MULTISCALE MODEL. SIMUL.
Vol. 8, No. 5, pp. 1599–1620

A FAST MULTIGRID ALGORITHM FOR ENERGY MINIMIZATION UNDER PLANAR DENSITY CONSTRAINTS*

DORIT RON†, ILYA SAFRO‡, AND ACHI BRANDT†

Abstract. The two-dimensional layout optimization problem reinforced by the efficient space utilization demand has a wide spectrum of practical applications. Formulating the problem as a nonlinear minimization problem under planar equality and/or inequality density constraints, we present a linear time multigrid algorithm for solving a correction to this problem. The method is demonstrated in various graph drawing (visualization) instances.

Key words. layout problems, graph drawing and visualization, density constraints, constrained optimization, geometric multigrid, full approximation scheme

AMS subject classifications. 65N55, 76M27, 90-08

DOI. 10.1137/090771995

1. Introduction.

The optimization problem addressed in this paper is to find an optimal layout of a set of two-dimensional (2D) objects such that (a) the total length of the given connections between these objects will be minimal, (b) the overlapping between objects will be as little as possible, and (c) the 2D space will be well used. This class of problems can be modeled by a graph in which every vertex has a predefined shape and area and each edge has a predefined weight. While the first two conditions are straightforward, the third requirement can be made concrete in different ways. To see its usefulness, consider, for example, the problem of drawing the "snake"-like graph shown in Figure 1.1(a). Most graph drawing algorithms would draw it as a line or a chord. In that case, when the number of nodes is big, the space is used very inefficiently, and the size of the nodes must decrease. One possible efficient space utilization for the graph "snake" is presented in Figure 1.1(b).

In many theoretical and industrial fields, this class of problems is often addressed and actually poses a computational bottleneck. In this work, we present a multilevel solver for a model that describes the core part of those applications, namely, the problem of minimizing a quadratic energy functional under planar constraints that bound the allowed amount of material (total areas of objects) in various subdomains of the entire domain under consideration. A similar definition appears, for example, in [6].

Given an initial arrangement, the main contribution of this work is to enable a *fast* rearrangement of the entities under consideration into a more evenly distributed state over the entire defined domain. This process is done by introducing a sequence of finer and finer grids over the domain and demanding, at each scale, *equidensity*,

*Received by the editors September 23, 2009; accepted for publication (in revised form) June 29, 2010; published electronically September 7, 2010. This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract DE-AC02-06CH11357. The U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes. Copyright is owned by SIAM to the extent not limited by these rights.
http://www.siam.org/journals/mms/8-5/77199.html
†Department of Applied Math, The Weizmann Institute of Science, Rehovot 76100, Israel (dorit.ron@weizmann.ac.il, abrandt@math.ucla.edu).
‡Department of Mathematics and Computer Science, Argonne National Laboratory, Argonne, IL 60439 (safro@mcs.anl.gov).

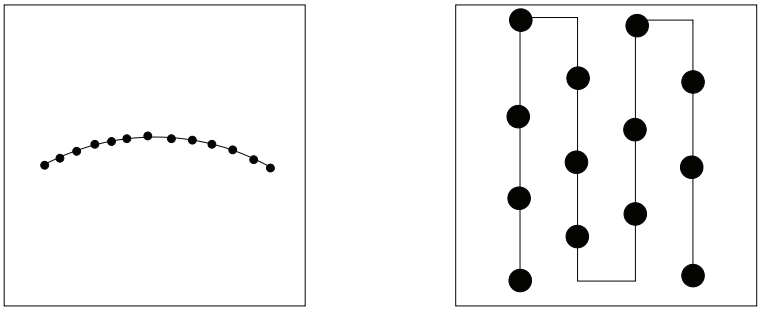

FIG. 1.1. *Possible ways to draw the "snake"-like graph:* (a) *when the drawing area is not used, the size of the nodes must decrease; and* (b) *a clearer picture is obtained when the space is used efficiently.*

that is, meeting equality or inequality constraints at each grid square, stating how much material it may (at most) contain. Since many variables are involved and since the needed updates may be large, we introduce a new set of *displacement* variables attached to the introduced grid points, which enables *collective* moves of many original variables at a time, at different scales including large displacements. The use of such multiscale moves has two main purposes: to enable processing in various scales and to *efficiently* solve the (large) system of equations of energy minimization under equidensity demands. The system of equations of the finer scales, when many unknowns are involved, is solved by a combination of well-known multigrid techniques (see [3, 4, 17]), namely, the *correction scheme* for the energy minimization part and the *full approximation scheme* for the inequality equidensity constraints defined over the grid's squares. We assume here that the minimization energy functional has a quadratic form, but other functionals can be used via quadratization. The entire algorithm solves the nonlinear minimization problem by applying successive steps of corrections, each using a linearized system of equations.

Clearly, for each specific application, one has to tune the general algorithm to respect the particular task at hand. We have chosen here to demonstrate the performance of our solver in some instances of the graph visualization problem showing the efficient use of the given domain. Let us review a few applications that have motivated our research.

Graph visualization. This addresses the problem of constructing a geometric representation of graphs and has important applications to many technologies. There are many different demands for graph visualization problems, such as draw a graph with a minimum number of edge crossings, or a minimum total edge length, or a predefined angular resolution (for a complete survey, see [2]). The ability to achieve a compact picture (without overlapping) is of great importance, since area-efficient drawings are essential in practical visualization applications where screen space is one of the most valuable commodities. One of the most popular strategies that does address these questions is the force-directed method [8], which has a quadratic running time if all pairwise vertex forces are taken into account. There are several successful multilevel algorithms [11] developed to improve the method's complexity. However, reducing the running time in these models usually means a loss of information regarding those forces.

Representation of higraphs. Higraphs, a combination and extension of graphs

and Euler/Venn diagrams, were defined by Harel in [9]. Higraphs extend the basic structure of graphs and hypergraphs to allow vertices to describe inclusion relationships. Adjacency of such vertices is used to denote set-theoretic Cartesian products. Higraphs have been shown to be useful for the expression of many different semantics and underlie many visual languages, such as statecharts and object model diagrams. The well-known force-directed method has been extended to enable handling the visualization of higraphs [10]. For small higraphs it has indeed yielded nice results; but, because of its high complexity, it poses efficiency challenges when used for larger higraphs.

Facility location problem. In this class of problems the goal is to locate a number of facilities within a minimized distance from the clients. In many industrial versions of the problem there exist additional demands, such as the minimization of the routing between the facilities and various space constraints (e.g., the factory planning problem) while being given a total area on which the facilities and clients could be located (for a complete survey, see [7]).

Wireless networks and coverage problems. These have a broad range of applications in the military, surveillance, environmental monitoring, and healthcare fields. In these problems, having a limited number of resources (like antennae or sensors), one has to cover the area on which many demand points are distributed and have to be serviced. In many practical applications there are predefined connections between these resources that can be modeled as a graph [12, 5].

The placement problem. The electronics industry has achieved phenomenal growth over the past two decades, mainly due to the rapid advances in integration technologies and large-scale systems design—in short, due to the advent of VLSI. The number of applications of integrated circuits in high-performance computing, telecommunications, and consumer electronics has been rising steadily and at a very fast pace. Typically, the required computational power of these applications is the driving force for the fast development of this field. The global placement is one of the most challenging problems during VLSI layout synthesis. In this application the modules must be placed in such a way that the chip can be processed at the detailed placement stage and then routed efficiently under many different constraints. This should be accomplished in a reasonable computation time, even for circuits with millions of modules, since it is one of the bottlenecks of the design process. For the most recent survey of the placement techniques see [13].

This paper is organized as follows. The problem definition is described in section 2. The multilevel formulation and solver are presented in section 3. Examples of graph drawing layout corrections are demonstrated in section 4. Finally, in section 5, we conclude and discuss possible future work.

