the boundaries of v and u, and $\epsilon > 0$ is a small constant. When v and u overlap, we let $l_b(u, v) = 0$. Also, w is a constant, typically set to 1. The direction of the forces is between the centers of gravity of the two vertices. The two forces cancel each other out when $l_b(u, v) = Len$. The repulsive force is dominant and subsumes the attractive force when the two vertices are tangential or when they overlap. This prevent overlaps, since when two vertices are too close a strong repulsive force detaches them.

We call this algorithm the *Modi£ed Spring Method* and it can be easily applied to rectangular vertices. A practical problem that may arise when implementing this method for various shapes (like ellipses) is that the computation of the shortest distance between their boundaries is too costly, especially considering that it is inside the innermost loop of the algorithm. We can overcome this problem by devising a variant of the algorithms in which we measure the distance between the boundaries of two vertices on the straight-line segment that connects their centers of gravity. The distance is de£ned to be 0 when vertices overlap on this straight-line.

Figure 2 shows layouts of rectangular versions of the graphs of Figure 1, as produced by the Modi£ed Spring Method.

Comparison

In Table 1, we provide various statistics regarding the results of the Modi£ed Spring Method (MSM) and the Elliptic Spring Method (ESM), both applied to elliptic vertices. For comparison, we also give the statistics of the Fruchterman-Reingold Method (FRM) [5] applied to the same graphs, but with dimensionless vertices. For each graph we give the number of sweeps needed for achieving an aesthetic picture. The actual running time on a Pentium III 700MHz PC is also provided. One can see that the running time of the ESM is slightly better than that of the MSM. This should be attributed to the fact that the forces in the ESM are milder, and hence easier to optimize. Moreover, the forces of the ESM can be computed more ef£ciently. However, it is very clear that the running time of the FRM is much faster than those of MSM and ESM. We will discuss this point in the next section.

Regarding output quality, for each graph we supply the average edge length ($AvgLen$), the standard deviation of the edge length ($StdDev$), and the ratio $StdDev/AvgLen$. This ratio is a normalized measure of the uniformity of the edge lengths, independent of the edge lengths. From these results it is obvious that the MSM is superior to the ESM regarding uniformity of edge-length. Moreover the uniformity of edge lengths of the MSM is comparable to this of the FRM, so in this sense the MSM is an optimal generalization of the FRM to handle non-uniform vertices. Another advantage of MSM over ESM is its ability to handle directly a variety of shapes, while the ESM can only approximate them by ellipses.

Our conclusion is that there is no clear winner between MSM and ESM. In the following sections we shall be describing better methods, and the central role of the two methods described here will be as a £nal beauti£cation of an "almost nice" picture. For this kind of use, the MSM may have some advantage.

The Paradox or Why Convergence is Slow?

The relatively slow convergence rate of the ESM and the MSM, comparing with the FRM, should not be attributed to a ¤aw in their design, but to a fundamental problem. Apparently, any force-directed algorithm that deals with sized vertices is doomed to slow convergence if its cost function is to prevent overlap between vertices. The reason is that when the vertices take a signi£cant fraction of the drawing area, there is less space for maneuver when seeking a nice layout, and this slows convergence considerably. For an illustration of this see Figure 3, in which some of the vertices are blocked from achieving their proper place, because of the prevention of vertex overlap.

It seems that there is an unavoidable tradeoff between the ability to prevent vertex overlaps and the convergence rate. Paradoxically, we cannot get a robust drawing algorithm, because the aspiration for nice layout contradict the need for fair convergence rate, and one that will not be trapped easily in local minima. We want to remark that the same kind of problem may arise in many common

