ACE: A Fast Multiscale Eigenvectors Computation for Drawing Huge Graphs *

Yehuda Koren Liran Carmel David Harel
Dept. of Computer Science and Applied Mathematics
The Weizmann Institute of Science, Rehovot, Israel
{yehuda,liran,harel}@wisdom.weizmann.ac.il

Abstract

We present an extremely fast graph drawing algorithm for very large graphs, which we term ACE (for Algebraic multigrid Computation of Eigenvectors). ACE exhibits an improvement of something like two orders of magnitude over the fastest algorithms we are aware of; it draws graphs of millions of nodes in less than a minute. ACE finds an optimal drawing by minimizing a quadratic energy function. The minimization problem is expressed as a generalized eigenvalue problem, which is rapidly solved using a novel algebraic multigrid technique. The same generalized eigenvalue problem seems to come up also in other fields, hence ACE appears to be applicable outside of graph drawing too.

Keywords: algebraic multigrid, multiscale/multilevel optimization, graph drawing, generalized eigenvalue problem, Fiedler vector, force directed layout, the Hall energy

1 Introduction

A graph is a structure $G(V, E)$, with $V = \{1, \ldots, n\}$ a set of n nodes, and E a set of weighted edges, w_{ij} being the weight of the edge connecting nodes i and j. Many datasets are given in the form of pairwise similarities between entities. Identifying these similarities with the weights, such data are naturally represented as graphs, and their visualization can be carried out using graph drawing techniques.

In a very general sense, we expect the drawing of a graph to visually capture its inherent structure. Interpreting this vague desire into strict well-defined criterion for the purpose of assessing the quality of a drawing can be done in various ways, leading to many approaches to graph drawing [5, 14]. One of the most popular approaches is to define an energy function (or a force model), whose minimization determines the optimal drawing. In this paper we concentrate on one particular form of an energy function, characterized by being simple and smooth, thus enabling rigorous analysis and a straightforward implementation. This particular function was first applied to graph drawing by Hall [9], and we therefore term it *Hall's energy*.

Most graph drawing methods suffer from lengthy computation times when applied to really large graphs. Several publications in the graph drawing conference of 2000 [10, 21, 6, 18] present fast graph drawing algorithms, but even the fastest of them ([21]) requires about 10 minutes for a typical 10^5-node graph. In fact, a naive implementation of the minimization of Hall's energy would also encounter real difficulties on a 10^5 node graph. Recently, we have been able to achieve computation times comparable to ACE [11], using a different approach to graph drawing.

In this paper we suggest an extremely fast algebraic multigrid (AMG) implementation for minimizing Hall's energy (see [3, 20] for information on AMG algorithms). It results in typical computation times of 10-20 seconds for 10^6-node graphs. Furthermore, the problem that we will be solving is of more fundamental nature, and our algorithm can be used in areas outside of graph drawing, such as clustering, partitioning, ordering, and image segmentation.

2 The Minimization Problem

A graph is uniquely described by the *Laplacian*, which will be proved to be a key feature of ACE:

Definition 2.1 (Laplacian) *Let $G(V, E)$ be a graph. The Laplacian of the graph is the symmetric $n \times n$ matrix*

$$L_{ij} = \begin{cases} \sum_{k=1}^{n} w_{ik} & i = j \\ -w_{ij} & i \neq j \end{cases} \qquad i, j = 1, \ldots, n.$$

It can be easily seen that for the commonly encountered graphs, in which all the weights are non-negative, $w_{ij} \geq 0$, the Laplacian is positive semi-definite. A key observation of our work is that nice drawings are feasible if the Laplacian is positive semi-definite, regardless of the sign of the weights. Hence, we define a more general entity, that we shall be calling a *PSD graph*:

Definition 2.2 (Positive Semi-Definite (PSD) Graph) *A PSD graph is a structure $G(V, E, \mathcal{M})$ in which:*

1. *$V = \{1, \ldots, n\}$ is a set of n nodes.*
2. *E is a set of weighted edges, such that the Laplacian of G is positive semi-definite.*

*This is an abridged version of the paper. For a full version see [15].

3. *$\mathcal{M}$ is a set of n strictly positive masses, m_i being the mass of the i'th node.*

We would like to dwell upon two notable features of a PSD graph. First, it contains node masses. Second, we permit negative edge weights, as long as the Laplacian remains positive semi-definite. Positive weights are interpreted as measuring the similarity between pairs of nodes, and negative ones measure dissimilarity — the larger the absolute value of $-w_{ij}$ the more dissimilar are nodes i and j. Consequently, in the drawing we would expect nodes connected by large positive weights to be close to each other, and those connected by large negative weights to be distantly located. For later use, let us arrange the masses in the diagonal *mass matrix* M, with $M_{ii} = m_i$. Since it is obvious that both the Laplacian L and the mass matrix M completely define a PSD graph, we will freely use both $G(V, E, \mathcal{M})$ and $G(L, M)$ to symbolize a PSD graph.