2. Problem definition. Given a weighted undirected graph $G = (V, E)$, let $vol(i) > 0$ be the (rectangular) area of vertex (node) $i \in V$, $i = 1, \ldots, |V|$, and w_{ij} be the nonnegative weight of the edge ij between nodes i and j ($w_{ij} = 0$ if $ij \notin E$). Also, assume a 2D *initial* layout is given; that is, the center of mass of node i is considered to be located at $(\tilde{x}_i, \tilde{y}_i)$ within a given rectangular domain. The purpose of the optimization problem we consider is to modify the initial assignment $(\tilde{x}, \tilde{y})$ by (ξ_x, ξ_y) so as to minimize the quadratic functional

$$\mathfrak{E}(\xi_x, \xi_y) = \frac{1}{2} \sum_{ij \in E} w_{ij} \big((\tilde{x}_i + \xi_{x_i} - \tilde{x}_j - \xi_{x_j})^2 + (\tilde{y}_i + \xi_{y_i} - \tilde{y}_j - \xi_{y_j})^2 \big), \tag{2.1}$$

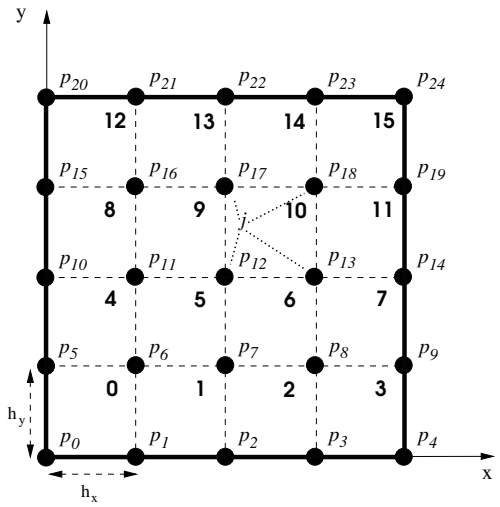

FIG. 2.1. *Example of a grid $\mathcal{G}$ with* 25 *grid points and* 16 *squares. The grid points and squares are labeled by p_i and bold numbers, respectively.*

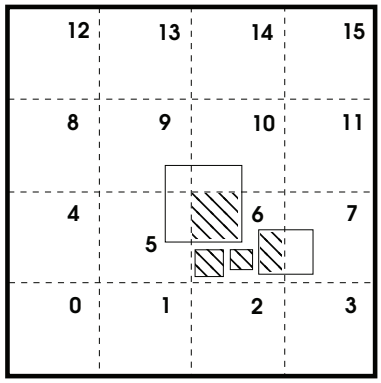

FIG. 2.2. *Example of $Y(s)$ for square* 6. *The total area of vertices overlapping with square* 6 *is dashed.*

subject to some *equidensity* demands on the area distribution of the nodes within the given rectangle. To apply such constraints, we discretize the domain by a standard grid $\mathcal{G}$ consisting of a set of squares $\mathcal{S}(\mathcal{G})$, where each square $s \in \mathcal{S}(\mathcal{G})$ is of area $\mathcal{A} = h_x h_y$, and h_x and h_y are the mesh sizes of $\mathcal{G}$ in the x^- and y^- directions, respectively (see Figure 2.1). Denote by $Y(s)$ the total area of the vertices overlapping with the square s; that is, $Y(s)$ is the sum over all the nodes coinciding with s, each contributing the (possibly partial) area that overlaps with s (see Figure 2.2).

The *planar* constraints (i.e., the constraints that are distributed over the 2D plane, where each constraint defines a demand regarding some bounded area) can then simply state how much area is required to be in every square; that is, for each square $s \in \mathcal{S}(\mathcal{G})$, the constraint is either $Y(s) = M(s)$, or $Y(s) \leq M(s)$, where $M(s)$ is the number of node areas desired or allowed for square s.

The constrained optimization problem with equality or inequality formulation can

thus be summarized by the following:

$$\begin{array}{ll} \textit{minimize} & \mathfrak{E} \quad \text{(given by (2.1))} \\ \textit{subject to} & Y(s) = (\leq) M(s) \quad \forall s \in \mathcal{S}(\mathcal{G}). \end{array} \tag{2.2}$$

3. The multilevel formulation and solver. The aim of the current work is to provide a *fast* first-order correction to the given approximate solution; that is, we are looking for such a displacement that would in some optimal sense (to be defined below) improve the planar equidensity demands and/or decrease $\mathfrak{E}$. (Note that unconstrained minimization of $\mathfrak{E}$ will bring all nodes to overlap at a single point, and thus we may often observe an *increase* in $\mathfrak{E}$ upon removing some of the initial overlap.)

To enable a direct use of the multigrid paradigm, and motivated by the need to perform *collective* moves of nodes (as explained in the introduction), we have actually reformulated the problem (2.1), as described in section 3.1. The multilevel solver of the (reformulated) system (3.10) below is introduced in section 3.2. This system of equations actually has to be solved for a *sequence* of different grid sizes to enhance the overall equidensity for a variety of scales, as presented in section 3.3.

3.1. Formulation of the correction problem. We have first introduced two new sets of variables u and v that correspond to *displacements* in the horizontal and vertical directions, respectively. These variables are located at the *grid points* $\mathcal{P}(\mathcal{G})$ which are sequentially counted from 0 to $|\mathcal{P}(\mathcal{G})|-1$, as shown in Figure 2.1. Each point $p \in \mathcal{P}(\mathcal{G})$ is associated with two variables u_p and v_p that influence the displacements of all the nodes located in the (up to four) squares intersecting at p. For example, the horizontal update of (the center of mass of) node j, depicted in Figure 2.1, is obtained from points p_{12}, p_{13}, p_{17}, and p_{18}:

$$x_j \leftarrow x_j + \alpha_{12,j} u_{12} + \alpha_{13,j} u_{13} + \alpha_{17,j} u_{17} + \alpha_{18,j} u_{18},$$

where α_{12}, α_{13}, α_{17}, and α_{18} are the standard bilinear interpolation coefficients (their sum equals 1). The vertical coordinate y_j is updated from the v variables using the *same* coefficients.

For a node i, denote the set of four closest points in $\mathcal{P}(\mathcal{G})$ (the corners of the square in which its center of mass is located) by $c(i)$. The new quadratic energy functional we would like to minimize for u and v given a current layout $(\tilde{x}, \tilde{y})$ of G (i.e., the coordinates of node i are initialized with $(\tilde{x}_i, \tilde{y}_i)$) is

$$\begin{aligned} \mathfrak{E}(u,v) = \frac{1}{2} \sum_{ij \in E} w_{ij} \Bigg[& \left(\tilde{x}_i + \sum_{p \in c(i)} \alpha_{pi} u_p - \tilde{x}_j - \sum_{p \in c(j)} \alpha_{pj} u_p \right)^2 \\ & + \left(\tilde{y}_i + \sum_{p \in c(i)} \alpha_{pi} v_p - \tilde{y}_j - \sum_{p \in c(j)} \alpha_{pj} v_p \right)^2 \Bigg], \end{aligned} \tag{3.1}$$

where α_{pi} are the bilinear interpolation coefficients.

The reformulation of the equidensity constraint in terms of the displacement variables relies on the rule of conservation of areas. The initial total amount of vertex areas at each square equals the current actual amount of areas dictated by $(\tilde{x}, \tilde{y})$. To estimate the amount of areas flowing inside and outside a given square induced by the u and v displacements, we assume the nodes are evenly distributed inside the squares. Under this assumption it is easier to estimate the amount of area being transferred between two adjacent squares, as explained below. Consider, for example, a square

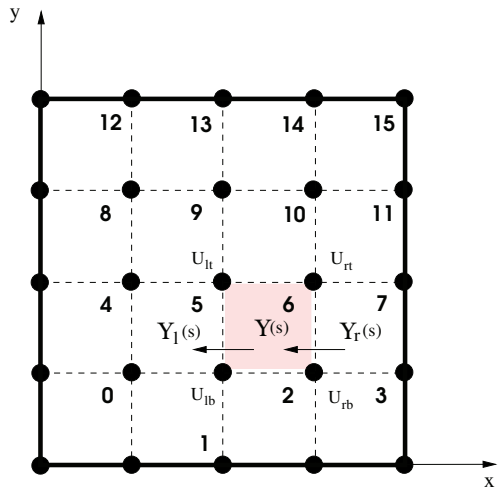

FIG. 3.1. *The horizontal direction flows of area considered for the square s (colored in square 6) in the equidensity constraint* (3.2).

s. Denote by $Y_{r(l,t,b)}(s)$ the total area of the overlap between the nodes and s's *right (left, top, bottom) neighbor* square. Let $u_{rt(rb,lt,lb)}(s)$ be the u values at the *top-right (bottom-right, top-left, bottom-left)* corner of s, as shown in Figure 3.1. To estimate the amount of areas entering s from the right, we first calculate the average area (per square unit) in *both* squares: $(Y(s)+Y_r(s))/2\mathcal{A}$. We have to multiply this by the actual entering area (of nodes), which is a rectangle of height h_y, the length of the border between the two squares, and the width, which is the average of the u displacement at the middle of that border, namely, $(u_{rt} + u_{rb})/2$. Thus the overall contribution of area from the right is approximated by

$$\frac{Y(s) + Y_r(s)}{2\mathcal{A}} \cdot h_y \cdot \frac{u_{rt}(s) + u_{rb}(s)}{2}.$$

A similar term is calculated at the left, and with v instead of u also at the top and bottom. Note that if the assumed direction of flow is wrong, the resulting displacement will just turn out to be negative.

