Uncluttering Graph Layouts Using Anisotropic Diffusion and Mass Transport

Yaniv Frishman and Ayellet Tal

***Abstract*— Many graph layouts include very dense areas, making the layout difficult to understand. In this paper, we propose a technique for modifying an existing layout in order to reduce the clutter in dense areas. A physically-inspired evolution process, based on a modified heat equation is used to create an improved layout density image, making better use of available screen space. Using results from optimal mass transport problems, a warp to the improved density image is computed. The graph nodes are displaced according to the warp. The warp maintains the overall structure of the graph, thus limiting disturbances to the mental map, while reducing the clutter in dense areas of the layout. The complexity of the algorithm depends mainly on the resolution of the image visualizing the graph and is linear in the size of the graph. This allows scaling the computation according to required running times. It is demonstrated how the algorithm can be significantly accelerated using a graphics processing unit (GPU), resulting in the ability to handle large graphs in a matter of seconds. Results on several layout algorithms and applications are demonstrated.**

***Index Terms*— Graph layout, graph visualization, GPU, anisotropic heat equation, mass transport.**

I. Introduction

Graph drawing addresses the problem of constructing geometric representations of graphs [1], [2]. It has applications in a variety of areas, including software engineering, software visualization , social networks and biology . A variety of graph layout algorithms exist. Each of these algorithms is suited for different applications. A few examples include hierarchical, planar, circular, orthogonal, and force directed layout [1], [2].

Graph layouts often contain a highly varying local density. While some regions in the generated layouts are sparse or even empty, others are very dense, containing many close-by or overlapping edges and nodes. This results in low efficiency in utilizing the available screen space.

Instead of developing a new layout algorithm, this paper describes an algorithm that can improve a given graph layout. This allows the user to select a layout algorithm that is suited for the application at hand. The clutter in the layout can then be reduced by our algorithm, resulting in a layout with a smaller node density in the high-density regions of the original layout. This is achieved while preserving the overall structure of the graph. Figure 1(a) shows an example of a cluttered layout. The layout is difficult to read and the available screen space is not used effectively. Figure 1(b) shows the enhanced layout. Note how the screen space is more efficiently used, allowing more details of the graph to become visible.

Some research has addressed the problem of reducing the visual clutter of graph layouts in the past. Lyons et al. [3] use a combination of a Voronoi diagram and a force-directed type approach [4]–[6] in order to disperse nodes clustered together. Merrick and Gudmundsson [7] modify the layout based on properties of the structure of the underlying graph. However, these algorithms employ schemes that are either computationally expensive or perform local improvements to the graph. In contrast, the algorithm in this paper is able to operate on large graphs, making a more global enhancement to the layout.

Instead of operating on the abstract graph representation, the algorithm proposed in this paper operates on an image of the density of the input layout. The density image is modified, making use of low-density regions in order to reduce the visual complexity in high-density regions of the layout. A physically-inspired evolution of the density image using a modified heat diffusion process is used to create the target density image. Given the target density, a warp of the 2D layout is computed, in which dense regions are allowed to expand and make use of available screen space. The warp is computed using results from optimal mass transport problems [8]–[10]. The evolution process attempts to retain the overall structure of the input graph, limiting potential disturbances to the user's mental map as outlined in Misue et al. [11].

This paper makes a couple of contributions. First, a new algorithm for uncluttering graph layouts in a mental-map preserving fashion is presented. Second, a method for accelerating the computation of the target density, which is the most time-consuming stage of the algorithm, using a graphics processing unit (GPU), is described. Several examples, using various layout algorithms and applications, are provided to demonstrate the capabilities of the algorithm.

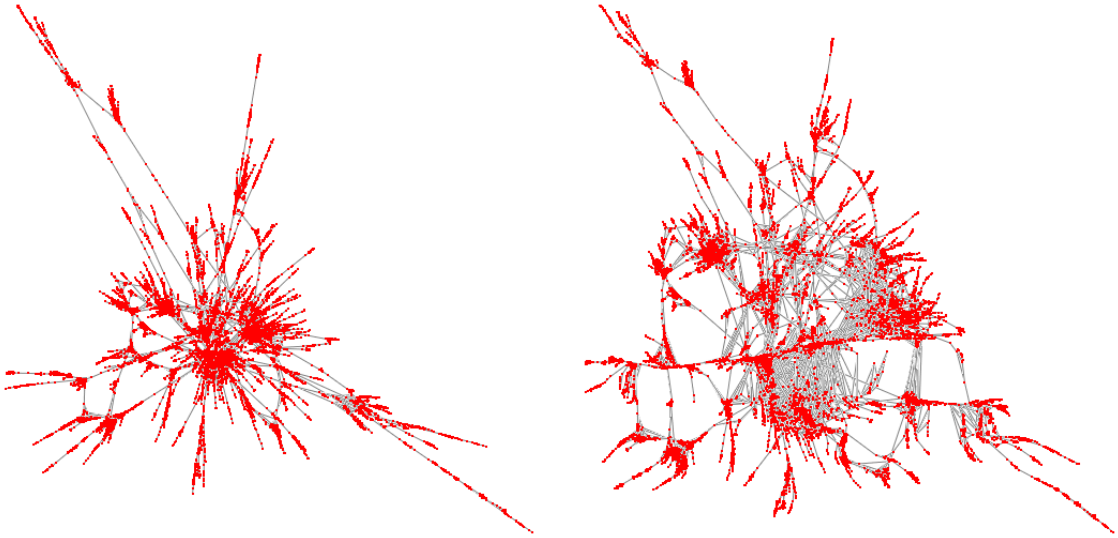
II. Related Work

This work is related to three sub-fields: algorithms for graph uncluttering, node overlap removal in graph drawing, and applications in areas outside of graph drawing. In this section we discuss related work in these fields.

Several papers have addressed the graph uncluttering problem. Lyons et al. [3] attempt to more evenly distribute the nodes while maintaining the user's mental map of the original layout. Two algorithms are presented. The first uses a Voronoi diagram in order to move nodes. The second algorithm repositions nodes inside a region defined by a Voronoi diagram, according to the forces acting on them, defined using a force-directed approach [4]–[6]. Using a Voronoi diagram performs only local enhancements, which may not be sufficient in order to reduce clutter in dense areas of the graph.

Merrick and Gudmundsson [7] propose a technique for enlarging dense areas of a given graph layout and shrinking sparse areas. Their algorithm first determines the important nodes, then calculates the desired edge lengths, and finally repositions vertices

Yaniv Frishman is with the Department of Computer Science, Technion - Israel Institute of Technology. E-mail: yfrishman@hotmail.com. Ayellet Tal is with the Department of Electrical Engineering, Technion - Israel Institute of Technology. E-mail: ayellet@ee.technion.ac.il



(a) (b)

Fig. 1. Protein graph (V=30727, E=1206654). (a) FM^3 [12] layout. (b) Improved layout. Note how displacing nodes outwards allows more details to become visible, especially in the center of the drawing. Also note that the overall structure of the graph is maintained.

using the algorithm of Shimizu and Inoue [13], which tries to minimize the change in the angles of the edges. Determining the important nodes, called *node centrality*, is an expensive operation, taking $O(V \cdot E)$ for V nodes and E edges. It is thus not scalable to large graphs. Centrality is determined according to graph-theoretic properties of the underlying graph, which do not take the actual layout into account. Therefore, the algorithm is not effective at uncluttering dense areas of the graph with non-central nodes. Our algorithm attempts to solve these problems.

There are two related, yet distinct, problems to graph uncluttering: graph overlap removal and overlap removal in other fields such as map cartography. Hereafter we describe some related work on these issues.

While most graph drawing algorithms assume that nodes are dimensionless (e.g. point-sized), in practice nodes may be labeled, and the labels may overlap. Several algorithms have been developed to remove overlaps between nodes.

Chuang et al. [14] use potential fields in order to remove overlaps. Gansner and North [15] use an iterative Voronoi diagram method in order to tidy up the layout. Harel and Koren [16] use a combination of a Kamada Kawai [5] method and a modified spring method, which takes node shapes into account when calculating forces in order to converge to an overlap free layout. Marriott et al. [17] use a constrained optimization approach in order to remove overlaps. Eades and Nikolov [18] remove overlaps using spring algorithms, followed by displacement of nodes in a way that preserves the mental map as measured by the orthogonal node ordering model. Huang et al. [19] discuss the force-transfer algorithm which pushes overlapping nodes away from each other. Dwyer et al. [20] use a constraint optimization problem for each dimension separately.