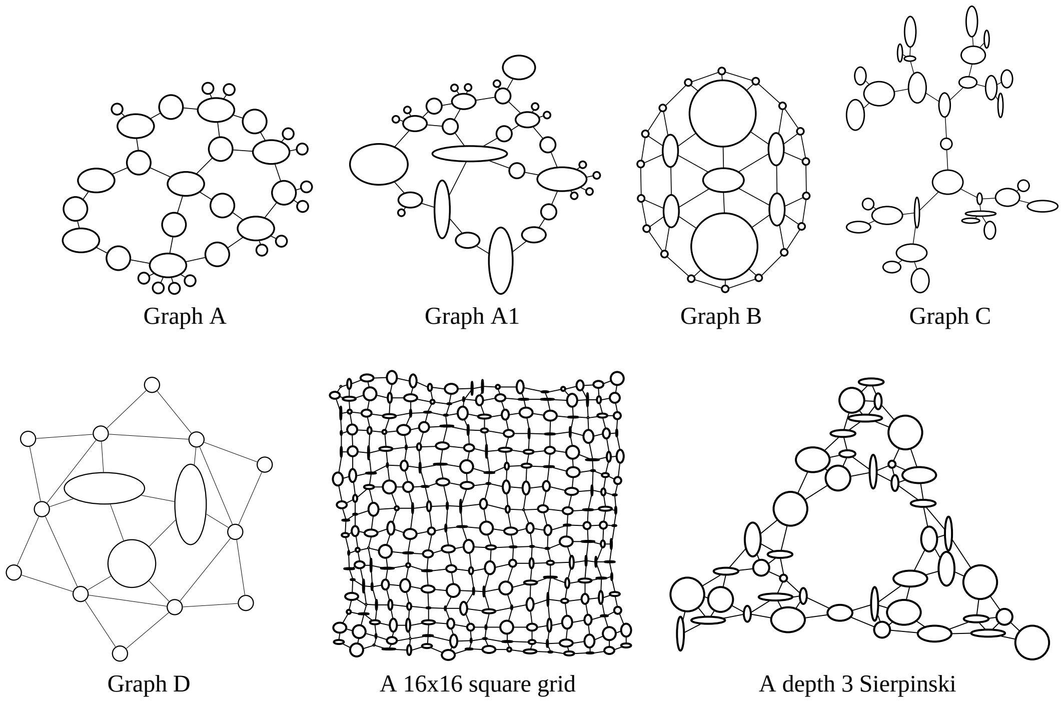


Figure 1: Results of the Elliptic Spring Method

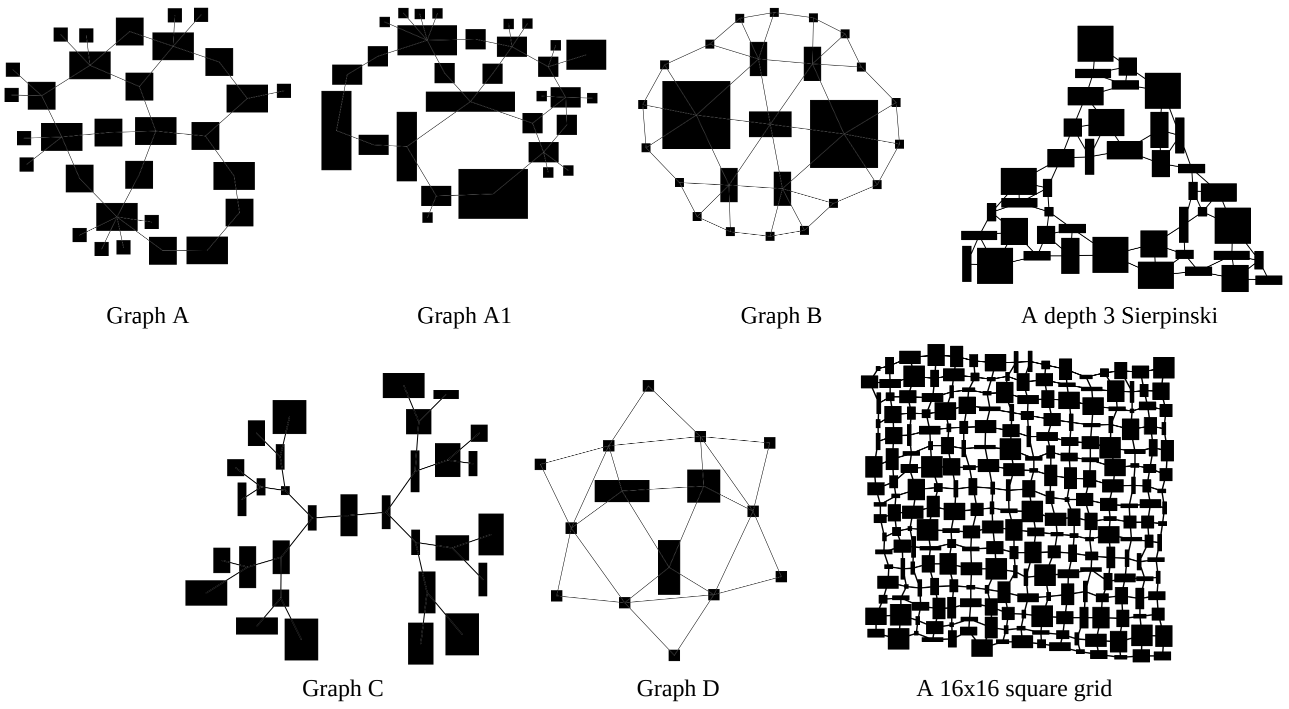


Figure 2: Results of the Modi£ed Spring Method

Graph name	**Method**	**Sweeps**	**Time (sec.)**	**Average edge length (AvgLen)**	**Standard deviation of lengths (StdDev)**	$\frac{\mathrm{StdDev}}{\mathrm{AvgLen}}$
A	MSM	1302	1	1.57	0.27	0.17
	ESM	650	1	1.75	0.94	0.54
	FRM	128	0	1.91	0.36	0.19
A1	MSM	1957	2	2.47	0.5	0.2
	ESM	780	1	2.64	1.75	0.66
	FRM	128	0	1.91	0.36	0.19
B	MSM	1954	1	4.5	1.32	0.29
	ESM	1302	0	5.19	1.68	0.32
	FRM	95	0	2.31	0.47	0.2
C	MSM	976	1	2.85	0.71	0.25
	ESM	976	0	3.13	1.87	0.60
	FRM	194	0	1.20	0.37	0.31
D	MSM	976	0	5.49	1.76	0.39
	ESM	976	0	6.54	2.36	0.36
	FRM	76	0	2.28	0.33	0.14
Sierpinski (depth 3)	MSM	4345	5	2.55	0.85	0.33
	ESM	4888	4	3.19	1.55	0.49
	FRM	389	0	1.37	0.41	0.3
Grid 16x16	MSM	1954	87	4.58	1.19	0.26
	ESM	1954	73	6.71	1.88	0.28
	FRM	432	15	2.20	0.41	0.19

Table 1: Statistics regarding the results of the modi£ed spring method (MSM), the elliptic spring method (ESM) and the Fruchterman-Reingold method (FRM). Running times are on a Pentium III 700MHz PC, and are rounded. For MSM and ESM, shorter edges are preferable, as they yield a more compact picture. For FRM, the average edge length is given only as a scaling measure.