The entire constraint for a square s stating that the net flow of areas into the square should be equal to or be smaller than some demand $M(s)$ minus the current area in u is given below:

$$\begin{aligned}(3.2)\quad \mathfrak{eqd}(s) = {} & \frac{Y(s) + Y_r(s)}{2\mathcal{A}} h_y \frac{u_{\mathrm{rt}}(s) + u_{\mathrm{rb}}(s)}{2} - \frac{Y(s) + Y_l(s)}{2\mathcal{A}} h_y \frac{u_{\mathrm{lt}}(s) + u_{\mathrm{lb}}(s)}{2} \\ & + \frac{Y(s) + Y_t(s)}{2\mathcal{A}} h_x \frac{v_{\mathrm{rt}}(s) + v_{\mathrm{lt}}(s)}{2} - \frac{Y(s) + Y_b(s)}{2\mathcal{A}} h_x \frac{v_{\mathrm{rb}}(s) + v_{\mathrm{lb}}(s)}{2} \le M(s) - Y(s).\end{aligned}$$

(For a possibly more accurate calculation of these constraints see the end of section 5.)

Next, to enforce the natural boundary conditions on u and v, namely, to forbid flows across the external boundaries, we simply nullify all corresponding u_p on the right and left boundary points $\mathcal{B}_u(\mathcal{G})$, and v_p on the bottom and top boundary points $\mathcal{B}_v(\mathcal{G})$. Then the entire constrained optimization problem in terms of u and v and the

initial approximation $(\tilde{x}, \tilde{y})$ is given by

$$\begin{array}{ll} \textit{minimize} & \mathfrak{E}(u,v) \quad \text{(given by (3.1)} \\ \textit{subject to} & \mathfrak{eq}\mathfrak{d}(s) = (\leq) M(s) - Y(s) \quad \forall s \in \mathcal{S}(\mathcal{G}); \\ & \text{if } p \in \mathcal{B}_u(\mathcal{G}), \text{ then } u_p = 0; \\ & \text{if } p \in \mathcal{B}_v(\mathcal{G}), \text{ then } v_p = 0. \end{array} \tag{3.3}$$

We will simplify the formulation of (3.3) by the concatenation of the two vectors u and v into one: $\mathbf{u} = [\{u_i\}_{i=0}^{|\mathcal{P}(\mathcal{G})|-1} \mid \{v_i\}_{i=0}^{|\mathcal{P}(\mathcal{G})|-1}]$. We will also omit the boundary conditions by directly replacing all variables in $\mathcal{B}_u(\mathcal{G}) \cup \mathcal{B}_v(\mathcal{G})$ by 0, and so, from now on, we will refer to $\mathfrak{E}$ as

$$\mathfrak{E}(\mathbf{u}) = \frac{1}{2} \sum_{i,j} q_{ij} \mathbf{u}_i \mathbf{u}_j + \sum_i l_i \mathbf{u}_i + C, \tag{3.4}$$

where i, j run over all the indices in $\mathbf{u} \setminus \mathcal{B}_u(\mathcal{G}) \setminus \mathcal{B}_v(\mathcal{G})$, C is a constant, and q_{ij}, l_i are the coefficients calculated directly from the previous definition (3.1) of $\mathfrak{E}$. Similarly, rewrite each $\mathfrak{eq}\mathfrak{d}(s)$ in (3.2) as

$$\mathfrak{eq}\mathfrak{d}(s) = \sum_i a_{si} \mathbf{u}_i = (\leq) b_s, \tag{3.5}$$

where $b_s = M(s) - Y(s)$.

Denote by λ_s, $s \in \mathcal{S}(\mathcal{G})$, the Lagrange multiplier corresponding to the equidensity constraint of square s. If all the constraints are equality ones, the Lagrangian minimization functional is

$$\mathfrak{L}(\mathbf{u}, \lambda) = \mathfrak{E}(\mathbf{u}) + \sum_{s \in \mathcal{S}(\mathcal{G})} \lambda_s (\mathfrak{eq}\mathfrak{d}(s) - b_s). \tag{3.6}$$

So, we are looking for a critical point of the Lagrangian function, which is expressed by the system of linear equations

$$\nabla \mathfrak{L}(\mathbf{u}, \lambda) = \begin{bmatrix} \nabla_{\mathbf{u}} \mathfrak{L}(\mathbf{u}, \lambda) \\ \nabla_{\lambda} \mathfrak{L}(\mathbf{u}, \lambda) \end{bmatrix} = 0. \tag{3.7}$$

There are at least two factors that may cause (3.7) to be singular. First, the rank of $\nabla \mathfrak{L}(\mathbf{u}, \lambda)$ is always less than its size by at least 1. This arises from the equations of equidensity constraints in (3.7): their sum always equals zero. The reason is that under the boundary constraints the total number of in-flows is always equal to the total number of out-flows. In fact, the second summand in (3.6) can be replaced by

$$\sum_{s \in \mathcal{S}(\mathcal{G})} (\lambda_s + Z)(\mathfrak{eq}\mathfrak{d}(s) - b_s)$$

for any Z without changing the minimization of $\mathfrak{L}$ since

$$Z \sum_{s \in \mathcal{S}(\mathcal{G})} (\mathfrak{eq}\mathfrak{d}(s) - b_s) = 0.$$

Thus, what is important are not the values of λ_s but only their *differences*, and the singularity can be treated by an additional constraint, say, $\sum_s d_s \lambda_s = 0$, where

$d_s = 1\ \forall s \in \mathcal{S}(\mathcal{G})$ (the introduction of d_s is necessary for the recursion of the multilevel solver; see section 3.2.1). The additional term in $\mathfrak{L}(\mathbf{u}, \lambda)$ is $\eta \sum_s d_s \lambda_s$, where η is a "pseudo-Lagrange" multiplier. The following proposition (with $k = 1$) motivates the nonsingularity of $\mathfrak{L}$ with $\sum_s d_s \lambda_s = 0$.

PROPOSITION 3.1. *Given a symmetric $n \times n$ matrix A, for which* rank $(A) = n-k$, *let x_i, $i = 1, \ldots, k$ be an orthogonal basis of the null space of A, that is, $Ax_i = 0$. Then the following block matrix B is nonsingular:*

$$B = \left(\begin{array}{c|c} A & X \\ \hline X^T & 0 \end{array} \right),$$

where $X = (x_1, \ldots, x_k)$ is an $n \times k$ matrix of rank k.

Proof. Let y be any vector in $\mathbb{R}^{n+k}$. Denote by y' the first n components of y and by y'' the last k components, that is, $y = (\frac{y'}{y''})$. We will prove that if $By = 0$, then $y = 0$. The vector By can be written in the following block form:

$$By = \left(\frac{Ay' + Xy''}{X^T y'} \right).$$

Multiplying $Ay' + Xy'' = 0$ by X^T from the left implies that $y'' = 0$, and hence $Ay' = 0$ and $y' = \sum_{i=1}^{k} \alpha_i x_i$. Substituting the last relation into each of the last k rows of B implies $x_j^T y' = x_j^T \sum_{i=1}^{k} \alpha_i x_i = \alpha_j x_j^T x_j = 0$, and thus $\alpha_j = 0$ for $j = 1, \ldots, k$, yielding $y' = 0$. Since $y' = 0$ and $y'' = 0$, we may conclude that $y = 0$ as needed. ∎

The second kind of singularity in (3.7) may appear from possible empty squares. This can be treated by adding a summand to (3.6) that minimizes the total sum of all corrections $\beta \sum_i \mathbf{u}_i^2$, that is, adds a 2β-term to the diagonal of $\nabla_{\mathbf{u}} \mathfrak{L}$, where β is small enough to cause only negligible change in a solution. This will prevent the inclusion of zero-rows in $\nabla_{\mathbf{u}} \mathfrak{L}$, while possibly also bounding the size of each correction in the solver below.