The graph uncluttering problem addressed in this paper is different from the node overlap removal problem. Overlap removal attempts to compute a minimal displacement of nodes in order to avoid overlaps, but may result in graphs that are still difficult to comprehend since they include very dense areas. Moreover, while the algorithms discussed above deal with removing overlaps between a small number of large, labeled nodes, our algorithm attempts to improve layouts of large, dense graphs in a mental-map preserving fashion. In addition, graph uncluttering attempts to maintain the original structure of the graph, while overlap removal does not necessarily have this aim. Finally, while overlap removal algorithms guarantee a final drawing free of node overlaps, graph uncluttering algorithms do not necessarily guarantee this property.

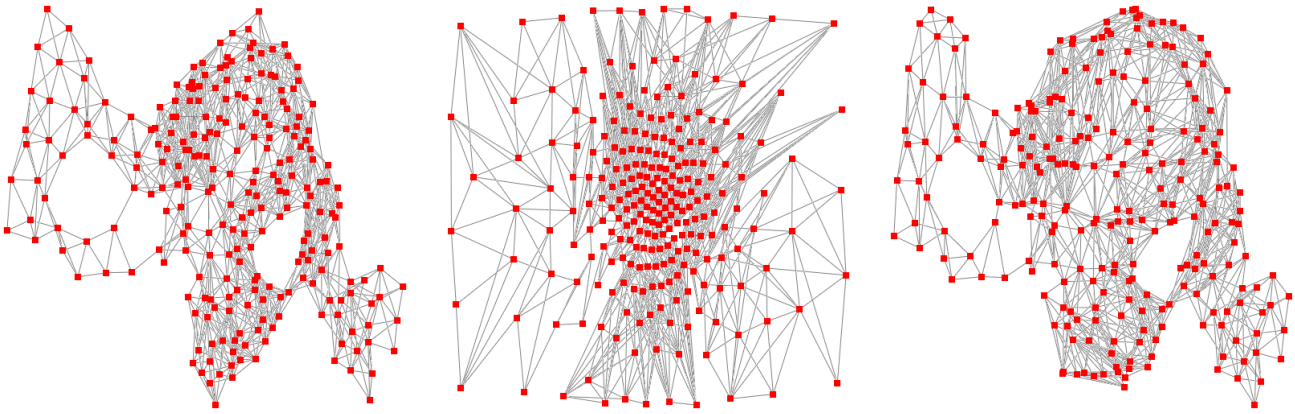
Figure 2 shows a comparison between the results of using a node overlap removal algorithm [15] and using our graph uncluttering algorithm. It can be seen that the overlap removal algorithm not only modifies the structure of the graph, but also leaves some dense areas (Figure 2(b)). Our uncluttering algorithm improves the layout in a mental-map conserving manner by expanding the graph to empty regions (Figure 2(c)).

Overlap removal and graph uncluttering problems arise in other fields outside of graph drawing. Deussen et al. [23] present an extension of Lloyd's method for distributing objects on the plane in order to create stipple drawings. Chan et al. [24] use a density constrained minimization formulation in order to compute overlap-free placements for components in integrated circuits. Hayashi et al. [25] present an $O(n^2)$ algorithm for finding the minimum area layout of a set of n rectangles that avoids intersections and preserves the orthogonal ordering of the rectangles.

Map cartography attempts to create maps in which the size of regions is in proportion to their population or some other analogous property. Gastner and Newman [26] perform diffusion in order to create maps which have a uniform information density. There are a couple of differences between their work and this paper. First, in cartography an attempt to conserve the area is made, while our algorithm tries to use sparse or empty regions of the screen. Second, while in [26] isotropic diffusion is used, here anisotropic diffusion is used in order to avoid "collisions" between neighboring dense areas of the graph.

III. The Algorithm

Given $L_{initial}$, which is a straight-edge layout of an un-directed graph $G = (V,E)$, the goal of the algorithm is to produce an



(a) Input layout (V=247, E=1230) (b) Removing node overlaps (c) Uncluttering using our algorithm

Fig. 2. Comparison between node overlap removal and graph uncluttering. (a) is a layout produced using *neato* [21] of a reduced version of the bcsstk32 graph from [22]. In (b) the node overlap removal algorithm from [15] is used. Note that although the overlaps between nodes are eliminated, the structure of the graph is not maintained and the center of the layout is cluttered. In (c) our algorithm is used. Note how the cluttered right side of the input layout is expanded, thus increasing node separation, while the structure of the graph is maintained.

enhanced layout L_{final}. This layout should make better use of the available screen space by dispersing nodes from high density regions to surrounding regions, while maintaining the structure of the original layout. The algorithm utilizes several key ideas. First, for each pixel in the image of the layout, we compute the density of the information it contains. Second, we perform an evolution process in order to improve this density, making use of unused areas of the image and reducing the density in congested areas. Third, a warp is computed between the initial and the improved densities. This image warp is used to modify the graph layout in a way that helps preserve the mental map, resulting in an enhanced layout. Algorithm 1 gives an overview of the steps of the algorithm. We elaborate on each of these steps below.

Algorithm 1 Layout improvement algorithm

input: $L_{initial}$, layout of a graph G=(V,E)
output: L_{final}, modified layout of G

1) Compute $D_{initial}$, the density image of the layout $L_{initial}$.
2) Calculate D_{smooth}, a smoothed density image of $L_{initial}$, using the heat equation.
3) Calculate D_{target}, the target density image, using a modified heat evolution.
4) Calculate an optimal mapping $\tilde{u}$ between D_{smooth} and D_{target}.
5) Calculate L_{final} by displacing nodes according to the mapping $\tilde{u}$.

Computing the density image of the layout (Step 1): The first step of the algorithm computes the density $D_{initial}$ of the given layout $L_{initial}$, as illustrated in Figure 3(a) and (c). The intensity of each pixel in the density image is proportional to the number of graph elements that cover the pixel. Using the density image, the cluttered areas of the graph, which we wish to visualize more clearly, can be identified.

The density image can be computed using only the nodes or both the nodes and edges of the graph. Our experiments indicate that using only the nodes produces better results. This is since each edge has a rigid structure, while node concentrations consist of individual points which can be dispersed by our algorithm to generate a more understandable layout. The resolution of the computed image is configurable by the user. While small grids reduce the running time of the algorithm, the quality of the results can suffer, especially for large, dense graphs. In our experience, using a resolution of 257 by 257 pixels gave good results at a reasonable running time for a large variety of graphs, and thus was used as the default. (Note that the multigrid algorithm requires a resolution equal to $k \cdot 2^m + 1$ where $k, m \in \mathbb{N}$ (see Section IV) [27].)

In our implementation, the density is computed using OpenGL and the GPU. Since we are interested in identifying areas where several graph elements (i.e. nodes) occupy the same screen pixel (i.e. overlap), we use *blending* in order to accumulate the density. This is achieved by using a rendering mode in which the color of different overlapping rendered primitives is accumulated. Thus, pixels that contain more graph elements will have a higher value in the density image. Anti-aliasing is used to render a smoother image.

Note that in this paper density images are used to compute an improved layout. However, there can be other uses of density images. For instance, in [28] they have been used to aid in visualization.

Smoothing the density image (Step 2): In this step the image $D_{initial}$ is smoothed it in order to create the image D_{smooth}. This is a pre-processing phase that creates an input that is more suitable and hence improves the numerical stability of the warping algorithm in Step 4.

We base the smoothing algorithm on the *heat equation* [29]. This is a partial differential equation (PDE) that models the variation of the temperature in a region over time. Intuitively, this PDE implies that the rate of change in temperature over time depends on the temperature difference between a point and its neighbors. The PDE describes a diffusion process that can be used for smoothing. In addition, it has the desirable property that given a potentially discontinuous initial temperature, it very rapidly becomes continuous.

Given a 2D domain Ω we define the temperature in each point in the domain as $u(x,y)$. The heat equation is

$$\frac{\partial u}{\partial t} = k\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \equiv k\nabla^2 u, \tag{1}$$

where ∇^2 is the *Laplacian* operator and k is a constant describing

the rate of heat diffusion. In our case, $u(x,y)$ is set to the density $D_{initial}(x,y)$ computed in Step 1 and it is evolved to compute the smoother density $D_{smooth}(x,y)$. Appropriate boundary conditions need to be set on the values of u. We define $u=0$ on the boundary $\partial\Omega$, corresponding to setting a zero density at the boundary of the image of the layout.