A reasonable approach to draw such a graph in one-dimension would be to solve the constrained minimization problem

$$\begin{aligned} &\min_x x^T L x \qquad (1)\\ &\text{given } x^T M x = 1\\ &\text{in the subspace } x^T M \cdot 1_n = 0, \end{aligned}$$

where $x = (x_1, \ldots, x_n)^T$ is the vector of node coordinates. Here, the energy function to be minimized is the Hall energy $E(x) \stackrel{def}{=} x^T L x = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(x_i - x_j)^2$, which was first introduced by Hall [9]. The positive semi-definiteness of the Laplacian assures that $E(x) \geq 0$ for any drawing and any graph, thus guaranteeing the existence of a global minimum to the problem. The constraint $x^T M x = 1$ poses an overall scaling to the drawing. For, if x_0 is a minimizer of (1) with energy $E_0 = x_0^T L x_0$, then $\sqrt{c}x_0$ (with energy cE_0) will be a minimizer of the same problem but with the constraint $x^T M x = c$. The constraint $x^T M \cdot 1_n = 0$, with 1_n being the n-vector $(1, \ldots, 1)^T$, limits us only to solutions that obey $\sum_i m_i x_i = 0$, thus avoiding the degenerate solution of putting all the nodes at the same location.

In the full version of the paper, [15], we show that problem (1) is closely related to the generalized eigenvalue problem of (L, M), $Lu = \mu M u$. Thus, it will be called hereinafter the *generalized eigenprojection problem*. L being positive semi-definite and M being positive definite leads to the generalized eigenvalues being non-negative, and to the existence of real M-orthonormal generalized eigenvectors. Throughout the paper we will use the convention $\mu_1 \leq \mu_2 \leq \ldots \leq \mu_n$ for the generalized eigenvalues of (L, M), and denote the corresponding real M-orthonormal eigenvectors by $u_1, u_2, \ldots, u_n$. The vector $u_1 = \alpha 1_n$, which corresponds to $\mu_1 = 0$, is the degenerate solution, which is forbidden by the second constraint. It is shown in [15] that the generalized eigenprojection problem (1) is solved by u_2, the lowest positive generalized eigenvector of (L, M), and that the Hall energy at the minimum is just μ_2. The subsequent local minima of (1) are obtained at $u_3, u_4, \ldots$, with the corresponding energies $\mu_3, \mu_4, \ldots$. If it is desired to plot the graph in a higher dimension, subsequent generalized eigenvectors may be taken. Thus, a 2-D drawing is obtained by taking the x-coordinates of the nodes to be given by u_2, and the y-coordinates to be given by u_3.

The special case analyzed by Hall [9], is obtained by taking graphs with positive weights and unit masses. In this case the vector u_2, also known as the *Fiedler vector*, is of fundamental importance to many fields besides graph drawing. Another special case is taking the mass m_i as the degree of the i'th node. Later, we show results of data drawn in this way. For more information, see [16].

3 The ACE Algorithm

Calculating the first few generalized eigenvectors of (L, M), is a difficult task that presents a real problem for standard algorithms when n becomes around 10^5. We suggest an algorithm that computes it very rapidly. ACE employs a technique common to the so-called algebraic multigrid (AMG) algorithms. These algorithms progressively express an originally high-dimensional problem in lower and lower dimensions, using a process called *coarsening*. On the coarsest scale the problem is solved exactly, and then starts a *refinement* process, during which the solution is progressively projected back into higher and higher dimensions, updated appropriately at each scale, until the original problem is reproduced. An AMG algorithm should be specifically designed for a given problem, so that the coarsening process keeps the essence of the problem unchanged, while the refinement process needs only fast updates to obtain an accurate solution at each scale.

As far as we know, this is the first time that a formal and rigorous AMG algorithm is developed for graph drawing. Several authors, including works from our own group, have designed "heuristic" multiscale/multilevel graph drawing algorithms, i.e., algorithms in which the relationship between the problems on the different scales is not rigorously defined; see [8, 10, 21, 6]. Another important heuristic multiscale algorithm, from which we draw some of our inspiration, was developed by Barnard and Simon [1]. It computes the Fiedler vector for use in the partitioning problem.

3.1 The Coarsening Process

Let G be a PSD graph containing n nodes. A single coarsening step would be to replace G with another PSD graph G^c containing only $m < n$ nodes (typically, $m \approx \frac{1}{2}n$). Of course, the structure of G^c should be strongly linked to that of G, such that both describe approximately the same entity, but on different scales. We now establish a general framework for the coarsening.

A key concept we will use is that of an *interpolation matrix*, which is the tool that links G and G^c. This is an $n \times m$

matrix A that interpolates m-dimensional vectors $y \in \mathbb{R}^m$ into n-dimensional ones $x = Ay$. If y is a solution of the generalized eigenprojection problem of G^c, a good interpolation matrix is one for which $x = Ay$ is close enough to a solution of the generalized eigenprojection problem of G. Such interpolation matrices can be designed in various ways, to be discussed in Subsection 3.3. In the meantime, we just state the general definition of A:

Definition 3.1 (Interpolation Matrix) *An* interpolation matrix A *is an $n \times m$ matrix (with $n > m$), such that*

1. *All elements of A are non-negative, $A_{ij} \geq 0 \quad \forall i, j$.*
2. *The sum of each row is one, $\sum_{j=1}^{m} A_{ij} = 1 \quad \forall i$.*
3. *A has a full column rank,* $\mathrm{rank}(A) = m$.

Properties 1 and 2 follow naturally by interpreting the i'th row of A as a series of weights that shows how to determine coordinate x_i given all the y's, namely, $x_i = \sum_{j=1}^{m} A_{ij} y_j$. Property 3 prevents the possibility of two distinct m-dimensional vectors being interpolated to the same n-dimensional vectors.