To summarize, the pseudo-Lagrangian functional $\mathfrak{L}$ for our correction problem with equality constraint is

(3.8)
$$\mathfrak{L}(\mathbf{u}, \lambda, \eta) = \frac{1}{2} \sum_{i,j} q_{ij} \mathbf{u}_i \mathbf{u}_j + \sum_i l_i \mathbf{u}_i + \beta \sum_i \mathbf{u}_i^2 + \sum_{s \in \mathcal{S}(\mathcal{G})} \lambda_s \left(\sum_i a_{si} \mathbf{u}_i - b_s \right) + \eta \sum_{s \in \mathcal{S}(\mathcal{G})} d_s \lambda_s,$$

leading to the following system of equations:

$$\begin{aligned} &\frac{1}{2} \sum_j q_{ij} \mathbf{u}_j + l_i + 2\beta \mathbf{u}_i + \sum_{s \in \mathcal{S}(\mathcal{G})} \lambda_s a_{si} = 0 && \forall i \text{ s.t. } \mathbf{u}_i \in \mathbf{u} \setminus \mathcal{B}_u(\mathcal{G}) \setminus \mathcal{B}_v(\mathcal{G}), \\ &\sum_i (a_{si} \mathbf{u}_i - b_s) + \ \eta d_s = 0 && \forall s \in \mathcal{S}(\mathcal{G}), \\ &\sum_{s \in \mathcal{S}(\mathcal{G})} d_s \lambda_s = 0. \end{aligned} \tag{3.9}$$

Since in real world situations the total area is usually *bigger* than the total area of all the vertices, the redefined minimization problem under *inequality* constraints will generally have the form

$$\begin{array}{lll} \textit{minimize} & \mathfrak{E}(\mathbf{u}) & \text{(given by (3.4))}, \\ \textit{subject to} & \mathfrak{eqd}(s) \le b_s \ \ \forall s \in \mathcal{S}(\mathcal{G}) & \text{(given by (3.5))}. \end{array} \tag{3.10}$$

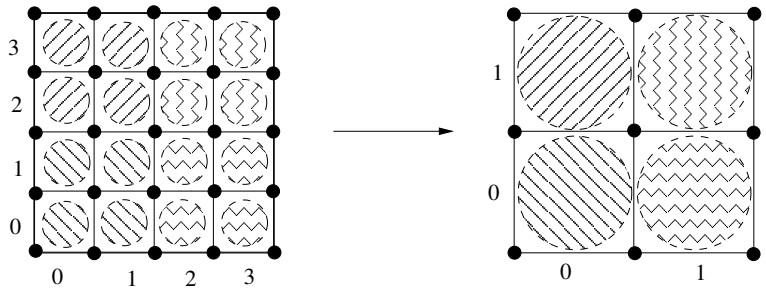

FIG. 3.2. *Geometric coarsening. The equidensity constraints of every four similarly patterned squares at the fine level form one equidensity constraint at the coarse level.*

3.2. Multilevel solver for problem (3.10). To solve the constrained minimization problem (3.10), we use multigrid techniques: standard geometric coarsening, linear interpolation, the correction scheme (CS) for the energy minimization, and the full approximation scheme (FAS) for the equidensity inequality constraints; all are presented in section 3.2.1. In addition, we have developed a fast window minimization relaxation, as explained in section 3.2.2. The multilevel cycle is schematically summarized in section 3.2.3 in algorithm **2D-layout-correction**.

3.2.1. Coarsening scheme. When the geometry of the problem is known we can choose a coarser grid by the usual elimination of every other line, as shown in Figure 3.2. The correction computed at the coarse grid points will be interpolated and added to the fine grid current approximation. Let us introduce the notation distinguishing between fine and coarse level variables and functions. By lowercase and uppercase letters we will refer to the variables, indexes, and coefficients of the fine ($\mathbf{u}_i$, i, q_j, etc.) and the coarse ($\mathbf{U}_I$, I, Q_J, etc.) levels, respectively. The subscripts f and c will be used to describe the energy $\mathfrak{E}_f$ and $\mathfrak{E}_c$ and pseudo-Lagrangian $\mathfrak{L}_f$ and $\mathfrak{L}_c$ functions at the fine and the coarse levels, respectively.

Thus, the minimization part of the pseudo-Lagrangian (3.8) at the fine level is

$$\mathfrak{E}_f = \frac{1}{2}\sum_{ij} q_{ij}\mathbf{u}_i\mathbf{u}_j + \sum_i l_i\mathbf{u}_i. \tag{3.11}$$

(Note that we have omitted the β term from the following derivation since it is merely an artificial added term.) Given a current approximation $\tilde{\mathbf{u}}$ of the fine level solution $\mathbf{u}$ and a correction function $\mathbf{U}$ calculated at the coarse level variables $\mathbf{U}$, $\tilde{\mathbf{u}}$ will be corrected by

$$\tilde{\mathbf{u}}_i \leftarrow \tilde{\mathbf{u}}_i + \sum_{I \ni i} \alpha_{iI}\mathbf{U}_I, \tag{3.12}$$

where the notation $\sum_{I\ni i}$ means that the sum is running over all coarse grid points p_I, from which standard bilinear interpolation is made to the fine grid point p_i.

Expressing the fine level energy functional E_f in terms of the coarse variables by

substituting (3.12) into (3.11) yields

$$\mathfrak{E}_f = \frac{1}{2}\sum_{ij} q_{ij}\left(\tilde{\mathbf{u}}_i + \sum_{I\ni i}\alpha_{iI}\mathbf{U}_I\right)\left(\tilde{\mathbf{u}}_j + \sum_{J\ni j}\alpha_{jJ}\mathbf{U}_J\right) + \sum_i l_i\left(\tilde{\mathbf{u}}_i + \sum_{I\ni i}\alpha_{iI}\mathbf{U}_I\right)$$
$$= \frac{1}{2}\sum_{IJ} Q_{IJ}\mathbf{U}_I\mathbf{U}_J + \sum_I L_I\mathbf{U}_I + C,$$

where $Q_{IJ} = \sum_{\substack{i\in I\\ j\in J}} q_{ij}\alpha_{iI}\alpha_{jJ}$, $L_I = \sum_{\substack{j\\ i\in I}} q_{ij}\tilde{\mathbf{u}}_j\alpha_{iI} + \sum_{i\in I} l_i\alpha_{iI}$, and C is a constant. Thus, the coarse level energy functional will be of the same structure as the fine level one, namely,

$$\mathfrak{E}_c = \frac{1}{2}\sum_{IJ} Q_{IJ}\mathbf{U}_I\mathbf{U}_J + \sum_I L_I\mathbf{U}_I.$$

For each fine square s the equidensity constraint $\mathfrak{eqd}(s)$ is given by (3.5). The coarse equidensity constraints are constructed by merging 2×2 fine squares into one coarse square S. The expression "$s \in S$" will refer to running over the four fine squares s that form the coarse square S (see Figure 3.2). The Sth planar equidensity constraint of the coarse level (in the case of equality constraints only) is obtained again by the substitution of (3.12):

$$\sum_{s\in S}\sum_i a_{si}\mathbf{u}_i - \sum_{s\in S} b_s = \sum_I A_{SI}\mathbf{U}_I - B_S,$$

where $A_{SI} = \sum_{i\in I}\sum_{s\in S} a_{si}\alpha_{iI}$ and $B_S = \sum_{s\in S}(b_s - \sum_i a_{si}\tilde{\mathbf{u}}_i)$. Similarly (in the case of equality constraints), the additional η-constraint over all squares at the coarse level as inherited from the fine level is $\sum_S D_S\Lambda_S = 0$, where $D_S = \sum_{s\in S} d_s$.

To complete the description of the coarse equations, we still need to transfer the equidensity *inequality* constraints. For this purpose we will use the FAS, which is the general multigrid strategy applied to nonlinear problems (see [3, 4, 17]). In fact, there is no need for the FAS for the equality equidensity constraints since it is a linear problem that can be solved by the regular CS. The FAS-like coarsening rules are needed and applied only on the set of equations derived from the equidensity inequalities. Thus, our scheme is a combination of the CS for the energy equations derived from (3.11) and (3.12) and FAS-like rules for the equidensity equations.

To derive these equations, we need to calculate the *residuals* for both the fine and the coarse grids. If $\mathfrak{L}_f$ is the pseudo-Lagrangian of the fine level system defined by

$$\mathfrak{L}_f = \mathfrak{E}_f + \sum_s \lambda_s\left(\sum_i a_{si}\mathbf{u}_i - b_s\right) + \eta\sum_s d_s\lambda_s, \tag{3.13}$$

where $\mathfrak{E}_f$ is given by (3.11), then the $\mathbf{u}_i$th residual of $\nabla\mathfrak{L}_f$, where $\tilde{\lambda}_s$ is the current value of the Lagrange multiplier λ_s, is

$$r_i^{\mathfrak{E}} = -l_i - \frac{1}{2}\sum_j q_{ij}\tilde{\mathbf{u}}_j - \sum_s \tilde{\lambda}_s a_{si}.$$