To solve this equation numerically it is necessary to discretize the grid and use numerical approximations for derivatives [30]. This results in the following discrete approximation of Equation 1:

$$\frac{u^{t+1}(i,j)-u^t(i,j)}{dt} = k\frac{u^t(i+1,j)-2u^t(i,j)+u^t(i-1,j)}{(dx)^2} + k\frac{u^t(i,j+1)-2u^t(i,j)+u^t(i,j-1)}{(dy)^2}, \quad (2)$$

where $u^t(i,j)$ is the value of the density at grid point (i,j) at time step t, dx and dy are the grid dimensions in the x and y directions, respectively, dt is the time step and $u^0(x,y)=D_{intitial}(x,y)$. Thus, given the density at every grid point at time t we are able to compute the density at time $t+1$. Figure 3 (c) and (d) shows the smoothing performed by the heat equation. The Laplacian operator on the right-hand side of Equation 2 can be represented by the following template [30]:

$$\nabla^2 \approx \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \quad (3)$$

which describes how the values in each grid point are updated, taking its neighbors into consideration.

It should be noted that it is possible to perform the smoothing by performing a convolution with the heat kernel. The iterative formulation discussed here serves as a basis for the anisotropic case discussed in Step 3.

The algorithm uses several parameters. We use a square grid and therefore set $dx=dy=1$. Using $k=1$ in the heat equation results in a reasonable diffusion rate. In order to maintain numerical stability, it is required to have $dt \leq \frac{1}{8}\frac{(dx)^2+(dy)^2}{k}$ [31]. We use $dt=0.23$. Thirty iterations of Equation 2 are run. This number represents a tradeoff. If too few iterations are used, the smoothing will not be sufficient for Step 4. If too many iterations are used, the image will be too smooth, potentially reducing the displacements computed in Step 4.

Calculating the target density image (Step 3): Although the algorithm in Step 2 has the advantage of creating a more uniform, evenly distributed density, it has the disadvantage that the diffusion process takes into account only local properties of the density, as governed by the heat equation. This is not desirable in our case since it may lead to cases of "collisions" between close-by high density regions. We would like to take the topology of the given graph density into consideration when calculating an alternative, more uniform density with lower maximal values, corresponding to a less cluttered layout. The goal of this step is to compute D_{target}, which is an improved density image, given $D_{initial}$.

Creating an improved, shape-aware density image is achieved by modifying the evolution described by the heat equation (Equation 1). Instead of performing isotropic diffusion as governed by the discrete Lapalcian operator (shown in Matrix 3), we modify the direction of the diffusion according to the shape of the density image. The diffusion is performed in a direction that makes use of empty and low-density regions of the image. This allows making more effective use of the screen space in the improved layout L_{final}.

To select the preferred direction θ_{best} at each time step and for each pixel of the current density image μ, a ray-shooting process is performed. For location (x,y) in the density image, given a possible diffusion direction θ, we calculate the following score

$$score(x,y,\theta) = \int_{l=0}^{l=l_{max}} \mu(x+lcos\theta, y+lsin\theta)\,dl,$$

where l_{max} corresponds to a point on the ray that is on the image boundary. The intuition behind this formula is that we sum up the amount of material we encounter when traveling in direction θ from (x,y) up to the boundary of the density image. In discrete form, the score is

$$score(x,y,\theta) = \sum_{l=0}^{l=\lfloor l_{max} \rfloor} \mu(x+lcos\theta, y+lsin\theta). \quad (4)$$

The final advancement direction is

$$\theta_{best}(x,y) = \underset{\theta\in[0,2\pi]}{argmin}\{score(x,y,\theta)\}, \quad (5)$$

which corresponds to the direction in which the least amount of material is encountered, hence making the best use of available screen space (since we disperse the material to the emptiest regions).

Since there are potentially several nodes located in the same pixel of the density image μ, it is required to use sub-pixel accuracy in the sampling performed in Equation 4. This is efficiently handled by using bilinear interpolation for sampling μ. Using higher fidelity kernels is also possible, but would result in a significant decrease in performance.

Given θ_{best} for every pixel in the current density image, we evolve the density according to equation 1, but replace the isotropic Laplacian operator in Matrix 3 with the following anisotropic operator:

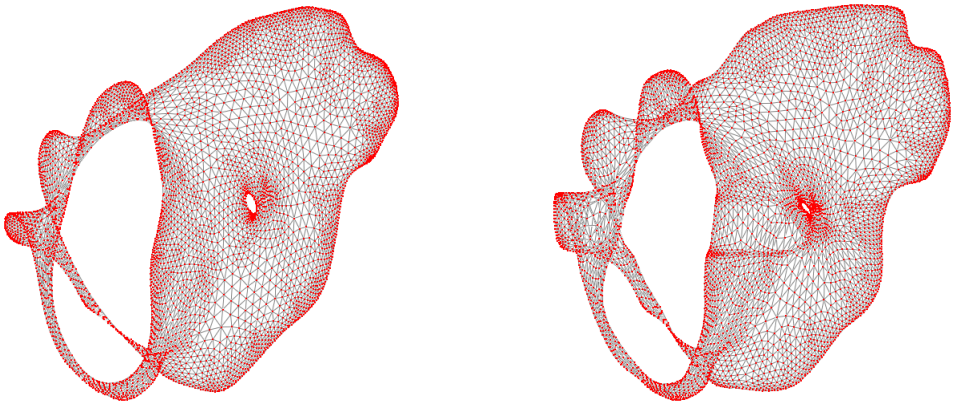
$$\nabla^2_{anisotropic} \approx \begin{pmatrix} 0 & 1+sin(\theta_{best}) & 0 \\ 1+cos(\theta_{best}) & -4 & 1-cos(\theta_{best}) \\ 0 & 1-sin(\theta_{best}) & 0 \end{pmatrix}. \quad (6)$$

The intuition behind this operator is that the averaging performed depends on the direction θ_{best}, resulting in a new density that is biased in the required direction.

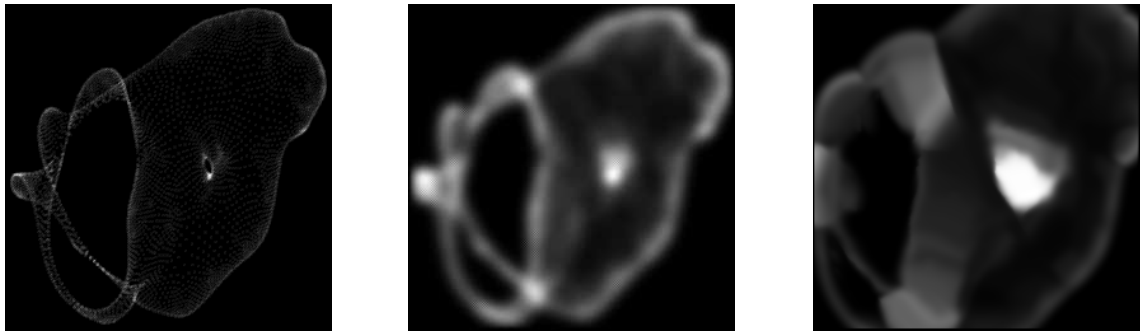
In summary, in this step, starting with $\mu = D_{initial}$, we iteratively compute Equation 5 and update μ using the anisotropic Laplacian Matrix 6, resulting in D_{target}.

In our implementation we calculate the best diffusion direction for 64 angles symmetrically distributed over the possible advancement directions (i.e. $[0,2\pi]$). Five iterations of the heat equation evolution (using Matrix 6) are performed between recalculations of the best direction (Equation 5). This is a tradeoff between computation speed and accuracy, which our experiments show produces good results. A total of 60 iterations of the heat equation evolution are performed. This number is used in order to ensure that the evolution of the target density D_{target} continues for more iterations than the evolution of D_{smooth}. Doing so allows the warp computed in Step 4 to expand the layout to unused portions of the screen.

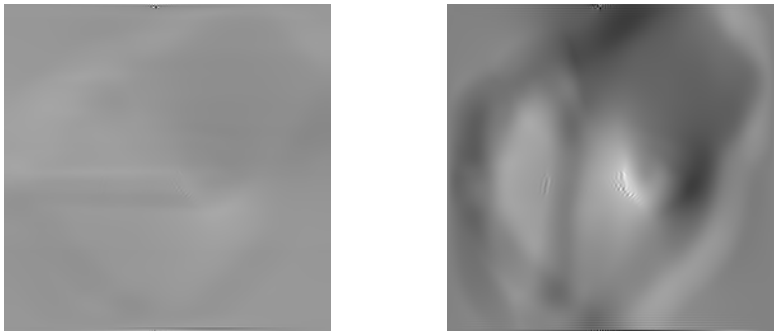
Computing an optimal warp (Step 4): After computing D_{smooth} and D_{target} in the previous steps, we are now ready to compute a warp $\tilde{u} = (u_1(x,y), u_2(x,y))$ that maps location (x,y)



(a) Input layout $L_{initial}$ for the 3elt graph, V=4720 E=13722 (b) Output graph L_{final}



(c) Initial density $D_{initial}$ (Step 1) (d) Smoothed density D_{smooth} (Step 2) (e) Target density D_{target} (Step 3)



(f) x-component of the warp $\tilde{u}$ (Step 4) (g) y-component of the warp $\tilde{u}$ (Step 4)

Fig. 3. Algorithm steps. Higher intensity represents higher values. Values are scaled to improve contrast.

in D_{smooth} to location $(u_1(x,y),u_2(x,y))$ in D_{target}. Using $\tilde{u}$, we are able to modify the layout, as discussed in Step 5, in order to compute L_{final}.