The interpolation matrix defines a coarsening step completely: Given a fine n-node PSD graph $G(L, M)$ and an $n \times m$ interpolation matrix A, we take the coarse m-node PSD graph to be $G^c(L^c, M^c)$. We now show how, given L, M, and A, we calculate the Laplacian L^c, and the mass matrix M^c.

3.1.1 Calculating the coarse mass matrix

The coarse masses are derived by:

Definition 3.2 (Mass law) *Let G be a fine n-node PSD graph with masses $m_1, m_2, \ldots, m_n$, and let A be an $n \times m$ interpolation matrix. The masses of the coarse PSD graph G^c, i.e., $m'_1, m'_2, \ldots, m'_m$, are given by:*

$$m'_j = \sum_{i=1}^{n} A_{ij} m_i.$$

Later we will justify this definition. In the meantime, let us just say that this law is nothing but a statement of mass conservation if we interpret the nodes of G as physical masses formed by breaking and re-fusing (as dictated by the interpolation matrix) pieces of the physical masses from which G^c is built up.

3.1.2 Calculating the coarse Laplacian

The Hall energy of $G(L, M)$ is $E = x^T L x$, where $x \in \mathbb{R}^n$. Substituting $x = Ay$, we can express this energy in terms of the m-dimensional vector y, $E = y^T A^T L A y$. It would then be quite natural to define the Laplacian of G^c to be

$$L^c = A^T L A,$$

so that the Hall energy of $G^c(L^c, M^c)$ is $E = y^T L^c y$. In [15] we prove that L^c is in itself a Laplacian, thus our definition is consistent.

Having found both L^c and M^c, G^c is completely defined and the coarsening step comes to its end. Our definition of coarsening clarifies why we needed the concept of PSD graphs; negative weights might occur in G^c even if all the weights of G are positive. A more detailed discussion of this appears in the full version of the paper, [15].

3.1.3 The fine and coarse minimization problems

Given the coarse graph G^c, its corresponding generalized eigenprojection problem is

$$\min_y y^T L^c y \quad \text{given } y^T M^c y = 1 \qquad (2)$$
$$\text{in the subspace } y^T M^c \cdot 1_m = 0.$$

The issue that needs to be addressed now is the relationship between this problem and the original generalized eigenprojection problem (1) of the fine graph G. To bring the two problems into the same dimension, we substitute $x = Ay$ in (1), obtaining the form

$$\min_y y^T A^T L A y \quad \text{given } y^T A^T M A y = 1 \qquad (3)$$
$$\text{in the subspace } y^T A^T M \cdot 1_n = 0.$$

Due to the equality $L^c = A^T L A$, the function to be minimized is the same in both forms. Moreover, since $M^c \cdot 1_m = A^T M \cdot 1_n$, we are looking for solutions to both of these in the same subspace. The only difference between them is in the scaling constraint, since in general $M^c \neq A^T M A$. One can force equality by adopting certain strategies for calculating A (see 3.3). Yet other strategies, that may yield more powerful interpolation matrices, do not, indeed, obey an equality. Nevertheless, in the full paper [15] we show that in these cases (2) is a reasonable approximation of (3), and consequently the solutions of both problems share much resemblance.

3.2 The Refinement Process

We keep coarsening further and further until we obtain a coarse PSD graph that is sufficiently small. Typically, we would stop the process when we have less than 100 nodes. The drawing of $G(L, M)$, the coarsest graph, is obtained from the lowest positive generalized eigenvectors of (L, M), namely $u_2, u_3, \ldots$. Since M is diagonal, we can formulate the problem as an eigenvalue problem $Bv = \mu v$, where $B = M^{-\frac{1}{2}} L M^{-\frac{1}{2}}$ and $v = M^{\frac{1}{2}} u$, which calls for finding the lowest positive eigenvectors of B. Due to the small size of B, finding its eigenvectors takes a negligible fraction of the total running time, which makes the choice of the algorithm a marginal issue. We chose to use a variation of the power iteration (PI) algorithm [22], which is designed for finding the largest eigenvectors of symmetric matrices. For S a symmetric matrix, its largest eigenvector is the asymptotic direction of $Sv_0, S^2 v_0, S^3 v_0 \ldots$, where v_0 is an initial guess. The next largest eigenvectors can be found using the same technique, taking v_0 orthogonal to the previously found (larger) eigenvectors. Our problem

needs the lowest eigenvectors rather than the largest ones. We therefore preprocess the matrix B, transforming it into a different matrix, $\hat{B}$, whose largest eigenvectors are identical with the lowest eigenvectors of B (see also [2]). This is done using the Gershgorin bound [22], which is a theoretical upper bound for (the absolute value of) the largest eigenvalue of a matrix. Specifically, for our matrix this bound is given by:

$$g = \max_i \left(B_{ii} + \sum_{j \neq i} |B_{ij}| \right).$$

The matrix $\hat{B} = g \cdot I - B$ has the same eigenvectors as B, but in reverse order — the largest eigenvectors has become the lowest ones. Consequently, it is now suitable for using with the PI algorithm. The pseudocode of this algorithm is given in Figure 1. The initial guesses $\hat{u}_2, \hat{u}_3, \ldots$ are picked at random. The PI seems to be a good algorithm to use here, being intuitive, simple to implement, having assured convergence to the right eigenvector, and most of all suitable to the refinement process too. Given a certain eigenvector of B, say v_i, we are able to calculate the generalized eigenvector u_i from $u_i = M^{-\frac{1}{2}} v_i$.