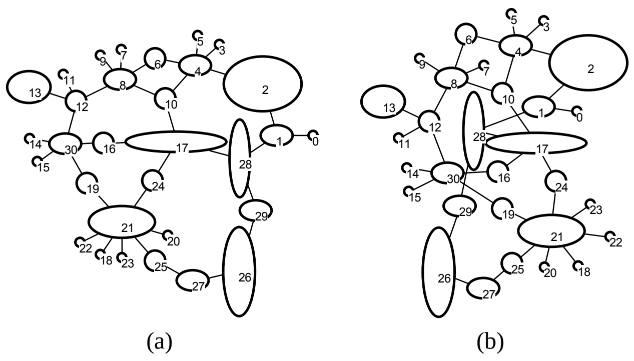


Figure 3: (a) Desired layout (b) Bad local minimum

cases, when the cost function tries to deal with a rich set of aesthetic criteria. A remarkable example is a cost function that tries to minimize edge crossings. When experimenting with the Simulated Annealing method of [1], we have found that when extracting the edge-crossing component from the energy function, the convergence rate becomes much faster.

Fortunately, in the case of vertex overlaps, we have found a way to get around this: *forcing the constraints gradually*. At the £rst stage we set things up so that the cost function is weak and does not completely prevent overlaps. As the drawing process proceeds and the coarse structure of the picture is found, the cost function fully prevents overlaps. This principle is used in the method described in Section 7, but before going into it we want to describe how the Kamada-Kawai method [9] can be generalized to incorporate arbitrarily sized vertices.

5. THE ITERATIVE KAMADA-KAWAI METHOD

In this section we introduce a method that £nds a nice layout of a graph with arbitrarily sized-vertices, by iteratively £nding a nice layout of a related weighted graph with conventional dimensionless vertices.

DEFINITION 5.1. *Let X be a layout of a graph $G(V, E)$ and let $u, v \in V$. We denote by $in^X(v, u)$ the length of those segments of the straight-line connecting $X(v)$ and $X(u)$ that are inside the drawing area of vertices v and u.*

We now de£ne a suitable metric on the vertices of a layout.

DEFINITION 5.2. *Let $G(V, E)$ be a graph and let X be a layout of G. We de£ne the weighted graph $G^X(V, E, w)$, with weighting function w, such that $w(u, v) = Len + in^X(u, v)$.*
The metric d^X on V is de£ned as the graph-theoretic distance between vertices in the graph G^X

Len is a constant chosen in an attempt to approximate the desired uniform edge length, which is independent of the shape and placement of the vertices.

Notice that d^X is a generalization of the graph-theoretic distance, and it expresses the fact that when vertices have non-zero sizes, the length of a path should be measured not only by the edges, but also by taking into account the appropriate segments that pass through the vertices.

The heuristic

Our heuristic for nice drawing relates to "good" isometry between the Euclidean metric and the metric d^X. More precisely, our position is that X is a nice layout if it minimizes the energy function:

$$E = \sum_{v,u \in V} k^X_{v,u} \cdot \left(|X(v) - X(u)| - d^X(v, u)\right)^2$$

Here $|X(v) - X(u)|$ is the Euclidean distance between $X(v)$ and $X(u)$ and $k^X_{v,u} \approxeq d^X(v, u)^{-2}$ is a normalizing factor. We will call this heuristic *GKK* (Generalized Kamada-Kawai).

The optimization method

The optimization method we propose consist of an incrementally improving sequence of layouts, as follows:

Graph	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Overall time (sec., rounded)
A	326	54	27	25	0
A1	393	62	42	10	0
B	207	34	5	1	0
C	307	51	22	11	0
D	193	12	1	1	0
Grid 16x16	2793	555	217	54	3
Grid 20x20	6634	987	407	1	10
Sierpinski (depth 4)	2902	710	329	133	1

Table 2: Convergence rate of the Iterative Kamada-Kawai method

- X_0 is the initial layout (which can be random; a better choice is described at the end of Section 6).
- X_{i+1} is a layout that minimizes the energy function:

$$\sum_{v,u \in V} k^{X_i}_{v,u} \cdot (|X_{i+1}(v) - X_{i+1}(u)| - d^{X_i}(v,u))^2$$

The minimization is carried out by optimizing each vertex separately, using the Newton-Raphson method (exactly as would be done had we executed the classical method of Kamada-Kawai on the weighted graph G^{X_i}, see [9] for more details).

The process stops when $d^{X_i} \approx d^{X_{i+1}}$. We can expect this event to happen after very few iterations, as we now explain. A satisfactory coarse global solution should exist already in X_1, because the distance between far enough vertices is not sensitive to changes in the layout (this important point will be discussed in detail in the next section), so for far enough vertices u and v, $d^{X_0}(u,v)$ should be a good approximation to the £nal distance. This means that X_1 should be a reasonable global rendition of the £nal picture. Now, when the global structure is nice we expect the angles between the vertices to be close to their £nal values. But because the angle between any pair of vertices v and u determines $in^{X_1}(u,v)$, the function $d^{X_1}(u,v)$ should be a good approximation of the £nal distance. Hence we expect the second approximation, X_2 (which is optimized by $d^{X_1}(u,v)$) to already be close to optimal.

The experimental results given in Table 2 help con£rm this. In it we summarize the number of operations in each of the £rst four iterations. This number reversely indicates the quality of the layout of the previous iteration. As can be seen, a signi£cant part of the work is carried out in the £rst iteration, and in many cases the second iteration is devoted merely to £nal improvements.

This method, which we call the *Iterative Kamada-Kawai Method (IKKM)* deals with the same aesthetics as [9], but deals with non-point vertices that take up part of the drawing area. Convergence is very fast. If the process converges as expected — after two iterations — it will cost only slightly more then the regular Kamada-Kawai method (and its performance can be vastly improved using the multi-scale techniques, see Subsection 2.3).