The warp procedure is based on the algorithm of Haker et al. [9], which is shown to compute a warp that minimizes displacements. In our case this helps maintain the overall structure of the graph, thus helping preserve the mental map . The key idea of the algorithm is to iteratively converge to an optimal mapping by using a gradient descent technique. More details are given in Section IV.

Computing the final layout (Step 5): In the final stage of the algorithm, the positions of the nodes are modified in order to create the output layout L_{final}. Given the optimal warping $\tilde{u} = (u_1(x,y),u_2(x,y))$ that was computed in Step 4, which is defined over a discreet, regular grid, this step computes the updated positions of each node in the graph, which are non-integral. Note that this stage modifies the node coordinates and not the image of the layout.

The optimal warping $\tilde{u} = (u_1(x,y),u_2(x,y))$ gives for each pixel in the input density a destination position in the image. Using the warp, new node positions are computed using an iterative process. Given a node n with current position (x_n,y_n) (initialized to the node position in $L_{initial}$), its updated position is set to

$$\begin{aligned} x_n^{updated} &= x_n + \alpha(u_1(x_n,y_n) - x_n) \\ y_n^{updated} &= y_n + \alpha(u_2(x_n,y_n) - y_n). \end{aligned} \tag{7}$$

The number of repetitions of Equation 7 is controlled by the user. Performing more iterations results in a larger displacement, representing a tradeoff between node separation and preserving the structure of the graph. The constant α, whose default value is 0.5 is used to scale the displacement.

In order to compute the value of the functions u_1 and u_2 at the non-integral node coordinates (x_n,y_n) bilinear interpolation is used. Using an interpolation method with sub-pixel accuracy

helps increase the separation between close-by nodes in the input layout.

Complexity: Step 1 requires traversing the nodes and edges of the graph, which is $O(E+V)$ for a graph with E edges and V nodes. In addition it requires rasterizing the nodes and edges, which is performed quickly on the GPU. Step 2 performs a fixed number of iterations, each of which takes $O(P)$ for an image containing P pixels. Step 3 uses a fixed number of directions, each requiring $O(\sqrt{P})$ work for summing up the densities along the ray emanating from each of the P pixels. The total here is $O(P^{1.5})$. As discussed in Section IV, Step 4 requires $O(P)$. Finally, the last step is $O(V)$. Hence, the total runtime is $O(E+V+P^{1.5})$. As shown in Section VI, it is dominated by the time spent in Step 3, which can be controlled by changing P.

IV. Computing an Optimal Mapping

In this section we describe a method, based on optimal mass transport, for finding a mapping between the two density images D_{smooth} and D_{target} in a way that minimizes displacements, thus preserving the structure of the graph.

First, a brief introduction to the optimal mass transport problem, which was first formulated by Monge in 1781 and later by Kantorovich [8] is provided. Next, the application of this problem to improving graph layouts is discussed. The section concludes by briefly describing how the mass-transport problem is efficiently solved using the algorithm of Haker et al. [9].

Let Ω_0 and Ω_1 be two subdomains of R^2, with smooth boundaries. Positive density functions $\mu_0(x,y)$ and $\mu_1(x,y)$ are defined on these domains, respectively. We assume that

$$\iint_{\Omega_0} \mu_0(x,y)\,dx\,dy = \iint_{\Omega_1} \mu_1(x,y)\,dx\,dy, \tag{8}$$

i.e. the same total mass is contained in both regions. In our case of density images of graph layouts, we assume $\Omega_0 = \Omega_1 = [0,1]\times[0,1]$.

Our purpose is to construct a mapping between D_{smooth} computed in Step 2 and D_{target} computed in Step 3. Unlike the classical setting described above, the densities used in our case can be zero in some regions of the image - the ones not occupied by the input graph layout. We therefore equalize the mass (in order to ensure Equation 8 is met) and add a constant ε to each of the input densities before computing the optimal mapping $\tilde{u}$, using the following relations:

$$\mu_0 = \varepsilon + D_{smooth} \quad , \quad \mu_1 = \varepsilon + D_{target}\frac{\iint_{\Omega_0} D_{smooth}(x,y)\,dxdy}{\iint_{\Omega_1} D_{target}(x,y)\,dxdy}, \tag{9}$$

where μ_0, μ_1 are the equalized and shifted densities which are used to compute the optimal warp. In our implementation $\varepsilon = 0.5$.

Diffeomorphisms $\tilde{u} = (u_1(x,y), u_2(x,y))$ from Ω_0 to Ω_1, which map one density function to the other according to the following relation

$$\mu_0(x,y) = |D\tilde{u}(x,y)|\mu_1(\tilde{u}(x,y)) \tag{10}$$

are considered. Here $D\tilde{u}$ is the Jacobian matrix and $|D\tilde{u}|$ is its determinant [29] . Equation 10 is called the *Mass Preservation* (MP) property and accordingly $\tilde{u} \in MP$. It implies, for example, that if a small region in Ω_0 is mapped to a large region in Ω_1, there must be a corresponding decrease in density in order for the mass to be preserved.

Many mappings $\tilde{u}$ that satisfy Equation 10 exist. We would like to choose an optimal one for our application. We use the squared L^2 Monge-Kantorovich distance, defined as follows

$$d_2^2(\mu_0,\mu_1) = \inf_{\tilde{u}\in MP} \iint \|\tilde{u}(x,y) - (x,y)\|^2 \mu_0(x,y)\,dx\,dy. \tag{11}$$

This distance places a penalty on the distance the map $\tilde{u}$ moves each bit of material, weighted by its mass. Hence, this distance fits our requirement of disturbing the input graph layout as little as possible, in order to reduce changes to the structure of the layout, thus conserving the user's mental map.

A fundamental theoretical result [10], [32], [33] states that there exists a unique optimal mapping $\tilde{u}$ that is a gradient of a convex function ω, i.e. $\tilde{u} = \nabla\omega$. In order to find the optimal mapping $\tilde{u}$ we use the algorithm of Haker et al. [9]. This algorithm has two main stages. First, an initial mapping u^0 is found. Next, the mapping is updated iteratively in order to decrease the functional in Equation 11.

Finding an initial mapping is achieved by first solving a one-dimensional problem of transporting mass in a direction parallel to the x-axis (Equation 12), followed by the solution of a series of problems transporting mass parallel to the y-axis (Equation 13). A function $a = a(x)$ is implicitly defined by the equation

$$\int_0^{a(x)}\int_0^1 \mu_1(\eta,y)\,dy\,d\eta = \int_0^x\int_0^1 \mu_0(\eta,y)\,dy\,d\eta. \tag{12}$$

$a(x)$ is determined by numerically calculating the integrals. Differentiating Equation 12 with respect to x gives

$$a'(x)\int_0^1 \mu_1(a(x),y)\,dy = \int_0^1 \mu_0(x,y)\,dy.$$

A function $b = b(x,y)$ is now defined implicitly by the equation

$$a'(x)\int_0^{b(x,y)} \mu_1(a(x),\rho)\,d\rho = \int_0^y \mu_0(x,\rho)\,d\rho. \tag{13}$$

Given $a(x)$, the function $b(x,y)$ can be computed by numerically performing the integrations in Equation 13. The initial mapping is set to be $u^0(x,y) = (a(x), b(x,y))$.

Considering u^0 to be a vector field, the Helmholtz-Hodge decomposition [29] states that u^0 can be decomposed into the sum of a curl-free vector field $\nabla\omega$ and a divergence free vector field χ, i.e. $u^0 = \nabla\omega + \chi$. In the 2D case a divergence free vector field χ can be written as $\chi = \nabla^\perp h$ for some scalar function h, were $\perp$ represents rotation by 90°, so $\nabla^\perp h = (-\frac{\partial h}{\partial y}, \frac{\partial h}{\partial x})$. In this case the decomposition is $u^0 = \nabla\omega + \nabla^\perp h$.