Function power_iteration $(\{\hat{u}_2, \hat{u}_3, \ldots, \hat{u}_p\}, L, M, \epsilon)$
% $\{\hat{u}_2, \hat{u}_3, \ldots, \hat{u}_p\}$ are initial guesses for $\{u_2, u_3, \ldots, u_p\}$
% L, M are the Laplacian and mass matrix of the graph
% ϵ is the tolerance. We recommend using $10^{-7} \leq \epsilon \leq 10^{-10}$

$v_1 \leftarrow M^{\frac{1}{2}} \cdot 1_n$ % first (known) eigenvector
$v_1 \leftarrow \frac{v_1}{\|v_1\|}$ % normalize the first eigenvector
$B \leftarrow M^{-\frac{1}{2}} L M^{-\frac{1}{2}}$
$g \leftarrow$ Gershgorin(B)
$\hat{B} \leftarrow g \cdot I - B$ % reverse order of eigenvalues
for $i = 2$ to p
  $\hat{v}_i \leftarrow M^{\frac{1}{2}} \hat{u}_i$
  $\hat{v}_i \leftarrow \frac{\hat{v}_i}{\|\hat{v}_i\|}$
  repeat
    $v_i \leftarrow \hat{v}_i$
    % orthogonalize against previous eigenvectors
    for $j = 1$ to $i - 1$
      $v_i \leftarrow v_i - (v_i^T v_j) v_j$
    end for
    $\hat{v}_i \leftarrow \hat{B} \cdot v_i$ % power iteration
    $\hat{v}_i \leftarrow \frac{\hat{v}_i}{\|\hat{v}_i\|}$ % normalization
  until $\hat{v}_i \cdot v_i > 1 - \epsilon$ % check convergence in direction
  $v_i \leftarrow \hat{v}_i$
  $u_i \leftarrow M^{-\frac{1}{2}} v_i$
end for
return $u_2, \ldots, u_p$

Figure 1: The power iteration.

Having solved the generalized eigenprojection problem of the coarsest PSD graph directly, we use the solution to accelerate the computation of the corresponding solution of the second coarsest PSD graph. We then proceed iteratively until the solution of the original problem is obtained.

How is this 'inductive step', from the coarser to the finer graph, to be carried out? Well, let $G^c(L^c, M^c)$ be a coarse m-node PSD graph, and let $u_2^c, u_3^c, \ldots u_p^c$ be the first few solutions of its generalized eigenprojection problem. For two-dimensional drawing u_2^c and u_3^c are all that we need (thus $p = 3$), but we do not specify p so as to keep the algorithm general. Let the next fine n-node PSD graph be $G(L, M)$, and let $u_2, u_3, \ldots, u_p$ be the first few solutions of its generalized eigenprojection problem. Let also A be the $n \times m$ interpolation matrix connecting G and G^c. The refinement step uses $u_2^c, u_3^c, \ldots, u_p^c$ to obtain a good initial guess for $u_2, u_3, \ldots, u_p$, thus enabling their fast calculation. The basic idea of the refinement is that for a reasonable interpolation matrix, $\hat{u}_i = A u_i^c$ is a good approximation of u_i. Then, we have to apply an iterative algorithm whose input is the initial guess $\hat{u}_i$ and whose output is the closest generalized eigenvector u_i.

The iterative algorithm that we use is the same power iteration which we have already used to solve the coarsest problem, see Figure 1.

3.3 The Interpolation Matrix

At this point, the multiscale scheme is completely defined, and the only thing left unexplained is how we construct a specific interpolation matrix A. As already mentioned, there is no unique recipe for this, and we can come up with many feasible methods. However, for the interpolation matrix to successfully fulfill its role, some guidelines should be followed: **a)** The interpolation matrix should faithfully retain the structure of G; i.e., similar fine nodes should be similarly interpolated. This is the most important property of the interpolation matrix, since it determines how close will the initial guess $A u_i^c$ be to the desired solution u_i. **b)** The interpolation matrix should be easy to compute in terms of running time. **c)** The sparser the interpolation matrix the better, since matrix manipulations will be faster.

Generally speaking, these guidelines are contradicting. Improving the preservation of the structure of a graph requires more accurate interpolation, and hence a denser and more complex matrix A. A good interpolation matrix thus reflects a reasonable compromise regarding the tradeoff between accuracy and simplicity. We now describe the two methods we have been using to construct A.

Edge Contraction Interpolation. In this method we interpolate each node of the finer graph from exactly one coarse node. To find an appropriate interpolation matrix, we first divide the nodes of the fine graph into small disjoint connected subsets, and then associate the members of each subset with a single coarse node. Consequently, the rows of all the members of the same subset in the interpolation matrix will be identical, having a single non-zero

element (which is, of course, 1). A well known method to efficiently create such disjoint subsets is by contracting edges that participate in max-matching (see, e.g., [21]), so that the cardinality of the subsets is either one or two.

The edge contraction method evidently prefers simplicity over accuracy. On the one hand, A is very sparse and is easy to compute, but on the other hand, the interpolation is crude, taking into consideration only the strongest connection of each node. The simple form of the resulting A gives rise to an elegant property of the edge contraction algorithm: it preserves the structure of the problem, that is, $M^c = A^T M A$, thus having identical structure to (2) and (3).