However, more than the previous methods, IKKM is prone to vertex overlaps. The main reason is that the repulsive force, which is linear[1], is weaker than the inverse-squared forces of the spring embedder based methods. Another reason is that IKKM, unlike the spring-embedder based methods, does not neglect far-away vertices. It thus involves global considerations too, and may forgo some local considerations, which could mean violating constraints.[2]

[1]Following [9], we de£ned the cost-function using an energy. The implicitly de£ned underlying "forces" (i.e., the derivatives) conform to Hooke's law, and are linear in the spring length.

[2]This phenomenon can be relaxed by adjusting the constants $k_{v,u}$ to give a higher priority to the relations between close vertices.

If indeed the layout violates the constraints, we can increase the constant Len or use a locally-focused method to improve the layout, which would solve the local problems at very high speed. We describe a related method in Section 7.

Figure 4 presents layouts of some of the graphs shown previously, as produced by the IKKM. The layouts were achieved by two iterations only, and more iterations did not improve the results. For each graph we provide the average edge length($AvgLen$), and the ratio between the standard deviation of edge lengths and the average length: $StdDev/AvgLen$. This ratio is a normalized measure of the uniformity of edge lengths.

6. GLOBAL LAYOUT BY REDUCTION TO WEIGHTED GRAPHS

In this section, following the idea of the IKKM method of the previous subsection, we show that the problem of drawing a graph of arbitrarily sized vertices is reducible to the problem of drawing a restricted kind of a weighted graph (or, equivalently, to the problem of drawing a graph with circular vertices). We argue that two isomorphic graphs, for which every two large enough matching sets of vertices have a similar overall size, will be drawn approximately the same. The only difference between the drawings will lie in local phenomena, which include violation of constraints. This is because we expect that the differences between local arrangements compensate each other as one considers larger and larger portions of the drawing, converging around some average. The reduction is carried out by substituting non-point vertices with circles of related sizes, or, equivalently, by adjusting the edge lengths, in such a way that the shape of large clusters of vertices will remain the same. Fortunately, drawing the new graphs can be carried out by modifying virtually any force directed method. We begin with the de£nition of the restricted kind of a weighted graph to which we want to reduce the problem (similar to the construction in the previous section):

DEFINITION 6.1. *A* wt-graph *is a structure $G_{wt}(V,E,w)$, such that (V,E) is a graph and $w : V \longrightarrow \mathbb{R}$. Let Len be a constant. We de£ne the* weight *of an edge (u,v) as $w(u,v) = w(u)+w(v)+Len$. We may think of G_{wt} as a graph whose vertices are circles with $w(v)$ being the radius of v.*

The most important bene£t of the reduction is that it is able to "show" the global structure of the graph. For simplicity, we assume that there is an optimal layout with respect to a £xed set of aesthetic criteria accepted in force-directed algorithms. We term this layout *nice*.

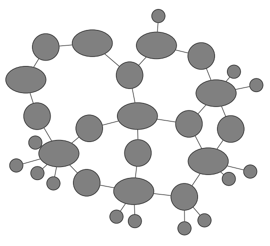
DEFINITION 6.2. *A layout X' is a* local change *of a layout X, with respect to some distance d, if for every vertex v, $|X'(v) - X(v)| < d$.*
A layout is globally nice *with respect to d, if it is a local change of a nice layout with respect to d.*

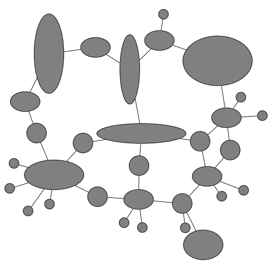
We now want to £nd a particular weight function, that will result in a graph with similarly sized clusters:

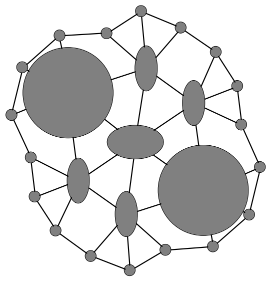
DEFINITION 6.3. *For a vertex v, denote by $g(v,\alpha)$ the length of the line-segment of angle α connecting the center of v to a point on its boundary.*

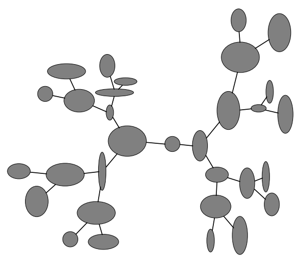
For example, if v is an ellipse with radii (r,R) then:

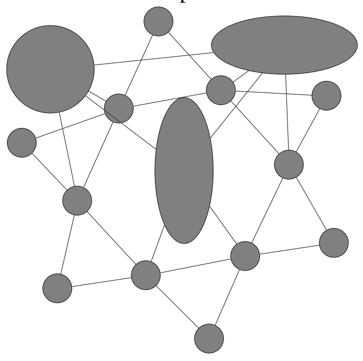
$$g(v,\alpha) = r \cdot R \cdot \sqrt{\frac{1+\tan^2\alpha}{R^2 + r^2 \cdot \tan^2\alpha}}$$