In order to compute the optimal MP mapping $\tilde{u} = \nabla\omega$, the second step of the algorithm removes the curl from u^0. This is achieved by using an iterative gradient descent method. In each iteration the current mapping u is modified in order to reduce the functional in Equation 11. Note that at all stages, the mapping u is a valid solution to the mass-transport problem. Setting $u = \nabla\omega + \nabla^\perp f$, f is found by solving the following Poisson Equation with a Dirichlet-type boundary condition:

$$\begin{aligned} \nabla^2 f &= -div(u^\perp) \\ f &= 0\, on\, \partial\Omega_0. \end{aligned} \tag{14}$$

The boundary condition ensures that the mapping will remain constrained in the given domain. It is shown in [9] that the functional in Equation 11 can be reduced by the following

evolution equation:

$$\frac{\partial u}{\partial t} = \frac{1}{\mu_0} Du \nabla^{\perp} f. \quad (15)$$

The time step $\triangle t$ is set as $\triangle t = \min_{x,i} \| \frac{1}{\mu_0} (\nabla^{\perp} f)_i \|^{-1}$, where the subscript i stands for the component of the vector. The algorithm iteratively solves the Poisson Equation and updates the mapping u until the curl of u is below a given threshold. Our experiments show that performing up to 30 iterations of Equation 15 is sufficient for obtaining a high-quality warp.

A multi–grid method [27], [30] is used in order to quickly solve Equation 14. The implementation uses the V-cycle algorithm to control the transition between grid levels, Jacobi iterations for smoothing the solution and full weighting for downsampling solutions between grids [27]. The complexity of the multi-grid method for an image containing P pixels is $O(P)$, resulting in a rapid solution. Equations 12,13,15 are linear in the image size. A fixed number of iterations of Equation 15 is performed. Hence, The total complexity of this step in the algorithm is $O(P)$.

V. Implementation on the GPU

Computing the target density (Step 3) is the most time consuming stage of the algorithm since we need to perform many computations for each pixel of the image. In this section we describe how this step is implemented on the GPU, resulting in a significant speedup of the running time of the algorithm, as shown in Section VI.

The GPU has several architectural characteristics that help improve the speed of computation compared to the CPU. First, the GPU is highly parallel. It is able to run hundreds of computational threads in parallel. In some cases, memory access latency is hidden by switching to executing a different thread. Second, the GPUs memory system is optimized for two-dimensional locality, as opposed to the one-dimensional locality employed in CPUs. Our implementation on the GPU takes advantage of these properties.

Given the current density image μ as an input, the goal is to calculate for each pixel the best advancement direction, θ_{best}, as in Equation 5. This is done by finding for each pixel the angle that minimizes the score in Equation 4.

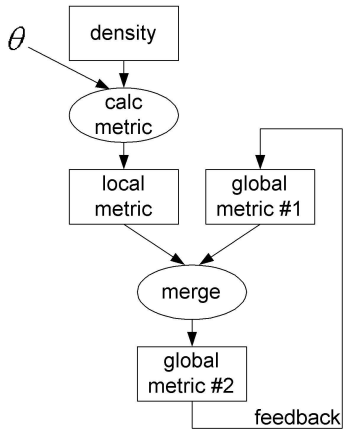


Fig. 4. Execution graph of finding the best advancement direction on the GPU in Step 3 (rectangles = textures, ovals=kernels, θ is the current direction being tested)

Several *textures*, which are two-dimensional images or data arrays, are used to store data on the GPU, as illustrated in Figure 4. The input density μ is stored in the *density* texture. For each candidate direction θ, the current score for each pixel is stored in the *local metric* texture. Two textures are used to store the current best angle for each pixel: *global metric #1* and *global metric #2*. We use two textures due to the GPU's inability to read and write to the same texture. At the end of the computation the global metric texture holds the best advancement direction θ_{best} for each pixel.

Computation on the GPU is achieved by running a *kernel* or *fragment program* for each pixel in the image. The GPU is able to split the computation into hundreds of parallel threads, thus achieving high performance. The computation, shown in Figure 4, is performed using two kernels. The first kernel, called *calc metric*, calculates Equation 4 for each pixel in the image given the current direction θ. Given the coordinates of the current pixel, l_{max} from Equation 4 is determined by calculating the closest intersection of the ray in direction θ, starting at the current pixel, with a boundary of the image. Next, the score is accumulated using Equation 4. During this process, Bilinear interpolation is used to access the density texture in the non-integral coordinates.

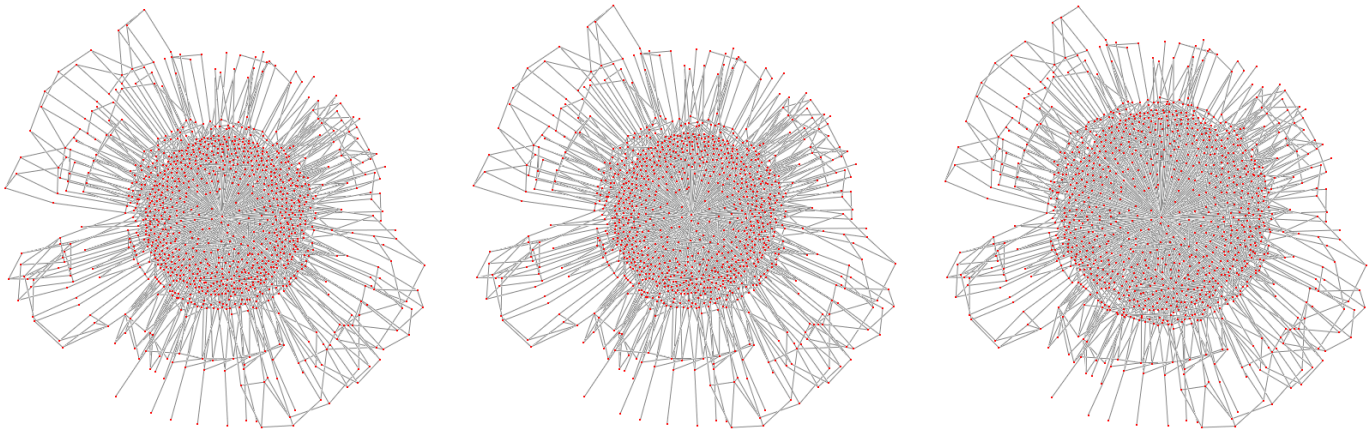
The GPU is able to efficiently execute the calc metric kernel. For each direction θ, when concurrently running the kernel on neighboring pixels, the accesses to the density metric have a 2D locality. This results in a good utilization of the caches and memory bandwidth of the GPU, which are optimized for 2D operations.

A second kernel, the *merge* kernel is used to update the current best advancement direction per pixel. This kernel accepts as input the previous best direction, stored in the global metric texture, and the value of the score calculated in the current direction θ, stored in the local metric texture. The kernel compares the two scores and writes to its output the merged best score. After iteratively running the calc metric and merge kernel for the set of all candidate angles, the global metric texture contains the value of the best angle θ_{best} for each pixel.

It should be noted that it is better to compute the best direction (Equation 5) in a single pass, using the current density μ as the input and the best direction $\theta_{best}(x,y)$ as the output. This would remove the necessity for having a temporary texture for the local result, performing the ping-pong algorithm between the two copies of the global metric texture and running the merge kernel. However, in order to protect the system from fatal errors, the graphics driver limits the amount of time a computational kernel is allowed to run. The allotted time is insufficient to perform the computation in one pass, especially in lower performance GPUs. Thus, we chose the multi-pass implementation discussed in the previous paragraphs.

VI. Results

Our algorithm was tested using the output of several state-of-the-art graph layout algorithms in a variety of applications. Table II gives information about the graphs and the parameters used in our algorithm. Below, we discuss the results of our algorithm and compare them to the results obtained by the node overlap algorithm of Dwyer et al. [20]. We were not able to compare the results of our algorithm to those of Gansner and North [15] since their algorithm is not designed to handle the large graphs used in our experiments.



(a) Input FM^3 layout (b) Removing node overlaps using [20] (c) Our improved layout

Fig. 5. ug_380 graph (V=1104, E=3231). Note how when using our algorithm the center expands, reducing node density while the outer ring is unchanged. When using [20] the layout is hardly changed.

Figures 1 and 5 show improvements of layouts computed by FM^3 [12], which is a multi-level force-directed algorithm. It uses solar systems, which consist of nodes at a distance of two edges or less from the center of the solar system, in order to create the graph hierarchy.