Weighted Interpolation In this method, inspired by [3], we interpolate each node of the fine graph from possibly several coarse nodes. The first stage is to choose a subset of m nodes out of the n nodes of G, which will be the nodes of G^c. Hereinafter, the elements of this subset, $R \subset V$, will be called *representatives*. A possible candidate for R could be a maximal independent set. For even better options, see [15]. We fix the coordinates of the representatives by their values as given in the coarse problem, and then determine the coordinates of the non-representatives as a weighted sum of their neighboring representatives, such that Hall's energy is minimized.

We denote the coarse node associated with the representative i by $[i]$. The interpolation matrix A is built as follows: Let P_i be the set of all representatives that are connected to fine node i by positive weights, and let p_i be the sum of these weights. Similarly, let N_i be the set of all representatives that are connected to fine node i by negative weights, and let n_i be their sum. Then we define:

$$\text{For } i \in V - R: \qquad (4)$$

$$A_{i[j]} = \begin{cases} w_{ij}/p_i & \text{if} j \in P_i \quad \text{and} \quad p_i \geq -n_i \\ w_{ij}/n_i & \text{if} j \in N_i \quad \text{and} \quad p_i < -n_i \\ 0 & \text{otherwise} \end{cases}$$

$$\text{For } i \in R:$$

$$A_{i[j]} = \begin{cases} 1 & \text{if} j = i \\ 0 & \text{if} j \neq i, \end{cases}$$

A complete derivation is given in the full paper, [15].

Clearly, in comparison with the edge contraction technique, this algorithm prefers accuracy over simplicity: A is less sparse, more expensive to compute, but far more accurate.

3.4 Comments on Running Time

In Subsection 3.3 we introduced two techniques for generating an interpolation matrix, edge contraction and weighted interpolation. Comparing the two with respect to the speed of ACE, necessitates taking into account two points, which will be briefly surveyed here. Further discussion and examples can be found in [15].

Sparsity vs. accuracy: The sparser the interpolation matrix, the faster is a single iteration of PI. Also, the more accurate is the interpolation matrix, the less PI iterations are required until convergence. It is difficult therefore to predict which would be faster in the refinement — the sparse but less accurate edge contraction, or the denser but more accurate weighted interpolation. We anticipate that the structure of homogeneous graphs, like grids, is satisfactorily preserved even with a sparse interpolation matrix. In these cases, we expect the edge contraction to be faster. However, for non-homogeneous graphs, the opposite is true, and we expect the weighted interpolation to be the faster.

Coarsening vs. refinement: Our implementation of the edge contraction method uses direct coarsening that does not involve matrix multiplication. Thus, with respect to coarsening time, edge contraction is much preferable to weighted interpolation. On the other hand, for non-homogeneous graphs, weighted interpolation exhibits faster refinement times. For relatively loose tolerance (large ϵ in PI), not much work is required during the refinement, and we can expect the coarsening time to be significant, resulting in faster performance of the edge contraction. However, as we decrease ϵ, the PI time becomes more and more dominant, yielding faster performance of the weighted interpolation. Sometimes, one is interested in computing more than two generalized eigenvectors. This might be the case when carrying out three-dimensional drawings, or in two-dimensional drawings where the coordinates are associated with generalized eigenvectors other than u_2 and u_3; see Figure 2e-f. In any case, when more than two generalized eigenvectors are requested, the expected PI time of ACE increases while the coarsening time does not change. Consequently, the advantages of weighted interpolation become more salient, which suggests to prefer it in such cases.

3.5 The Algorithm in a Nutshell

Here now is an outline of the full ACE algorithm, in the form of the recursive function AMG_Graph_Drawing:

Function AMG_Graph_Drawing (L, M)
% L — the Laplacian of the graph
% M — the mass matrix

if $dimension(L) < threshold$
  $(u_2, u_3) \leftarrow \text{direct_solution}(L^c, M^c)$
else
  Compute the interpolation matrix A ;
  Compute the Laplacian, $L^c \leftarrow A^T L A$;
  Compute the mass matrix M^c ;
  $(u_2^c, u_3^c) \leftarrow \text{AMG_Graph_Drawing}(L^c, M^c)$;
  $(\hat{u}_2, \hat{u}_3) \leftarrow (Au_2^c, Au_3^c)$
  $(u_2, u_3) \leftarrow \text{power_iteration}(\{\hat{u}_2, \hat{u}_3\}, L, M, \epsilon)$;
end if
return u_2, u_3

4 Examples

We have tested our algorithm against a variety of graphs, taken from diverse sources. Here we present some of the more interesting results, all obtained using a dual processor Intel Xeon 1.7GHz PC. The program is non-parallel and ran on a single processor.

Table 1 shows the results of applying ACE to a number of large graphs, each with more than 10^5 nodes, and gives a pretty good feeling for the speed of the algorithm. We used the edge contraction method, and a tolerance of $\epsilon = 10^{-7}$, asking ACE to compute the two generalized eigenvectors u_2 and u_3. In the table, we provide the complexity of the graphs, as well as the computation times of the different parts of ACE. The last column of the table gives the number of PI iterations performed on the finest scale (the most expensive ones) during the computation of u_2.