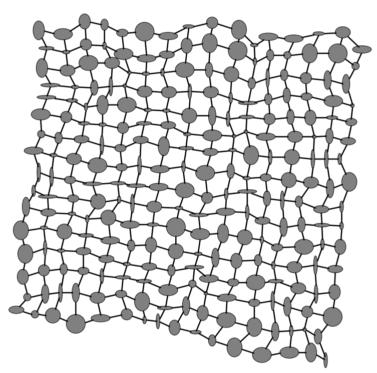

$AvgLen = 1.92, \frac{StdDev}{AvgLen} = 0.33$
Graph A


$AvgLen = 3.41, \frac{StdDev}{AvgLen} = 0.23$
Graph A1


$AvgLen = 5.02, \frac{StdDev}{AvgLen} = 0.34$
Graph B


$AvgLen = 3.57, \frac{StdDev}{AvgLen} = 0.25$
Graph C


$AvgLen = 4.43, \frac{StdDev}{AvgLen} = 0.35$
Graph D


$AvgLen = 5.62, \frac{StdDev}{AvgLen} = 0.16$
A 16x16 square grid

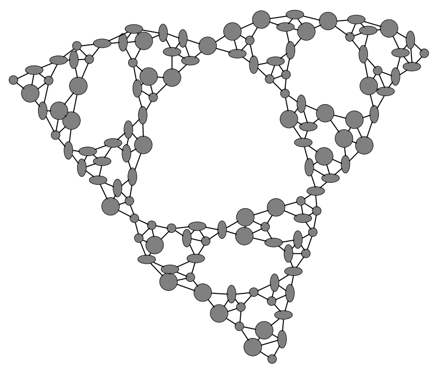

$AvgLen = 2.26, \frac{StdDev}{AvgLen} = 0.36$
A depth 4 Sierpinski

Figure 4: Results of the Iterative Kamada-Kawai Method

DEFINITION 6.4. *Let α be uniformly distributed in $[0, 2\pi)$. Denote by $f(v)$ the expected value of $g(v, \alpha)$. That is:*

$$f(v) = \int_0^{2\pi} \frac{1}{2\pi} g(v, \alpha)\, d\alpha$$

For example, if v is an ellipse with radii (r, R), then it can be shown that:

$$\frac{\sqrt{2} \cdot r \cdot R}{\sqrt{r^2 + R^2}} \leqslant f(v) \leqslant \sqrt{r \cdot R}$$

Both inequalities become equalities for $R = r$. We have chosen $f(v) = \sqrt{r \cdot R}$, so we can substitute the ellipse with a circle of the same area, whose radius is $f(v)$.

If v is a rectangle with sides of length r and R, respectively, then:

$$f(v) = \frac{1}{2\pi} \cdot \left(r \cdot \ln \frac{1 + \sin(\arctan \frac{R}{r})}{1 - \sin(\arctan \frac{R}{r})} + R \cdot \ln \frac{1 + \sin(\arctan \frac{r}{R})}{1 - \sin(\arctan \frac{r}{R})} \right)$$

The reduction

The problem of £nding a globally nice layout of a graph $G(V, E)$ with non-uniform vertices can be reduced to the problem of drawing the wt-graph $G_{wt}(V, E, f)$.

In way of supporting this claim, consider the GKK heuristic. Since $f(v) + f(u)$ is the expectation of $in^X(v, u)$, the probability that two matching paths in G and in G_{wt} will have the same length (with respect to the metric of Def. 5.2) grows with the number of vertices in the path. So for vertices far enough from each other the considerations taking into account by the GKK heuristic for the two graphs are exactly the same. This means that the only differences are local and lie in the relationships between close vertices; these should be drawn as closely as possible.

Nice layouts of wt-graphs can be achieved by any algorithm that is capable of drawing general weighted graphs: Kamada-Kawai [9] is a good example. The spring-embedder methods are not suitable for drawing general weighted graphs. But we do not have to draw a general weighted graph, only a restricted version of it, a wt-graph. Thus, the spring-embedder methods can be adapted to drawing $G_{wt}(V, E, f)$, by making the "ideal distance" between vertices non-uniform. Accordingly, The strengths of the forces in the Fruchterman and Reingold method [5] are changed to:

$$f_{attractive}(u, v) = l(u, v)^2 / k_{u,v}$$
$$f_{repulsive}(u, v) = -k_{u,v}^2 / l(u, v)$$

where $l(u, v)$ is the distance between the vertex centers and $k_{u,v}$ is correlated with $f(v) + f(u)$.

We can now improve the Iterative Kamada-Kawai method in the following way. Instead of basing the initial metric d^{X_0} on a random layout X_0, we can de£ne it as the graph-theoretic distance between vertices in $G_{wt}(V, E, f)$, where f is as in Def. 6.4. In such a way, d^{X_0} is a better approximation of the £nal metric.

7. THE COMBINED METHOD

As a consequence of the previous section, we now argue that we can divide the problem of £nding a nice layout of a graph with arbitrarily sized vertices into two complementary problems. The £rst consists of £nding a globally nice layout and the second of optimizing that layout locally. The two constraints C1 and C2 of Section 3 refer only to small areas of the drawing, so we address them only in the local optimization problem.

Each of the two problems has an advantage that is a drawback of the other. The problem of £nding a globally nice layout has the advantage that it can be reduced to the problem of drawing a wt-graph, which does not have any special constraints and can be solved quickly using familiar methods even for large graphs. On the other hand, the £nal optimization problem has to deal with only a small number of vertices, and thus can use rigorous, albeit slow, methods. We now try to exploit these observations.

The Combined Method

1. *Constructing a globally nice layout:*
   Use the Kamada-Kawai method for £nding a nice layout of $G_{wt}(V, E, f)$.
2. *Local beauti£cation:*
   Use the Modi£ed (or Generalized) Spring Method with a very low temperature (about a dozen sweeps) for improving the result of Step 1.

Remarks

In Step 1 any method for drawing wt-graphs can be applied. For example, large graphs can be drawn by multi-scale methods. (See Subsection 2.3.) We can improve the speed of Step 2 by considering only small neighborhoods at a time, as in [7, 8]. This leads to a signi£cant performance gain when the graph is large.

We £nd that the Combined Method is the best of all our methods. It integrates results that nicely satisfy the constraints with ef£ciency of operation. The £rst step of constructing a globally nice layout takes the dominant fraction of the running time. This step is implemented using the same algorithms that are used for drawing graphs with dimensionless vertices. Hence, the speed of the combined method is very close to those methods that draw graphs with dimensionless vertices.