Figure 1 shows a layout of the protein graph, which is the unweighted version of the protein homology graph presented in [34]. The layout contains a large, dense central cluster. Applying our algorithm increases the percentage of screen space devoted to the elements of the graph. This allows more of the fine details of the graph to become visible, especially in the central region of the graph. Note how the overall structure of the different elements of the graph, such as the different ”spokes” it contains, is retained. In comparison, the algorithm from [20] was not able to remove all of the overlaps and the changes to the layout were small, similarly to Figure 8 (b).

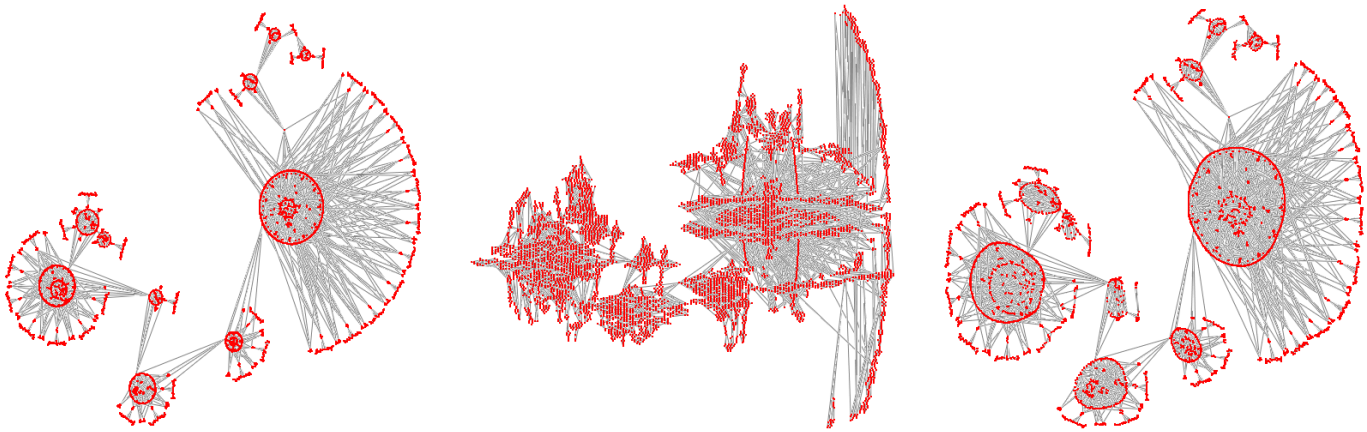
Figure 5 shows a layout of the ug_380 graph [35], which contains one node with a very high degree. The layout contains a central core which is packed with many nodes. In (b) the results of a node overlap removal algorithm [20] are shown. Since the input layout contains hardly any overlaps, the result in (b) is very similar to (a) and the graph remains cluttered. Applying our algorithm to this challenging case, shown in (c), results in an increase in the radius of the central core, increasing the separation between the nodes. The exterior nodes, which are sparser, are unaffected.

Figure 6 shows an improvement of the layout produced by TopoLayout, which is a feature-based multi-level graph drawing algorithm [36]. It creates a subgraph hierarchy by recursively detecting topological features in the graph and replacing them with meta-nodes. Each feature is drawn using an algorithm tuned for the specific topology. The graph hierarchy is drawn bottom-up using an area-aware algorithm. The figure shows the add32 graph [22], which describes a 32-bit adder that contains many biconnected components. In (b) the results of a node overlap removal algorithm [20] are shown. Note that the structure of the input layout is significantly distorted, making it difficult to comprehend the structure of the graph. Our improved layout, shown in (c), is able to expand the circular clusters contained in the graph, better visualizing the intricate details of the graph. For example, additional details about the composition of the inner circle in the leftmost part of the graph become visible. Also, expanding the small circular formation at the bottom right hand side of the graph allows more detail about the sub-clusters it contains to become visible. Moreover, as opposed to (b), the layout in (c) maintains the overall structure of the layout.

Figures 7 and 8 show improvements of the layouts produced by [37], which is a multi-level forced directed graph layout algorithm. Spectral partitioning is used to create the graph hierarchy. KD-tree type partitioning is used to accelerate the computation and allows for an efficient GPU implementation.

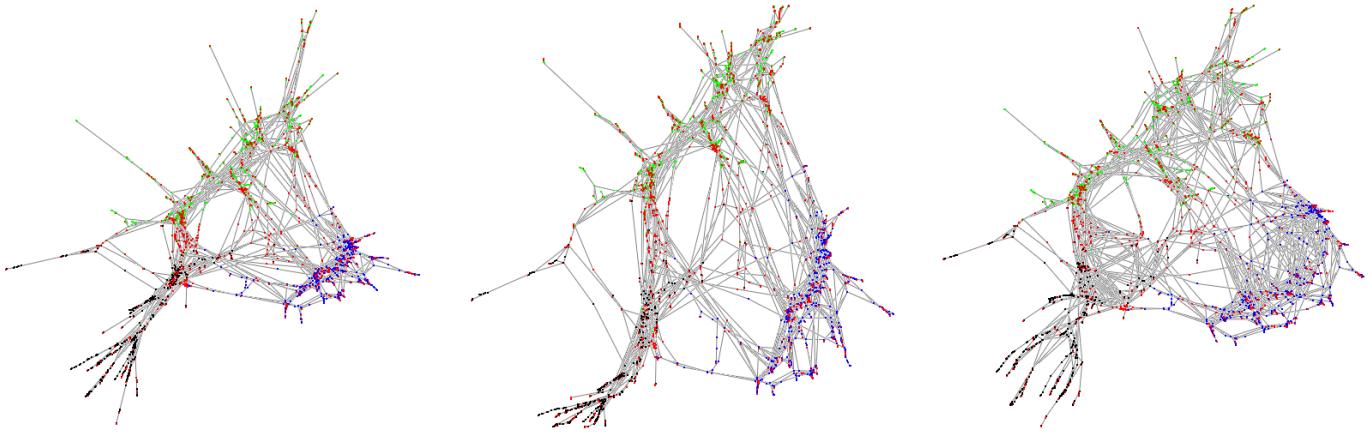
Figure 7 shows the ISP graph, which represents the router networks of several internet service providers (ISPs) [38]. In the layout, green, black and blue nodes represent routers belonging to the ISPs visualized, while red nodes show other routers used to connect to the Internet. The layout in (b), computed by the algorithm from [20], manages to displace nodes in order to avoid overlaps, while generally maintaining the overall structure of the graph. Unlike our algorithm, the resulting layout does not attempt to make use of sparse regions of the layout. Instead, small displacements are used in order to avoid overlaps. Applying our algorithm to this layout, as shown in (c), improves the separation between the nodes of the graph, while maintaining important characteristics of the graph, such as the separation to clusters (excluding the red nodes). This is especially evident in the blue cluster at the bottom right and among the red nodes in the center left part of the graph. Note how the algorithm is able to expand each of the clusters into surrounding sparse areas, allowing more details to become visible inside the clusters, while still preserving the overall clustered structure of the graph.

Figure 8 shows the bcsstk32 graph [22], which represents a stiffness matrix. It has a very high edge density: $E/V > 22$. The layout in (b), computed by the algorithm from [20], is nearly identical to the input layout. The algorithm is not able to remove all of the overlaps of the graph, even when we change the size of the squares representing the nodes. In (c) our uncluttering algorithm is used. It stretches the input layout, making the mesh-like structure of the graph more evident. Note that the overall structure and features of the graph are conserved after the uncluttering process. Also note that in the improved layout there are less highly-concentrated areas, where the edges are totally hidden. This makes the mesh structure of the graph visible in a



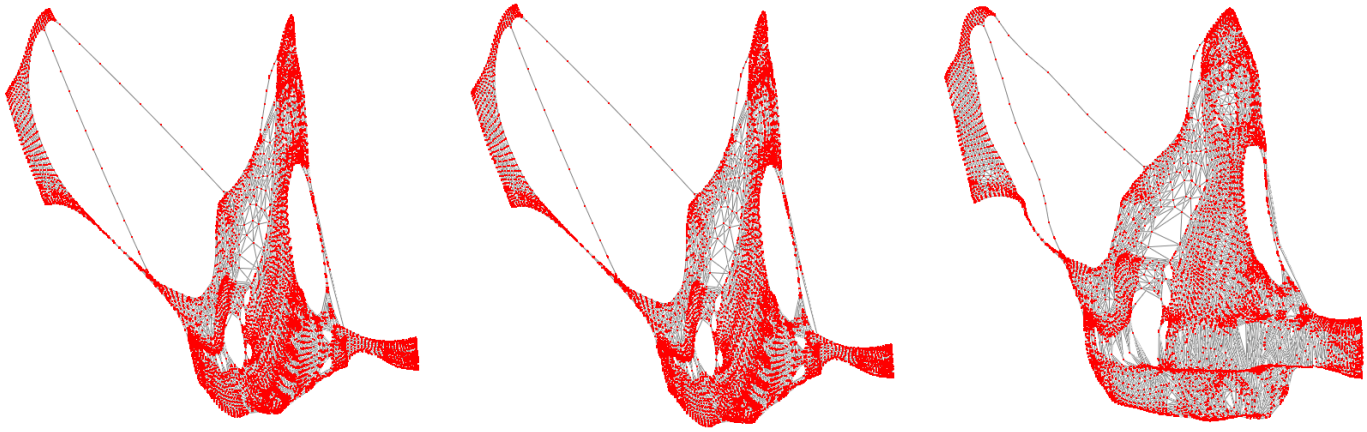
(a) Input layout by TopoLayout [36] (b) Removing node overlaps using [20] (c) Our improved layout

Fig. 6. Add32 graph (V=4960, E=9462). Note how in (c) each of the rings is expanded, showing more detail.