Graphs of around 10^5 nodes are drawn in a few seconds; with 10^6 nodes the algorithm takes 10-20 seconds. The largest graph consists of $7.5 \cdot 10^6$ nodes, and the running time is about two minutes. Thus, ACE exhibits a truly significant improvement in computation time for drawing large graphs. Moreover, one can use it to draw huge graphs of 10^6 – 10^7 nodes, which we have not seen dealt with appropriately in the literature, in quite a reasonable amount of time.

Unfortunately, limitations of file size and printing resolution prevent us from bringing here full drawings of really huge graphs. Yet, for the reader to obtain a visual impression of the kind of drawings produced by ACE, we bring here a collection of drawings of smaller graphs.

Before discussing specific examples, here are some general comments about the nature of drawings produced by minimizing Hall's energy. On the one hand, we are assured to be in a global minimum of the energy, thus we might expect the global layout of the drawing to faithfully represent the structure of the graph. On the other hand, there is nothing in Hall's energy that prevents nodes from being very close. Hence, the drawing might show dense local arrangement.

These general claims are nicely demonstrated in the examples drawn in Figure 2. Figure 2a shows a folded grid, obtained by taking a square grid, removing the horizontal edges in its central region, and connecting opposing corners. This graph has high level of symmetry, which is perfectly reflected in the drawing. Figures 2b-c show additional examples of symmetric graphs. Besides the excellent preservation of symmetry in the two-dimensional layout, these graphs show how ACE handles graphs in which many different "textures" are embedded. The drawing of Figure 2d, the 4elt graph, resembles the one shown in the technical report version of [21], which was obtained using a different graph drawing approach. Figures 2e-f show the same graph, dwa512, drawn using two different sets of generalized eigenvectors, $\{u_2, u_3\}$ and $\{u_3, u_4\}$, respectively. The graph is comprised of two grids, strongly connected via their centers. The Fiedler vector u_2 is known for its ability to divide the graph into its natural clusters, as is nicely demonstrated in Figure 2e. Whereas, Figure 2f reveals the grid-based structure.

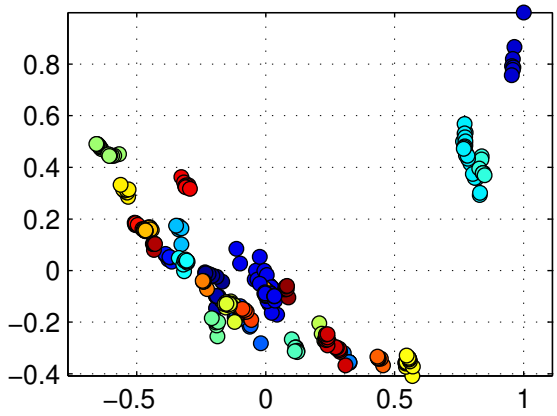

Figure 3: Application of ACE on data obtained from an electronic nose.

As to be expected from the previous discussion, these drawings indeed exhibit rather impressive global layout, but also have locally dense regions.

We would like to give an example how ACE can be applied to information visualization. *Electronic noses* are chemical devices used to quantify odors (see, e.g., [7]). They consist of an array of sensors that, upon stimulation by an odor, gives an odor-unique response pattern. These patterns are represented as multidimensional vectors (typically 8-64 dimensions), which calls for the use of exploratory visualization techniques. The data provided by the electronic nose can be transformed to the form of similarity matrix, and therefore can be represented as a graph (see details in [4]). Giving each node a mass equal to its degree, we obtain the drawing shown in Figure 3. Here, the data is comprised of $n = 300$ measurements, taken by repeatedly measuring 30 different kinds (color coded in the figure) of odors. Two observations should be made with respect to this drawing. First, the odors in the upper-right corner are real outliers. ACE isolates them dramatically from the other, "normal", odors. Second, the different clusters of odors are very well separated.

5 Discussion

It appears that the time performance of the ACE algorithm is sufficiently good to finally enable graph drawing tools to be used to visualize huge systems such as the World Wide Web or networks of protein-protein interactions. This remains to be seen.

Obviously, the quality of the algorithm's results depends on the appropriateness of Hall's energy to the problem of graph drawing. In comparison with many of the other energy functions used in graph drawing, Hall's energy is distinguished by its simple form. The tractability of

Table 1: Running times of the various components of ACE. Data for most graphs were taken from web sources, as indicated in the table. The graphs grid1000 and grid1415, which are simple rectangular grids, were produced by us.

Graph	Size		Degree			Times [sec]			PI iterations
Name	V	E	min	avg.	max	PI	coarsening	total	(finest level)
598a†	110,971	741,934	5	13.37	26	3.1	0.8	4	7
ocean‡	143,437	409,593	1	5.71	6	1	0.6	1.7	4
144†	144,649	1,074,393	4	14.86	26	4.4	1.2	5.7	8
wave§	156,317	1,059,331	3	13.55	44	3	0.8	3.9	12
m14b†	214,765	1,679,018	4	15.64	40	5.4	1.7	7.3	7
mrngA†	257,000	505,048	2	3.93	4	3.9	1.5	5.5	5
auto†	448,695	3,314,611	4	14.77	37	14.1	4.5	19.1	7
grid1000	1,000,000	1,998,000	2	4.00	4	2.3	3.5	6.2	3
mrngB†	1,017,253	2,015,714	2	3.96	4	14	7.8	22.3	6
grid1415	2,002,225	4,001,620	2	4.00	4	3.7	7.0	11.6	2
mrngC†	4,039,160	8,016,848	2	3.97	4	34.7	24.7	61.6	4
mrngD†	7,533,224	14,991,280	2	3.98	4	61.1	48.7	114.2	4