Figure 5 contains layouts of some of the previously shown graphs, as produced by the Combined Method.

8. EDGE-VERTEX OVERLAPS

The combined method of Section 6 is the general way we propose to solve the problem this paper addresses. However, this method does not properly deal with the second constraint — preventing overlaps between vertices and edges. Fortunately, in typical outputs of our methods, we need not worry about encountering *many* such overlaps:

CLAIM 8.1.
Denote the length of the longest edge $MaxLen$. A drawing contains no edge-vertex overlaps if the distance between the boundaries of every two vertices is no less than $MaxLen/2$.

PROOF. By contradiction. Consider a typical case of edge-vertex intersection: given three vertices u, v, w, such that vertex v intersects edge (u, w) (see Figure 6). Suppose that the related layout satis£es the condition in the claim. Divide the edge (v, u) into three segments: the segment connecting the boundaries of u and v, whose length is l_{uv}, the segment passing inside v, whose length is $l_v > 0$, and the segment connecting the boundaries of v and w, whose length is l_{vw}. Clearly, the length of (u, w) is $l(u, w) = l_{uv} + l_v + l_{vw}$. Hence, we get:

$$l(u, w) > l_{uv} + l_{vw} \quad (*)$$

Since the layout satis£es the condition in the claim, we know that the distance between the boundaries of the vertices is at least $MaxLen/2$. Hence, $l_{uv} \geq MaxLen/2$ and $l_{vw} \geq MaxLen/2$. Substituting in (*) results in: $l(u, w) > MaxLen/2 + MaxLen/2 = MaxLen$, contradicting the fact that $MaxLen$ is the maximal edge length. $\square$

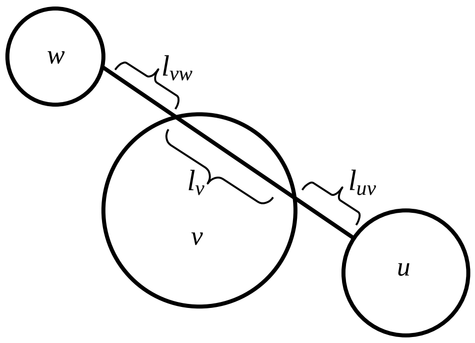


Figure 6: Typical vertex-edge crossing

As a consequence of Claim 8.1, ful£lling the following two aesthetic criteria, which our drawing algorithms clearly seek to attain, implies no edge-vertex overlaps:

A1 Uniform edge lengths.

A2 Distance between non-adjacent vertices is longer than the (uniform) edge length.

The reason for this is that as long as an edge is drawn as the shortest line segment that connects the vertices boundaries[3], the condition in claim 8.1 is equivalent to requiring that the ratio between the maximal edge length and the minimal edge length is less than 2 (which A1 implies), and that the distance between non-adjacent vertices is at least half of the maximal edge length (which A2 implies).

In practice, although the two aesthetic criteria, A1 (uniform edge length) and A2 (adjacency closeness), are not fully achieved, we expect that in a layout that strives to comply with these criteria the deviation between the lengths of the edges would not be large, so edges will rarely intersect vertices.

For those cases in which some edges intersect vertices, it would be nice if we had a dedicated *local* beauti£cation step that would eliminate cases of overlap between edges and vertices. For an already nice drawing, there is need for only local changes, since the edges are short and it is very unlikely that a single edge will overlap with more than a single vertex. It is important that this £nal step include the other aesthetics issues, so it will not ruin the aesthetic properties already achieved.

A candidate method for this can be simulated annealing, which was £rst used for graph drawing in [1]. Simulated annealing is a general stochastic method for optimizing energy functions that include discrete terms, It can thus be used to explicitly “punish” overlaps. When the input layout is close to optimal, simulated annealing will quickly £nd an overlap-free layout.

Another possibility is to add repulsive forces between vertices and edges (that really act between one vertex and a pair of adjacent vertices). Details can be found e.g., in [11, 1]. We have not yet implemented these forces, but we believe they could be very ef£cient.

If after all this, there still exist edge-vertex intersections, one can give up straight-line edges and draw edges as curves, bypassing the vertices. Routing an edge should be relatively easy since it typically has to bypass only a single vertex. The edge routing algorithm of [2] may be used for this.

[3]If the edges are drawn on the straight-line segment that connects the centers, we can treat their length as an approximation of the distance between adjacent vertices.