(a) Input layout from [37] (b) Removing node overlaps using [20] (c) Our improved layout

Fig. 7. ISP router graph (V=5044, E=8043) . Nodes are color-coded by the ISP they belong to. Note how in (c) the blue nodes are uncluttered.



(a) Input layout from [37] (b) Removing node overlaps using [20] (c) Our improved layout

Fig. 8. Bcsstk32 graph (V=44609, E=985046). Note how in (c) reducing the node density allows more of the mesh structure of the graph to be uncovered in the top left, bottom and middle of the graph.

larger portion of the layout.

In order to quantitatively measure the quality of the improvement to the layout, we define the following metric:

$$Q = \frac{1}{V(V-1)} A_G \sum_{u,v \in V, u \neq v} \frac{1}{dist(u,v)}, \quad (16)$$

where V is the number of nodes in the graph, A_G is the area of the layout, and $dist(u,v)$ is the distance between nodes u and v. The metric sums the inverted distances between all node pairs, multiplied by a normalization factor. Under this metric, a lower score is better since it implies that nodes are further away (and hence $\frac{1}{dist(u,v)}$ is smaller). In order to normalize the metric we

graph name	input	node overlap removal [20]	our algorithm
add32	10763	15535	9392
bcsstk32	1465.9	1301.2	623.56
ISP	251.41	511.36	198.37
ug_380	7811.8	7811.8	6751.5

TABLE I

QUALITY RESULTS. VALUES IN THE TABLE ARE CALCULATED ACCORDING TO THE QUALITY METRIC IN EQUATION 16. LOWER VALUES IMPLY A HIGHER QUALITY LAYOUT. THE INPUT COLUMN REPRESENTS THE QUALITY METRIC FOR THE INPUT GRAPH. THE NODE OVERLAP REMOVAL COLUMN GIVES RESULTS WHEN APPLYING THE ALGORITHM FROM [20]. THE RESULTS WHEN USING OUR ALGORITHM ARE SHOWN IN THE RIGHT–MOST COLUMN.

graph information			node overlap removal [20]	our algorithm			
graph	V	E	CPU	$\sqrt{P}$	ITRS	CPU	CPU+GPU
protein	30727	1206654	543	257	8	643	6.62
add32	4960	9462	2.23	257	15	641	4.86
bcsstk32	44609	985046	462	257	4	642	5.84
ISP	5044	8043	0.9	257	25	643	5.19
ug_380	1104	3231	0.03	257	30	643	4.86

TABLE II

GRAPH INFORMATION AND RUNNING TIMES. THE LEFT SIDE OF THE TABLE GIVES INFORMATION ABOUT THE GRAPHS. V AND E ARE THE NUMBER OF GRAPH NODES AND EDGES, RESPECTIVELY. THE CENTRAL PART OF THE TABLE GIVES THE RUNNING TIMES IN SECONDS OF THE ALGORITHM FROM [20], USING THE SAME MACHINE USED TO RUN OUR ALGORITHM. THE RIGHT SIDE OF THE TABLE SHOWS THE RESULTS OF OUR ALGORITHM. THE WIDTH AND HEIGHT IN PIXELS OF THE DENSITY IMAGE USED IS EQUAL TO $\sqrt{P}$. ITRS IS THE NUMBER OF ITERATIONS OF EQUATION 7 IN STEP 5. CPU IS THE TOTAL RUNNING TIME OF THE ALGORITHM IN SECONDS WHEN USING ONLY THE CPU. CPU+GPU IS THE TOTAL RUNNING TIME OF THE ALGORITHM IN SECONDS WHEN USING THE GPU TO ACCELERATE STEP 3.

divide by the number of node-node combinations ($V*(V-1)$). The metric is multiplied by the area of the layout in order to take into account changes to the size of the layout. A larger layout will receive a higher (i.e. worse) metric since enlarging the layout will naturally lead to improved distances between the nodes.

Table I shows the results of using the quality metric defined in Equation 16 on the graphs in Figures 5 - 8. The table shows the metric for the input graph as well as the results when using the node overlap removal algorithm from [20] and when using our algorithm. As can be seen, using our algorithm improves the metric (corresponding to a lower value) consistently. This demonstrates that indeed our algorithm manages to increase the separation between nearby nodes, thus improving the readability of the layouts and showing finer details. When using the algorithm from [20], the quality is improved in some cases (such as the bcsstk32 graph) while it is reduced in others. For example, in the ISP graph, the graph is stretched in the y axis (this is not visible Figure 7 (b), which is square), thus reducing the quality metric of the computed layout. A similar case, contributing to a less desirable high score, happened with the add32 graph, whose area was significantly enlarged in order to avoid node overlaps.

For our performance tests, we used a PC running Windows XP equipped with 2GB RAM, an Intel Core 2 Duo E6750 2.66 GHz CPU and an NVIDIA 8800GTS GPU with 96 shader processors running at 1.2GHz. The algorithm was implemented using C++, OpenGL and Cg.

Table II gives information about the graphs and the running times. It is evident that the running time is relatively independent of the size of the graph and the number of displacement iterations made. This is so since the bulk of the computation time is spent working on the different images the algorithm operates on. More specifically, as can be seen from comparing the CPU and CPU+GPU columns, most of the time is spent in Step 3, which involves a computationally demanding ray-shooting process (Equations 4 and 5). Using the GPU results in a very large speedup of this step, accelerating the total runtime by up to 130 times. This reduces the total runtime to a few seconds.

Table II compares our running times to those of [20]. In the latter, there is a big variation in the running time, since it depends on the number of overlaps. When there are few overlaps (add32, ISP, ug_380), the algorithm runs quickly. Consequently, the changes to the layout are small. In other cases (protein, bcsstk32), the running time is higher. Due to the large variation in running times, in some cases it runs faster than our GPU implementation while in others it runs slower.

There are several reasons why the GPU is able to accelerate Step 3 and therefore the execution of the entire algorithm so significantly. First, since the amount of work per-pixel is similar, there is good load balance between the different processors in the GPU. Thus, the GPU is able to make efficient use of its computing power, which is much higher than the CPU's. Second, due to the 2D locality in the memory access pattern during the ray-shooting process, the GPU is able to make efficient use of its caches. On the CPU, however, accessing a 2D image requires lookups using pointers, which is inefficient. Finally, as opposed to the CPU, the GPU contains built-in instructions for performing the clamping operations needed for performing the interpolation of the values in the density texture. In summary, this is a good example in which the architecture of the GPU is able to provide a significant speedup compared to a CPU implementation.

VII. CONCLUSION

This paper proposes a new algorithm for reducing the cluttering commonly occurring in graph layouts. Given any graph layout,

the algorithm moves nodes to empty regions of the screen while attempting to retain the overall structure of the graph and thus reduce disturbances to the user's metal map.

The algorithm has several key ideas. First, the density image of the computed graph layout is used to decide how nodes will be displaced. Second, a diffusion process that takes the structure of the density image into account computes an alternative node distribution, making better use of the available screen space. Third, an optimal and mental-map preserving warp, based on results from mass-transport problems, determines how to displace the nodes. Although the mathematical techniques used in this paper require a great deal of computation, the paper demonstrates how improved layouts can be computed in a matter of seconds, by using the GPU to significantly accelerate the algorithm.

It has been shown that our algorithm is able to improve layouts of large graphs, produced by a variety of well-known algorithms.

In future research we plan to integrate the algorithm into an interactive system that allows presenting user-selected regions of interest in the graph with more detail. Another research direction is using the algorithm to enhance the visualization of changes in a dynamic graph sequence.