† Taken from George Karypis' collection at: `ftp.cs.umn.edu/users/kumar/Graphs`

‡ Taken from Francois Pellegrini's Scotch graph collection, at: `www.labri.u-bordeaux.fr/Equipe/PARADIS/Member/pelegrin/graph`

§ Taken from the University of Greenwich Graph Partitioning Archive, at: `www.gre.ac.uk/~c.walshaw/partition`

the mathematical analysis that this brings with it yields the vastly improved computation speed and a guaranteed convergence to a global minimum. All this lends ACE stability and causes its results to be "globally aesthetic". On the other hand, local details of the graph might be aesthetically inferior with respect to the results of other, more complicated, energy functions. For certain applications it might be possible to combine the advantages of different graph drawing algorithms, obtaining the global layout of a huge graph with ACE, and then use slower methods to beautify particular regions of interest.

Having ACE quickly determine the coordinates of the nodes of huge graphs poses new challenges for their actual display. Obviously, graphs with millions of nodes cannot be beneficially displayed or printed as is, and new display tools would be required. A promising direction is to display only a portion of a graph at any given time, using various smooth navigation tools. A survey of such methods can be found in, e.g., [12]. Another interesting approach is to use the coarse graphs produced by ACE during the coarsening process as abstractions of the original graph for various display purposes. Moreover, in line with the previous paragraph, we could use ACE to produce the global layout and its abstractions, and then use other algorithms for the instantaneous drawing of zoomed portions.

In developing the ACE algorithm we restricted ourselves to the problem of graph drawing. However, what we have been actually doing is to devise an algorithm that quickly finds the first few generalized eigenvectors of any problem of the form

$$Lx = \mu Mx, \tag{5}$$

with L a Laplacian and M a real diagonal positive definite matrix. It seems that these limitations on L and M can be removed by some modifications to the algorithm.

Problems of the form (5), with L a Laplacian and M real diagonal positive definite, appear frequently also outside graph drawing; for example, in image segmentation [19], partitioning [1], and linear ordering [13], and we hope that ACE may be found useful by researchers in these and other fields.

Finally, we would like to emphasize that the two algorithms we proposed for calculating an interpolation matrix should be taken as suggestions only. Since there is no strict criterion for the evaluation of an interpolation matrix, those that we were using are not necessarily "optimal", and in fact we have a number of additional ideas about this issue that require further inspection.

References

[1] S. T. Barnard and H. D. Simon, "A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems", *Concurrency: Practice & Experience* **6** (1994), 101–117.

[2] U. Brandes and T. Willhalm, "Visualizing Bibliographic Networks with a Reshaped Landscape Metaphor", Proc. 4th Joint Eurographics - IEEE TVCG Symp. Visualization (VisSym '02), pp. 159-164, ACM Press, 2002.

[3] A. Brandt, "Algebraic Multigrid Theory: The Symmetric Case", *Applied Mathematics and Computation*, **19** (1986) 23–56.

[4] L. Carmel, Y. Koren and D. Harel, "Visualizing and Classifying Odors Using a Similarity Matrix", to appear in the 9^{th} International Symposium on Olfaction and Electronic Nose (ISOEN02), September 29 – October 2, 2002.

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

[6] P. Gajer, M. T. Goodrich, and S. G. Kobourov, "A Multi-dimensional Approach to Force-Directed Layouts of Large Graphs", *Graph Drawing 2000*, LNCS **1984**, pp. 211–221, Springer Verlag, 2000.

[7] J. W. Gardner and P. N. Bartlett, *Electronic Noses, Principles and Applications*, Oxford University Press, New York, USA, 1999.