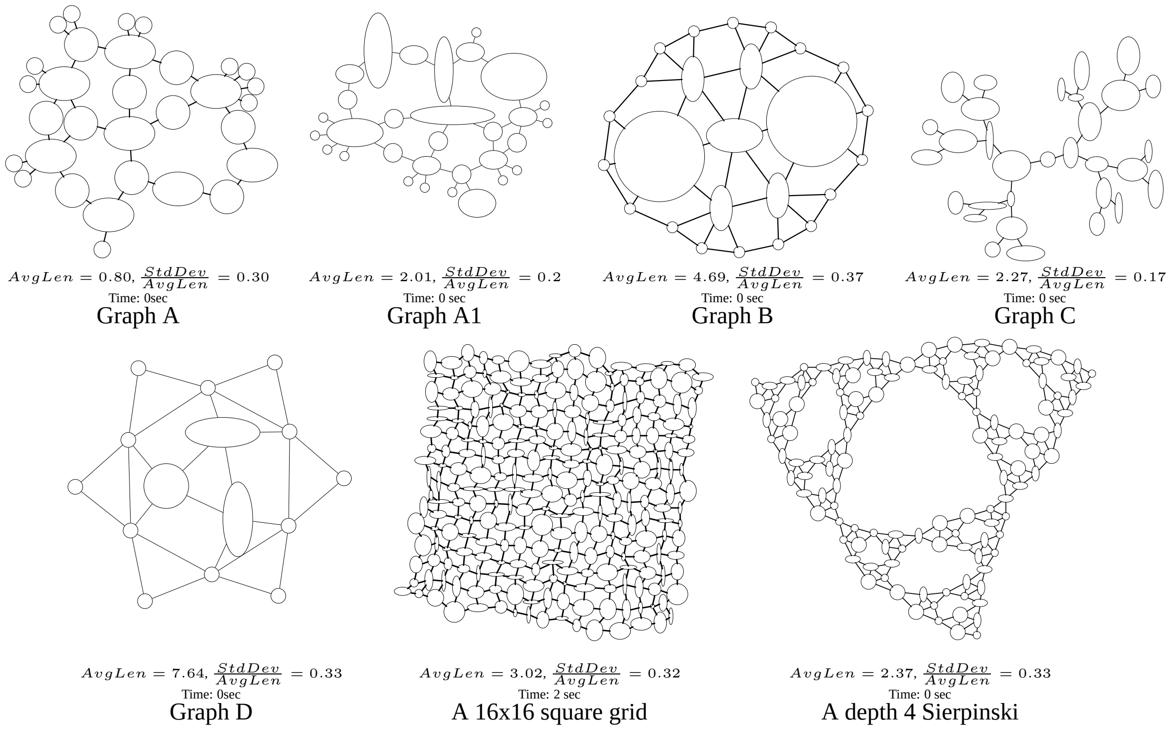

$AvgLen = 0.80, \frac{StdDev}{AvgLen} = 0.30$
Time: 0sec
Graph A
$AvgLen = 2.01, \frac{StdDev}{AvgLen} = 0.2$
Time: 0 sec
Graph A1
$AvgLen = 4.69, \frac{StdDev}{AvgLen} = 0.37$
Time: 0 sec
Graph B
$AvgLen = 2.27, \frac{StdDev}{AvgLen} = 0.17$
Time: 0 sec
Graph C
$AvgLen = 7.64, \frac{StdDev}{AvgLen} = 0.33$
Time: 0sec
Graph D
$AvgLen = 3.02, \frac{StdDev}{AvgLen} = 0.32$
Time: 2 sec
A 16x16 square grid
$AvgLen = 2.37, \frac{StdDev}{AvgLen} = 0.33$
Time: 0 sec
A depth 4 Sierpinski

Figure 5: Results of the Combined Method

9. EDGE LENGTH CONSIDERATIONS

The reader might have noticed that all our algorithms have a parameter Len that re¤ects the desired edge length. This parameter plays a rather important role in our work, as it distinguishes graphs with large vertices from standard ones. In laying out a standard graph the edge length is of no signi£cance; it is only a matter of scaling up the picture. In laying out a graph with large vertices the situation is more complicated: the edge lengths are not the only sizable entities in the picture, so the desired length must depend on the sizes of the vertices too, and actually this length is an inherent factor in a resulting layout.

Here are two examples of our algorithms' results, that clarify this point:

Figure 7 shows two layouts of the same graph. In Figure 7(a) the length of the all edges is equal to 1.61 units (where the radius of each circle is 1 unit), and we have a perfectly symmetric result. In Figure 7(b) the average length of the edges is shorter, only 1.29 units, but we have a badly symmetric layout. The reason for the different results is in the constant Len. In Figure 7(b) the value of Len was too small, so the algorithm crowded the vertices together, and their areas have a negative impact on the £nal layout.

Figure 8 illustrates a case in which, if the edges are too short, the surrounding circle of vertices in the good layout of Figure 8(a) is not large enough to enclose the two large interior vertices, resulting in the non-planar layout of Figure 8(b).

It seems that the problem of optimizing the edge length is actually the problem of *minimizing* the edge length, but so that it still yields an "intuitive" picture. We have incorporated this observation into our toolkit of algorithms, with a program that can be used to search for the optimal edge length by running the more substantial methods described in this paper with various edge lengths. The program initializes the edge length to a large value and then decreases it, using binary search, until something "bad" occurs, e.g.,

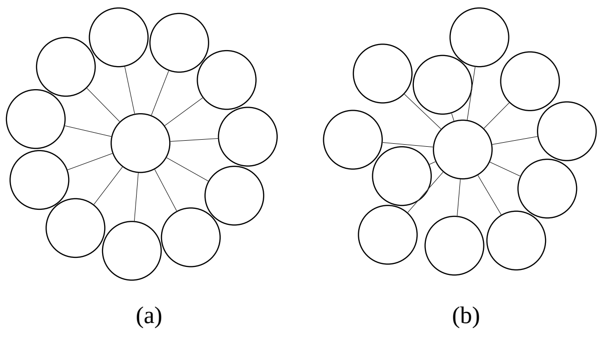


Figure 7: Short edges may con¤ict with symmetry

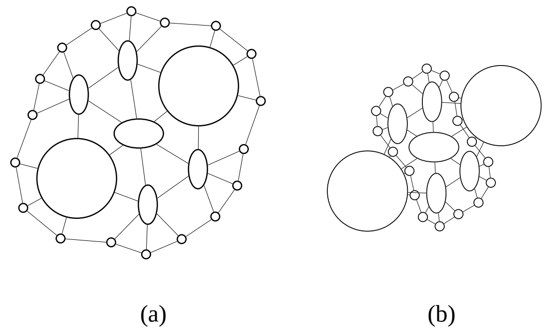


Figure 8: Too short edges may con¤ict with planarity

loss of an important aesthetic criterion like planarity or uniformity of edge lengths, or a violation of one of the two constraints (C1 or C2). Each iteration of the search is initialized to the result of the previous iteration, so it converges quickly. Notice that initialization with dimensionless vertices is equivalent to setting an in£nite edge length.