Thus, the residual corresponding to the variable $\mathbf{U}_I$ of $\nabla\mathfrak{L}_c$ (where $\nabla\mathfrak{L}_c$ is the coarse level system of equations analogous to (3.13)) is

$$R_I^{\mathfrak{E}} = \sum_{i\in I}\alpha_{iI} r_i^{\mathfrak{E}}, \tag{3.14}$$

where α_{iI} are as in (3.12); that is, the fine-to-coarse transfer is the adjoint of our coarse-to-fine interpolation. The residual of the sth equidensity constraint is

$$r_s^{\mathfrak{eqd}} = b_s - \sum_i a_{si}\tilde{\mathbf{u}}_i - \tilde{\eta}d_s,$$

where s runs over all fine squares and $\tilde{\eta}$ is the current value of η. Therefore, the coarse equidensity residual of square S is

$$R_S^{\mathfrak{eqd}} = \sum_{s\in S} r_s^{\mathfrak{eqd}}. \tag{3.15}$$

Finally the residual of the η-constraint is

$$r_\eta = -\sum_s d_s\tilde{\lambda}_s = R_H.$$

Denote by $LP(I)$ the linear part of the $\mathbf{U}_I$th equation in the system $\nabla\mathfrak{L}_c$:

$$LP(I) = \frac{1}{2}\sum_J Q_{IJ}\mathbf{U}_J + \sum_S \Lambda_S A_{SI}.$$

From the FAS rule for the Ith coarse equation stating that $LP(I) = R_I^{\mathfrak{E}}+$ the current approximation of $LP(I)$, we can derive the Ith $\nabla\mathfrak{L}_c$ equation

$$\frac{1}{2}\sum_J Q_{IJ}\mathbf{U}_J + \sum_S \Lambda_S A_{SI} - R_I^{\mathfrak{E}} - \frac{1}{2}\sum_J Q_{IJ}\mathbf{U}_J^0 - \sum_S \Lambda_S^0 A_{SI} = 0, \tag{3.16}$$

where $R_I^{\mathfrak{E}}$ is given by (3.14), $\mathbf{U}_J^0 = 0$, and $\Lambda_S^0 = \frac{1}{4}\sum_{s\in S}\tilde{\lambda}_s$. Similarly, the Sth square coarse equation for the equality (inequality) constraint is

$$\sum_I A_{SI}\mathbf{U}_I + HD_S - R_S^{\mathfrak{eqd}} - \sum_I A_{SI}\mathbf{U}_I^0 - H^0 D_S = (\leq)0, \tag{3.17}$$

where $R_S^{\mathfrak{eqd}}$ is given by (3.15). The last equation for the H-constraint is

$$\sum_S D_S\Lambda_S - R_H - \sum_S D_S\Lambda_S^0 = 0. \tag{3.18}$$

Note that (3.16)–(3.18) are the coarse grid equations analogous to the system (3.9). (A $2\beta\mathbf{U}_I$ term may be added to (3.16) for stability if needed.) The correction received from the coarse level for the $\mathbf{u}$ variables is given by (3.12) and for the Lagrange multipliers λ by

$$\tilde{\lambda}_s \leftarrow \tilde{\lambda}_s + \Lambda_{S\ni s} - \Lambda_{S\ni s}^0. \tag{3.19}$$

3.2.2. Relaxation. In our multigrid solver, as usual, the relaxation process is employed as the smoother of the error of the approximation before the construction of the coarse level system and immediately after interpolation from the coarse level. For this purpose we have developed the *window relaxation* procedure, which extracts from the entire system small subproblems of $m \times m$ supersquares and solves each separately, as explained below and summarized in **SingleWindowSolver**.

Let $\mathcal{W} = \{s \in \mathcal{S}(\mathcal{G}) \mid \text{all squares within an } m \times m \text{ supersquare}\}$ be a window of squares. To solve the quadratic minimization problem in $\mathcal{W}$, we fix at their current position all $\mathbf{u}$ *outside* $\mathcal{W}$, as well as all those which are on the boundary of $\mathcal{W}$ and represent movement *perpendicular* to $\mathcal{W}$. The minimization is done under the set of equidensity constraints for the squares $s \in \mathcal{W}$. The solution process for each single window is a simplified version of the *active set method* [1] and is iterative. At each iteration t, we first extract the set of squares S_t for which the respective inequality equidensity constraints are *violated* or *almost violated* according to (3.5):

$$S_t = \{s \in \mathcal{W} \mid \mathfrak{eq}\mathfrak{d}(s) > b_s - \epsilon\},$$

where $\epsilon > 0$; we have used $\epsilon = 0.0001*$(the square's area). Then the inequality constraints of S_t are set to equalities. Let $\mathcal{P}_{\mathcal{W}}$ be the set of all $\mathbf{u}$ indexes inside $\mathcal{W}$ (including those on the boundary of $\mathcal{W}$ directing *parallel* to it). For every $\mathbf{u}_i$, $i \in \mathcal{P}_{\mathcal{W}}$, we associate a correction variable δ_i and reformulate the pseudo-Lagrangian for $\mathcal{W}$ as a functional of the δ_i variables analogous to (3.8):

$$\begin{aligned}(3.20)\quad \mathfrak{L}_{\mathcal{W}}(\delta,\lambda) = {} & \frac{1}{2}\sum_{i,j\in\mathcal{P}_{\mathcal{W}}} q_{ij}(\tilde{\mathbf{u}}_i+\delta_i)(\tilde{\mathbf{u}}_j+\delta_j) + \frac{1}{2}\sum_{i\in\mathcal{P}_{\mathcal{W}},\ j\notin\mathcal{P}_{\mathcal{W}}} q_{ij}(\tilde{\mathbf{u}}_i+\delta_i)\tilde{\mathbf{u}}_j \\ & + \sum_{i\in\mathcal{P}_{\mathcal{W}}} l_i(\tilde{\mathbf{u}}_i+\delta_i) + \beta\sum_{i\in\mathcal{P}_{\mathcal{W}}}(\tilde{\mathbf{u}}_i+\delta_i)^2 + \sum_{s\in S_t}\lambda_s\left(\sum_{i\in\mathcal{P}_{\mathcal{W}}} a_{si}\tilde{\mathbf{u}}_i - b_s\right),\end{aligned}$$

where $\tilde{\mathbf{u}}_i$ is the current value of $\mathbf{u}_i$ and the β term is added for stability with $\beta = 1$. Solving $\nabla\mathfrak{L}_{\mathcal{W}}(\delta,\lambda) = 0$ we obtain the corrections δ_i for $\tilde{\mathbf{u}}_i$, $i \in \mathcal{P}_{\mathcal{W}}$, which confine the respective active set variables to the boundary of the equality constraint manifold. However, while accepting this correction, we may violate other inequality constraints that were already satisfied at the previous iteration $t-1$. This can be avoided by accepting only a *partial* correction $\theta\delta$, $\theta < 1$. We continue to the next iteration $t+1$, excluding from the redefined S_{t+1} the set of satisfied (by equality) constraints from S_t with negative Lagrange multipliers λ_s. The number of iterations is kept small; we have used up to 6.

SingleWindowSolver($\mathcal{W}$, $\tilde{\mathbf{u}}$)
begin
  $t = 0$
  Repeat until all constraints are satisfied **or** $t < 6$
    If $t = 0$
      $S_t = \{$the violated equidensity constraints$\}$
    Else
      $S_t = \{$the violated equidensity constraints$\}\backslash$
        $\{$those in S_{t-1} which satisfy equality and have $\lambda_s < 0\}$
    Solve $\nabla\mathfrak{L}_{\mathcal{W}}(\delta,\lambda) = 0$ and extract θ
    Accept the correction $\tilde{\mathbf{u}} \leftarrow \tilde{\mathbf{u}} + \theta\delta$
    $t \leftarrow t+1$
end

To achieve corrections for all variables, we cover by these windows the entire area in red-black order [17]. For computational reasons we have chosen to apply this relaxation for very small windows (of size 4×4 squares). To minimize the effects of the boundary of the windows and to enforce the equidensity constraints over different

windows, we scan the entire domain two more times: once with half-window size *shift* in the horizontal direction and once in the vertical direction. Thus the overall relaxation process covers the domain three times.

3.2.3. The multilevel cycle. Having defined the window relaxation, the interpolation, and the coarsening scheme, the multilevel cycle naturally follows. Starting from the given approximation $(\tilde{x}, \tilde{y})$, discretize the domain by a standard grid on which the $\mathbf{u}$ variables are initially defined. Construct the system of equations (3.9), and solve for the $\mathbf{u}$ variables as follows. After applying ν_1 window relaxation sweeps, define the coarser level equations for the coarser grid, apply ν_1 window relaxation sweeps there, and continue to a still coarser level. This process is recursively repeated until a small enough problem is obtained. Solve this coarsest problem directly, and start the uncoarsening stage by interpolating the solution of the coarse level to the finer levels followed by ν_2 window relaxation sweeps on the finer level. Repeat until the correction to the original problem is obtained. This entire multilevel cycle, usually referred to as the *V-cycle*, is summarized in procedure **V-cycle-correction** below, where the superscript index refers to the level number. (We have used $\nu_1 = \nu_2 = 3$.)