It may be possible to accelerate the algorithm further by moving more parts to the GPU. These include the multi-grid solution of the Poisson equation [39] (Equation 14) and the iterative mass-transport evolution [40] (Equation 15).

Acknowledgments

We would like to thank Daniel Archambault for supplying some of the graph layouts shown. This work was partially supported by the Israeli Ministry of Science, Culture & Sports, grant 3-3421 and by the Technion V.P.R. Fund. We would like to thank the anonymous reviewers for their constructive comments.

References

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

[2] M. Kaufmann and D. Wagner, Eds., *Drawing Graphs: Methods and Models*, ser. LNCS. Springer-Verlag, 2001, no. 2025.

[3] K. A. Lyons, H. Meijer, and D. Rappaport, "Algorithms for cluster busting in anchored graph drawing," *J. Graph Algorithms and Applications*, vol. 2, no. 1, pp. 1–24, 1998.

[4] P. Eades, "A heuristic for graph drawing," *Congressus Numerantium*, vol. 42, pp. 149–160, 1984.

[5] T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," *Information Processing Letters*, vol. 31, no. 1, pp. 7–15, 1989.

[6] T. M. J. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software—Practice and Experience*, vol. 21, no. 11, pp. 1129–1164, 1991.

[7] D. Merrick and J. Gudmundsson, "Increasing the readability of graph drawings with centrality-based scaling," in *Asia Pacific Symposium on Information Visualisation (APVIS2006)*, ser. CRPIT, vol. 60, 2006, pp. 67–76.

[8] L. V. Kantorovich, "On a problem of Monge," *Uspekhi Mat. Nauk.*, vol. 3, no. 2, pp. 225–226, 1948.

[9] S. Haker, L. Zhu, A. Tannenbaum, and S. Angenent, "Optimal mass transport for registration and warping," *International Journal of Computer Vision*, vol. 60, no. 3, pp. 225–240, 2004.

[10] J.-D. Benamou and Y. Brenier, "A computational fluid mechanics solution to the Monge–Kantorovich mass transfer problem," *Numerische Mathematik*, vol. 84, no. 3, pp. 375–393, Oct. 2000.

[11] K. Misue, P. Eades, W. Lai, and K. Sugiyama, "Layout adjustment and the mental map," *Journal of Visual Languages and Computing*, vol. 6, no. 2, pp. 183–210, 1995.

[12] S. Hachul and M. Jünger, "Drawing large graphs with a potential-field-based multilevel algorithm." in *Proc. 12th Int. Symp. on Graph Drawing*, ser. LNCS, no. 3383, 2004, pp. 285–295.

[13] E. Shimizu and R. Inoue, "Time-distance mapping: visualization of transportation level of service," in *Proc. of Symposium on Environmental Issues Related to Infrastructure Development*, 2003, pp. 221–230.

[14] J. H. Chuang, C. C. Lin, and H. C. Yen, "Drawing graphs with nonuniform nodes using potential fields," in *Proc. 11th Int. Symp. Graph Drawing (GD 2003)*, ser. LNCS, no. 2912, 2004, pp. 460–465.

[15] E. R. Gansner and S. C. North, "Improved force-directed layouts," in *Graph Drawing*, ser. LNCS, vol. 1547, 1998, pp. 364–373.

[16] D. Harel and Y. Koren, "Drawing graphs with non-uniform vertices," in *Proc. Working Conference on Advanced Visual Interfaces (AVI'02)*. ACM Press, 2002, pp. 157–166.

[17] K. Marriott, P. J. Stuckey, V. Tam, and W. He, "Removing node overlapping in graph layout using constrained optimization," *Constraints*, vol. 8, no. 2, pp. 143–171, 2003.

[18] W. Li, P. Eades, and N. Nikolov, "Using spring algorithms to remove node overlapping," in *Asia Pacific Symposium on Information Visualisation (APVIS2005)*, ser. CRPIT, vol. 45, 2005, pp. 131–140.

[19] X. Huang, W. Lai, A. S. M. Sajeev, and J. Gao, "A new algorithm for removing node overlapping in graph visualization," *Inf. Sci.*, vol. 177, no. 14, pp. 2821–2844, 2007.

[20] T. Dwyer, K. Marriott, and P. J. Stuckey, "Fast node overlap removal," in *Graph Drawing*, ser. Lecture Notes in Computer Science, vol. 3843. Springer, 2005, pp. 153–164.

[21] E. R. Gansner and S. C. North, "An open graph visualization system and its applications to software engineering," *Software — Practice and Experience*, vol. 30, no. 11, pp. 1203–1234, 2000.

[22] C. Walshaw, "graph collection," `http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/`.

[23] O. Deussen, S. Hiller, C. van Overveld, and T. Strothotte, "Floating points: A method for computing stipple drawings," *Computer Graphics Forum*, vol. 19, no. 3, Aug. 2000, ISSN 1067-7055.

[24] T. F. Chan, J. Cong, and K. Sze, "Multilevel generalized force-directed method for circuit placement," in *ISPD*, P. Groeneveld and L. Scheffer, Eds. ACM, 2005, pp. 185–192.

[25] K. Hayashi, M. Inoue, T. Masuzawa, and H. Fujiwara, "A layout adjustment problem for disjoint rectangles preserving orthogonal order," in *Graph Drawing*, ser. Lecture Notes in Computer Science, S. Whitesides, Ed., vol. 1547. Springer, 1998, pp. 183–197.

[26] M. T. Gastner and M. E. J. Newman, "Diffusion-based method for producing density-equalizing maps," *Proc. Nat. Acad. Sci. USA*, vol. 101, no. 20, pp. 7499–7504, 2004.

[27] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A multigrid tutorial: second edition*. SIAM, 2000.

[28] R. van Liere and W. de Leeuw, "GraphSplatting: Visualizing graphs as continuous fields," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 2, pp. 206–212, 2003.

[29] G. Strang, *Introduction to Applied Mathematics*. Wellesley-Cambridge press, 1986.

[30] J. W. Demmel, *Applied Numerical Linear Algebra*. SIAM, 1997.

[31] S. C. Chapra and R. P. Canale, *Numerical methods for engineers: with programming and software applications, 3rd edition*. McGraw Hill, 1998.

[32] Y. Brenier, "Polar factorization and monotone rearrangement of vector-valued functions," *Communications on pure and applied mathematics*, vol. 44, no. 4, pp. 375–417, 1991.

[33] M. Knott and C. S. Smith, "On the optimal mapping of distributions," *Journal of Optimization Theory and Applications*, vol. 43, no. 1, pp. 39–49, 1984.

[34] A. T. Adai, S. V. Date, S. Wieland, and E. M. Marcotte, "LGL: creating a map of protein function with an algorithm for visualizing very large biological networks," *J. Molecular Biology*, pp. 179–190, 2004.

[35] "AT&T graph library," linked from http://www.graphdrawing.org/.

[36] D. Archambault, T. Munzner, and D. Auber, "TopoLayout: Multilevel graph layout by topological features," *IEEE Trans. on Visualization and Computer Graphics*, vol. 13, no. 2, pp. 305–317, 2007.

[37] Y. Frishman and A. Tal, "Multi-level graph layout on the GPU," *IEEE Trans. on Visualization and Computer Graphics (Proc. InfoVis)*, vol. 13, no. 6, pp. 1310–1317, 2007.

[38] "Rocketfuel maps and data," `http://www.cs.washington.edu/research/networking/rocketfuel/` .

[39] N. Goodnight, C. Woolley, G. Lewin, D. Luebke, and G. Humphreys, "A multigrid solver for boundary value problems using programmable graphics hardware," in *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, 2003, pp. 102–111.

[40] T. Rehman, E. Haber, G. Pryor, J. Melonakos, and A. Tannenbaum, "3D nonrigid registration via optimal mass transport on the GPU," *Accepted - Elsevier Journal of Medical Image Analysis*, 2008.

Yaniv Frishman received the BSc degree (summa cum laude) in computer engineering and the masters degree from the Technion, Israel Institute of Technology, Haifa, Israel, in 1997 and 2006, respectively. He completed his PhD in the department of Computer Science of the Technion, Israel Institute of Technology in 2009. His research interests include information visualization, graph drawing, general-purpose computation on graphics processing units, and computer graphics.

Ayellet Tal received the BSc degree (summa cum laude) in mathematics and computer science and the MSc degree (summa cum laude) in computer science from Tel-Aviv University and the PhD degree in computer science from Princeton University. She is an associate professor in the Department of Electrical Engineering, Technion, Israel Institute of Technology, Haifa, Israel. Her research interests include computer graphics, information and scientific visualization, computational geometry, and multimedia.