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

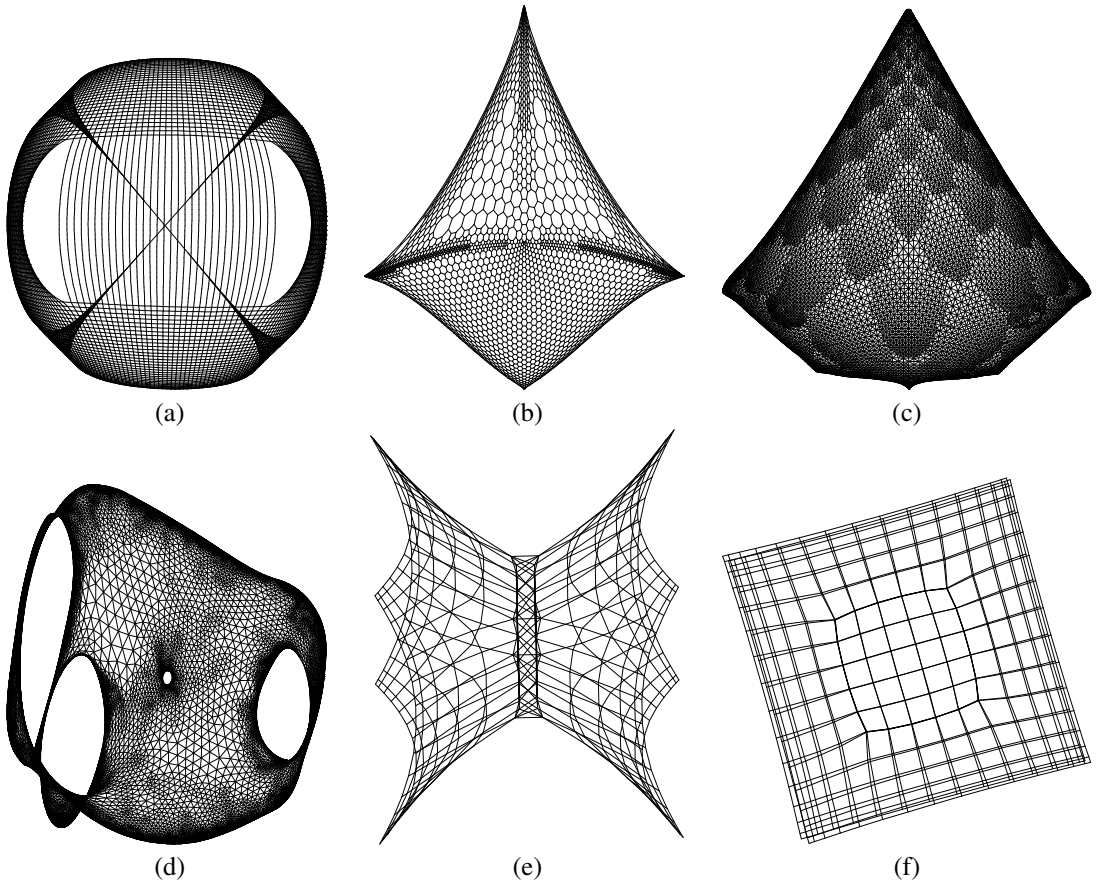

Figure 2: Examples of ACE drawings. **a)** A folded-grid, based on a 100×100 rectangular grid. $|V| = 10,000$, $|E| = 18,713$. **b)** The 4970 graph, taken from [21]. $|V| = 4970$, $|E| = 7400$. **c)** The Crack graph, taken from Jordi Petit's collection, at `www.lsi.upc.es/~jpetit/MinLA/Experiments`. $|V| = 10,240$, $|E| = 30,380$. **d)** The 4elt graph, taken from Francois Pellegrini's Scotch graph collection, at: `www.labri.u-bordeaux.fr/Equipe/PARADIS/Member/pelegrin/graph`. $|V| = 15,606$, $|E| = 45,878$. **e,f)** The dwa512 graph, taken from the Matrix Market, at `math.nist.gov/MatrixMarket`. $|V| = 512$, $|E| = 1004$. Drawn using $\{u_2, u_3\}$ (e) and $\{u_3, u_4\}$ (f).

[9] K. M. Hall, "An r-dimensional Quadratic Placement Algorithm", *Management Science* **17** (1970), 219-229.

[10] D. Harel and Y. Koren, "A Fast Multi-scale Method for Drawing Large Graphs", *Graph Drawing 2000*, LNCS **1984**, pp. 183–196, Springer Verlag, 2000.

[11] D. Harel and Y. Koren, "Graph Drawing by High-Dimensional Embedding", to appear in *Proceedings of Graph Drawing 2002*, Springer Verlag.

[12] I. Herman, G. Melancon and M. S. Marshall, "Graph Visualisation and Navigation in Information Visualisation", *IEEE Transactions on Visualization and Computer Graphics* **6** (2000), 24–43.

[13] M. Juvan and B. Mohar, "Optimal linear labelings and eigenvalues of graphs",*Discrete Applied Mathematics* **36** (1992), 153–168.

[14] M. Kaufmann and D. Wagner (Eds.), *Drawing Graphs: Methods and Models*, LNCS **2025**, Springer Verlag, 2001.

[15] Y. Koren, L. Carmel and D. Harel "ACE: A Fast Multiscale Eigenvectors Computation for Drawing Huge Graphs", Technical Report MCS01-17, The Weizmann Institute of Science, 2001. Available at: www.wisdom.weizmann.ac.il/reports.html.

[16] Y. Koren "On Spectral Graph Drawing", manuscript, 2002.

[17] C. C. Paige and M. A. Saunders, "Solution of Sparse Indefinite Systems of Linear Equations", *SIAM J. Numer. Anal.* **12** (1975), 617–629.

[18] A. Quigley and P. Eades, "FADE: Graph Drawing, Clustering, and Visual Abstraction", *Graph Drawing 2000*, LNCS **1984**, pp. 183–196, Springer Verlag, 2000.

[19] J. Shi and J. Malik, "Normalized Cuts and Image Segmentation", *IEEE Conference Computer Vision and Pattern Recognition*, pp. 731–737, 1997.

[20] K. Stueben, "An Introduction to Algebraic Multigrid", Appendix A in *Multigrid* (U. Trottenberg, C.W. Oosterlee, and A. Schuller, eds.), Academic Press, London, 2000.

[21] C. Walshaw, "A Multilevel Algorithm for Force-Directed Graph Drawing", *Graph Drawing 2000*, LNCS **1984**, pp. 171–182, Springer Verlag, 2000.

[22] D. S. Watkins, *Fundamentals of Matrix Computations*, John Wiley, New York, NY, USA, 1991.