V-cycle-correction($\mathcal{G}^i$, $\mathbf{u}^i$, $\mathcal{C}^i$, λ^i, $\nabla\mathfrak{L}^i$)
begin
  If $\mathcal{G}^i$ is a small enough grid
    Solve the problem exactly
  Else
    Set $\mathbf{u}^i = 0$
    Apply ν_1 *window relaxation* sweeps
    Construct $\mathcal{G}^{i+1}$ the coarse level grid
    Define $\mathcal{C}^{i+1}$ to be the set of equidensity constraints
    Initialize the system of equations $\nabla\mathfrak{L}^{i+1}$ given by (3.16)–(3.18)
    Initialize $\mathbf{u}^{i+1}$ and λ^{i+1}
    V-cycle-correction($\mathcal{G}^{i+1}$, $\mathbf{u}^{i+1}$, $\mathcal{C}^{i+1}$, λ^{i+1}, $\nabla\mathfrak{L}^{i+1}$)
    Interpolate from level $i+1$ to level i using (3.12) and (3.19)
    Apply ν_2 *window relaxation* sweeps
Return $\mathbf{u}^i$

3.3. The full multigrid external driving routine. The solution of (3.9) is primarily dependent on the chosen grid size. To enforce equidensity at all scales, it can be used within the full multigrid (FMG) framework. This is done by using a *sequence* of increasing grid sizes (progressively finer mesh sizes), while employing a small number of V-cycles for each grid size. Then, a new linear system can be formulated around the new solution to obtain yet a new correction, and so forth. Thus, by small steps of corrections, we solve the original nonlinear problem via the corrections calculated from the linear system of equidensity constraints. For instance, we have tried to employ grids of sizes $2, 4, 8, \ldots$ up to a grid with the number of squares comparable to the number of nodes in the graph. For each grid size the corresponding set of equations (in terms of the displacement $\mathbf{u}$) is solved either directly (for small enough grids) or by employing the V-cycles described in section 3.2.3. In either case the obtained solution $\mathbf{u}$ is *interpolated* back to the (x, y) variables, introducing the desired correction to the original variables of the problem. We emphasize that the original problem (2.2) is highly nonlinear, while the system of equations with corrections in terms of the displacement $\mathbf{u}$ is linearized around the current solution $(\tilde{x}, \tilde{y})$. Therefore, it might happen that only a *fraction* should actually be taken from the $\mathbf{u}$ displacements when

these $(\tilde{x}, \tilde{y})$ are being updated (we have limited these updates to a maximum of half the mesh size).

Various driving routines can actually be used: each chosen grid size may be solved more than once (e.g., use grids $2, 2, 4, 4, 8, \ldots$); the entire sequence of grids may be repeated (e.g., $2, 2, 4, 4, 8, \ldots, 2, 2, 4, 4, 8, \ldots$), and so on. (See section 4 for examples.) These parameters should in fact be optimized for each application according to the concrete needs of the model. However, it should be remembered that the goal here is to find a better minimum and a local minimum, not a global one. The process thus should be stopped when no significant improvement is achieved anymore. Our objective is compounded of both the energy functional and the quality of the equidensity constraints in different grids, and thus it should be stopped when neither of the above advances substantially. In our current code we have not yet included any such automatic stopping criteria; we use only some predefined sequences.

The entire algorithm for the 2D layout correction is summarized below in algorithm **2D-layout-correction**, where the superscript 0 refers to the current chosen grid size.

2D-layout-correction(graph G, current layout $(\tilde{x}, \tilde{y})$)
begin
  Apply for a sequence of grid sizes
    Construct and initialize $\mathcal{G}^0$, $\mathbf{u}^0$
    Define $\mathcal{C}^0$ to be the set of equidensity constraints
    Initialize the system of equations $\nabla\mathfrak{L}^0$
    V-cycle-correction($\mathcal{G}^0$, $\mathbf{u}^0$, $\mathcal{C}^0$, λ^0, $\nabla\mathfrak{L}^0$)
    Update $(\tilde{x}, \tilde{y})$ from $\mathbf{u}^0$
Return $(\tilde{x}, \tilde{y})$

3.4. Complexity of the algorithm and running times. The total complexity of the algorithm with equality constraints depends on two main parts: the complexity of the construction of the linear system and the size of the grid used for the domain discretization. The construction of the linear system of equations is linear in the number of edges in the graph since each node contributes to four u variables and to four v variables; thus each edge appears in (at most) 32 different equations. After the construction of the system, the running time of the V-cycle itself depends on the finest grid size used for the discretization of the domain. In practice, there is no need to create a discretization grid of linear size of more than twice $\sqrt{|V|}$.

To show that, we have measured running times (of a standard C++ program on a Lenovo laptop with 2.53GHz processor and 3GB RAM) for some of the hypergraphs at the Peko benchmark [14] (Suite3) after translating them into graphs by replacing each hyperedge by a clique, i.e., by taking all possible pairs of edges within. See Table 3.1. We have used four different mesh sizes for each produced graph: 16×16, 32×32, 64×64, and 128×128. The running times of one V-cycle for the equality constraint algorithm are shown in Figure 3.3; clearly, the running time is linear in the number of edges in the graph and in the number of grid points. The relation between these two factors was calculated by least squares, showing that the best linearity is seen when the number of edges is divided by 139.2 compared to the number of points in the mesh. For example, for the largest graph we have run, Peko15, one V-cycle with six relaxation sweeps (of 4×4 windows) at each level for grid size 128×128 took 7.5 seconds. We have further measured the time that three relaxation sweeps take and found out that, as expected, it depends only on the number of grid points

TABLE 3.1
These graphs were obtained from the Peko benchmark of hypergraphs at [14] *(Suite3) by replacing each hyperedge by all possible pairs of simple edges involved in it.*

Graph	Number of vertices	Number of edges
Peko04	27938	206629
Peko09	54114	444351
Peko12	71595	731095
Peko14	148760	1068055
Peko15	162935	1594475

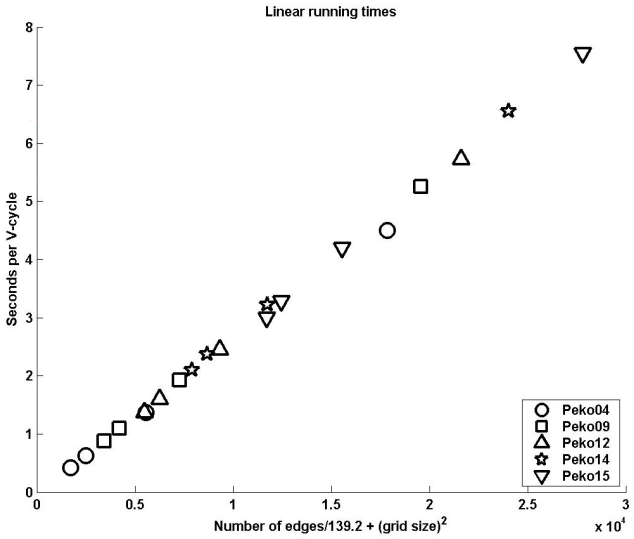

FIG. 3.3. *Linear running times of one V-cycle of the equality constraint multilevel algorithm. There are four marks for each graph which correspond (from the bottom-left to the top-right) to the four different grid sizes used:* 16×16, 32×32, 64×64, *and* 128×128. *The relation between the number of edges and the number of grid points was calculated by least squares.*

and does not depend on the size of the original graph. In particular, three relaxation sweeps for grid size 128×128 involve 127×127 different 4×4 windows and take about 0.4 seconds, while each 4×4 window takes 2.5×10^{-5} of a second.

We have actually written two versions for the algorithm, one for the equality constraints and one for the inequality ones. The first one has been more optimally coded, while the second one has not. The main difference between the two algorithms is the relaxation process, whose running time strongly depends on the algorithm for solving one window. There exist many versions of well-known algorithms for the quadratic minimization problem under linear inequality constraints (for a survey see [1]). However, since each window need be solved only to a first approximation (because of the iterative nature of the overall algorithm), in order to keep the running time low, we have implemented the above simple algorithm for approximately solving each single window. It is clear that several parameters must for efficiency be kept very small. For example, (1) the number of *window relaxation* sweeps should be fixed between 1 and 3; (2) the number of iterations t in **SingleWindowSolver** needs to be small, say, $t < 6$; and (3) the size of $\mathcal{W}$ in **SingleWindowSolver** is very robust; that is, the same results can be obtained with sizes 4×4, 8×8, and 16×16. We have used only 4×4, as it runs the fastest. With the above time measurements, an *upper bound* to

the above Peko15 example (with $t = 5$ and three relaxation sweeps) is 11.8 seconds, which includes the 7.5 seconds and (at the most) eight additional relaxation sweeps (of three iterations each) at all levels.