10. RELATED WORK

Various changes to the force model of classical force-directed methods to handle vertex overlaps were introduced in several papers, e.g., [10, 12, 11]. (However, all these changes are related to the spring-embedder. To our best knowledge, the GKK heuristic (in Section 5) is the £rst generalization of the Kamada-Kawai energy to handle non-uniform vertices.) As mentioned, such changes are not enough. If the forces are too weak, one has to reserve a wasted area around the vertices to prevent overlaps. On the other hand, when the forces are strong and their £eld tenses tightly around the vertices, the convergence would be very slow, if at all.

An alternative approach is developed in the interesting paper of Gansner and North [6]. Their method is carried out in three phases. The £rst phase draws the graph using the algorithm of [9], regardless of the vertex shapes. The second phase removes overlapping vertices by iteratively constructing a Voronoi diagram using vertex centers as sites and moving vertices to the centers of their Voronoi cells. When needed a surrounding bounding box is enlarged, giving more area for the drawing. The process ends when there is no overlap. The third phase draws edges as smooth curves to avoid edge-vertex overlaps. Such a third phase is absent in our method and may be helpful. Since it is independent of the previous two phases, it can be incorporated into our method, as a post-processing stage. However, regarding the £rst two stages we think that our approach is advantageous, as we explain now.

Since the voronoi diagram is based solely on the vertices' centers, the movement of vertices in the second phase of [6] does not treat the vertex shapes, so it cannot achieve a layout as compact as ours. Moreover, when the iterative movement does not succeed in preventing all overlaps, the bounding box of the drawing is enlarged, which is equivalent to a global, possibly wasteful, growth of the drawing area, increasing the distances even between non-overlapping vertices. For example, compare our results for graphs A1 and D in Figures 1–5, with those of [6] reproduced in Figure 9 (scaled down). We intentionally set the parameters of [6] so as to minimize the drawing area. As can be seen, their results occupy much more space than ours. Moreover, the results of [6], seem to be less pleasing aesthetically compared to our results. However, if we were to let [6] use more space their results would be comparable to ours. This brings us to another advantage of our approach. In [6], the problem is divided into two parts, one is solely responsible for the aesthetics and the other is responsible for satisfying the constraints without knowing about aesthetics. We fully agree with the divide and conquer strategy adopted there that accelerates the running time, but we think that the correct division should be different: a division between global aesthetics and local aesthetics. Our division is built upon two parts that enrich each other and are "aware" of the same aesthetic issues.

11. CONCLUSIONS

We have investigated the problem of drawing general undirected graphs with arbitrarily sized vertices, and have attempted to formulate and analyze the special issues that arise when tackling this problem. Several algorithms of varying speed and quality for laying out such graphs are suggested. We have found that oversimpli£ed application of known methods yields non-aesthetic pictures, and on the other hand, careless extension of the cost function leads to unsatisfactory convergence time. Our £nal methods are smooth extensions of methods for drawing graphs with dimensionless vertices in the sense that they are not worse than those methods, in terms of speed and output quality. In fact, we have suggested a fast reduction of the problem of drawing graphs with arbitrarily sized vertices to the problem of drawing a restricted kind of weighted graph, which can be solved by small changes to any force-directed method we are aware of.

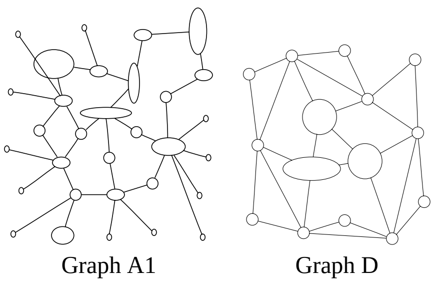


Figure 9: Results of Gansner-North [GN98]

12. REFERENCES

[1] R. Davidson and D. Harel, "Drawing Graphs Nicely Using Simulated Annealing", *ACM Trans. on Graphics* **15**:4 (1996), 301-331.

[2] D. Dobkin, E. Gansner, E. Koutso£os and S. North, "Implementing a General-Purpose Edge Router", *Proceedings of Graph Drawing 97*, LNCS **1353**, pp. 262-271.

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

[4] P. Eades, "A Heuristic for Graph Drawing", *Congressus Numerantium* **42** (1984), 149-160.

[5] T.M.G. Fruchterman and E. Reingold, "Graph Drawing by Force-Directed Placement", *Software-Practice and Experience* **21**:11 (1991), 1129-1164.

[6] E. Gansner and S. North, "Improved Force-Directed Layouts", *Proceedings of Graph Drawing 98*, LNCS **1547**, pp. 364-373.

[7] R. Hadany and D. Harel, "A Multi-Scale Method for Drawing Graphs Nicely", *Discrete Applied Mathematics* **113** 3-21 (2001).

[8] D. Harel and Y. Koren, "A Fast Multi-Scale Method for Drawing Large Graphs", *Proceedings of Graph Drawing 2000*, LNCS **1984** , pp. 183–196.

[9] T. Kamada and S. Kawai, "An Algorithm for Drawing General Undirected Graphs", *Information Processing Letters* **31** (1989), 7-15.

[10] T. Kamps, J. Kleinz and J. Read, "Constraint-Based Spring-Model Algorithm for Graph Layout", *Proceedings of Graph Drawing 95*, LNCS **1027**, pp. 349-360.

[11] D. Tunkelang, *A Numerical Optimization Approach to General Graph Drawing*, Ph.D. Thesis, Carnegie Mellon University, 1999.

[12] X. Wang and I. Miyamoto, "Generating Customized Layouts" *Proceedings of Graph Drawing 95*, LNCS **1027**, pp. 504-515.