4. Examples of graph drawing layout correction. As previously mentioned, the graph drawing problem is of interest for many applications. Therefore, we have chosen to demonstrate the abilities of our algorithm for this problem. In this section we will present several results of the 2D layout correction algorithm using inequality constraints. The set of examples is shown in Figure 4.1 and Figures 4.5–4.8, each organized in two columns. The initial and final layouts of the graph are shown in the same row, in the left and the right columns, respectively. Note that finalizing the "nice graph" representation of these examples is beyond the scope of this work. The various "beautifying" procedures used by different applications may, of course, be used at the end of our cycles to enhance the visualization results.

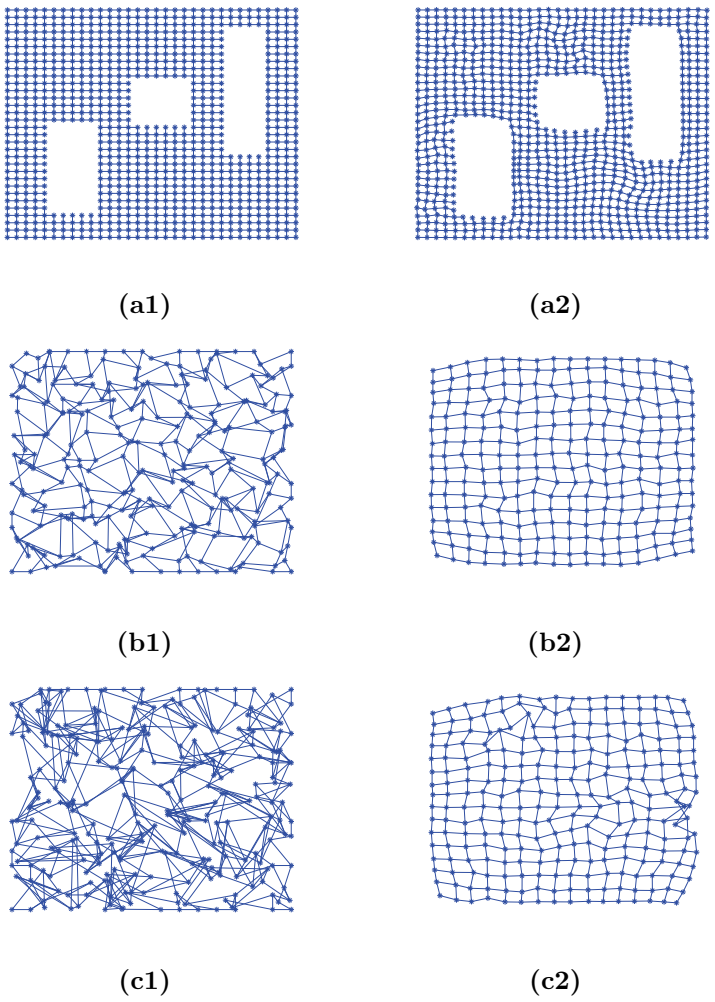

FIG. 4.1. *Examples of the 2D layout of graphs with equal vertices.*

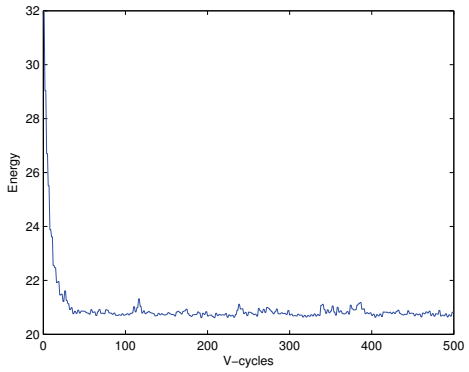

FIG. 4.2. *Energy behavior of the mesh at Figure* 4.1, row (c)*, when employing complete V-cycles with* 16×16 *and* 32×32 *alternately.*

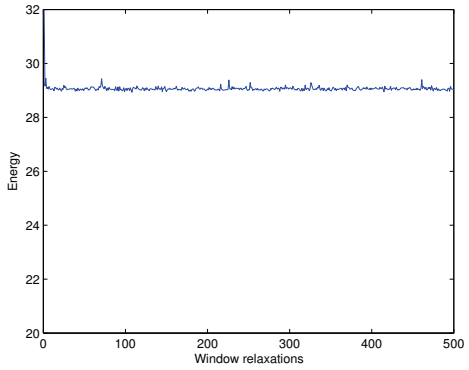

FIG. 4.3. *Energy behavior of window relaxation iterations (*16×16 *grid) of the mesh at Figure* 4.1, row (c).

The first example consists of a mesh graph with three holes (Figure 4.1, row (a)). It is intended to demonstrate that the empty space stays empty, and the energy is thus kept low. More complicated examples are shown in Figure 4.1, rows (b) and (c). The initial optimal positions of the mesh's vertices were randomly changed by independent shifts in different directions within a distance d:

$$d \leq \begin{cases} 2h_x & \text{in example (b)}, \\ 4h_x & \text{in example (c)}, \end{cases}$$

where h_x is the length of a square on the initially taken 32×32 grid, such that the mesh size of the graph is actually $2h_x$. Let us call these meshes M_1 and M_2, respectively. While the correction of M_1 looks really nice, two switched vertices on the right-hand side of M_2 demonstrate a weak point in our algorithm that certainly must be improved by a local "beautifying" procedure, which in general depends on the real application. The initial layout (c1) is more complicated than (b1), while the desired final layouts are similar.

A typical example of the energy behavior is presented in Figures 4.2–4.4. These figures refer to the mesh example in Figure 4.1, row (c). The general energy minimiza-

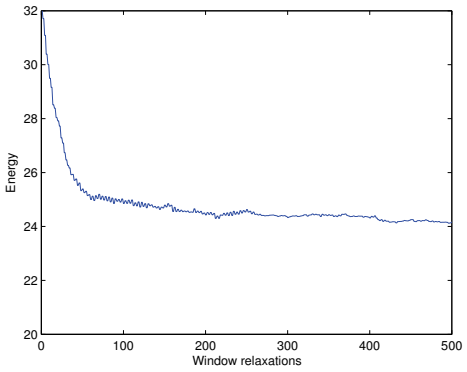

FIG. 4.4. *Energy behavior of window relaxation iterations (*16×16 *and* 32×32 *grids) of the mesh at Figure* 4.1, row (c).

tion progress is shown in Figure 4.2. In this example the driving routine alternates between two grid size V-cycles: each odd V-cycle solves the correction problem for the 16×16 grid, while even V-cycles improve the previous iterations with the grid 32×32. Figures 4.3 and 4.4 show the energy behavior of the window relaxations (without V-cycles) for 16×16 grid iterations and alternately 16×16 and 32×32 grid sizes, respectively. Clearly, the V-cycle algorithm is more powerful in minimizing the energy than just employing the window relaxations.

A more complicated example is shown in Figure 4.5, in which the 64×64 mesh graph is randomly perturbed by vertex shifts (up to $2h_x$ of a 64×64 grid), compressed at the bottom-left corner and augmented by 50 randomly chosen edges (Figure 4.5(a)). The final result of the algorithm is presented in Figure 4.5(c), where all vertices are placed almost at their optimal locations (note the different scales of the two figures). We have used two FMG-cycles with two V-cycles at each level as the main driving routine. Such a driving routine works with the following grid sizes: $2, 2, 4, 4, 8, 8, \ldots, 128, 128, 2, 2, 4, 4$, and so forth. After these two FMG-cycles the total energy was very close to its real minimum, and additional iterations have only slightly corrected the layout. If, however, for each grid size, instead of a V-cycle we use only a relaxation sweep and this is repeated 1000 times, we end up with the picture shown in Figure 4.5(b), which is clearly much different from the minimum. This example shows the power of the V-cycles. The next experiment consists of the 64×64 compressed mesh with three holes. The initial and final layouts are presented in Figures 4.6(a) and 4.6(b), respectively. These two examples show how the space is indeed well utilized, and that the initial compressed configuration is stretched while maintaining its structure.

Two additional examples demonstrate the layout corrections for graphs whose vertices have nonequal volumes (see Figures 4.7 and 4.8). In both cases the initial layout of these graphs was random.

It should be noted that whether or not the nodes are evenly distributed over the domain depends mostly on the initial approximation. If, for example, in an extreme example all nodes are initiated at the same spot, then it is obvious that they will never be separated by the moves suggested in this paper. So, we assume that the

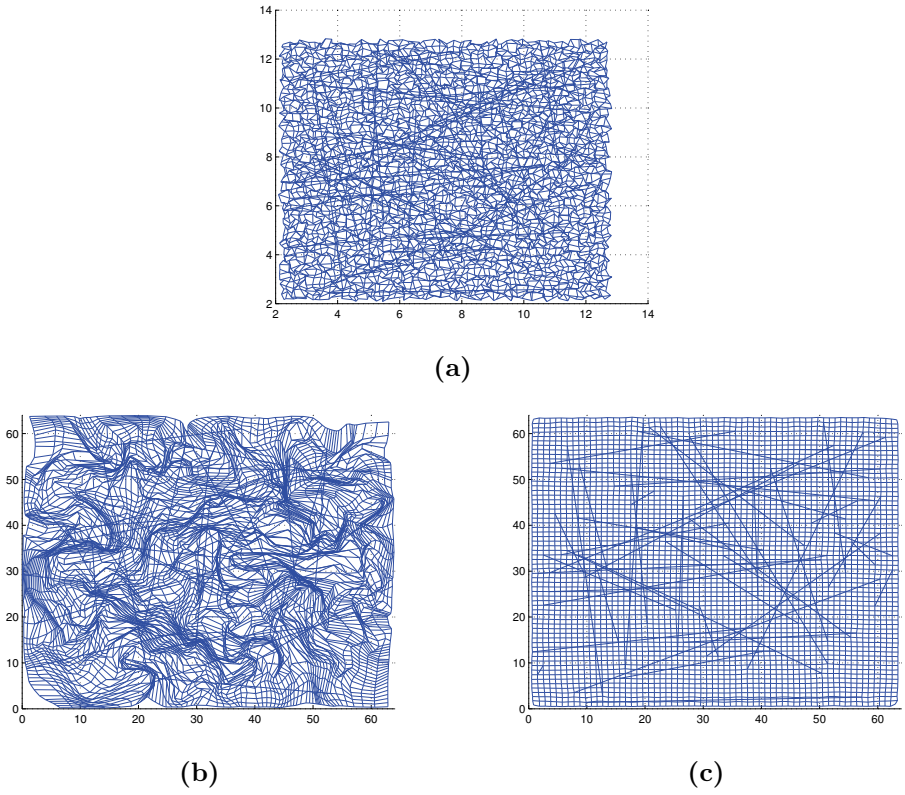

FIG. 4.5. *Example of the layout of the* 64×64 *mesh with additional random edges (note the different scales of figure* (a) *compared with those of* (b) *and* (c)*):* (a) *starting from a compressed and perturbed configuration at the bottom-left corner;* (b) *the resulting picture using only relaxation sweeps;* (c) *the resulting picture using V-cycles.*

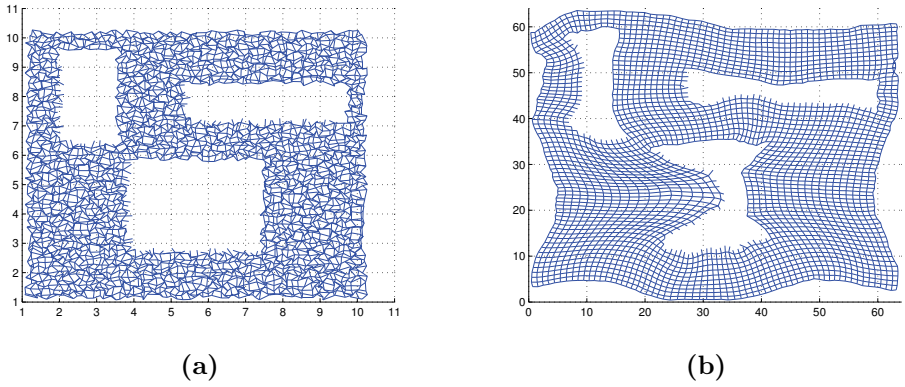

FIG. 4.6. *Example of the* 64×64 *mesh with three-hole layout (note the different scales of the two figures):* (a) *starting from a compressed and perturbed configuration at the bottom-left corner;* (b) *the resulting picture using V-cycles.*

initial approximation does not suffer severely from overlap among the nodes; then the moves induced by the **u** variables are smooth and will usually remove some of the

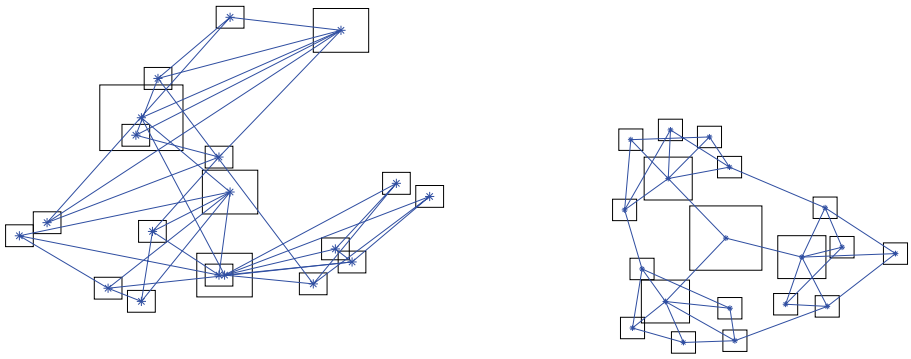

FIG. 4.7. *Example of the 2D layout of a graph with nonequal volumes.*

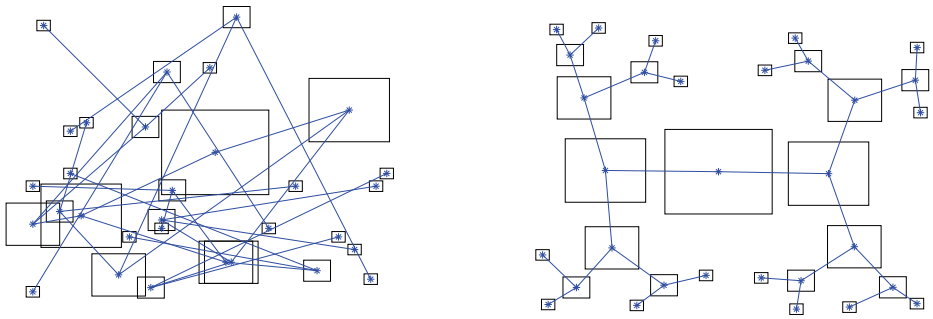

FIG. 4.8. *An example of the 2D layout of a five-level binary tree with nonequal vertices.*

overlap (as can be seen in Figures 4.7 and 4.8) while moving the nodes more evenly through the domain, especially as finer and finer grids are used.

5. Conclusions and future work. We have presented a linear time multilevel algorithm for solving a correction to the nonlinear minimization problem under planar (in)equality constraints. By introducing a sequence of grids over the domain and a new set of global displacement variables defined at those grid points, we formulated the minimization problem under planar equidensity constraints and solved the resulting system of equations by multigrid techniques. This approach enabled fast collective corrections for the optimization objective components. We believe that this formulation can open a new direction for the development of fast algorithms for efficient space utilization goals. Among many possible motivating applications [2, 8, 9, 10, 7, 12, 5, 13] we focused on the demonstration of the method on the graph visualization problem with efficient space utilization demand.

We recommend this multilevel method as a general practical tool in solving, possibly together with other tools, the nonlinear optimization problem under planar (in)equality constraints.

It should, however, be remembered that what we have described in this paper

is good only for obtaining a *correction* for the original problem given some first approximation. In general, to solve the entire problem, when no initial solution is given, an FMG algorithm should be used. That is, a hierarchy of graphs is constructed for the original graph, obtaining a first solution on the coarsest (smallest) graph (by a direct solver or by an exhaustive search); then this solution is interpolated to finer graphs until a solution for the original graph is obtained. In fact, while solving the coarsest level, one can afford to take *many* solutions of this small graph (see [16]) and pursue them to finer levels, eliminating those which are less suitable while proceeding to finer and finer levels. This is still efficient enough while enabling a better sampling of the minimization landscape and allowing the comparison of several local minima. Such a hierarchy was introduced in our work on graph optimization problems [16] and recently in our more sophisticated coarsening scheme [15].

Additional techniques should be added for dealing with too much overlap. In such cases the algorithm will fail to separate the overlapping nodes, as it will apply almost the same movement for all of them. A relaxation that would move each node *individually* is thus needed. Also, to enhance the accuracy of the equidensity constraints (3.2), a more precise estimation for the amount transferred in between the squares might be needed. This can be achieved by calculating the change in each square's area as a function of each node's movement by actually applying a small movement to each node and measuring the resulting influence on the relevant squares. Preliminary results show that this is indeed more accurate and that it is mostly needed when the squares are small compared with the nodes' sizes, and that when the squares are big and contain many nodes this is not really necessary.

Acknowledgment. We would like to thank two anonymous referees, whose suggestions helped to improve this article significantly.

REFERENCES

[1] M. Avriel, *Nonlinear Programming: Analysis and Methods*, Dover, Mineola, NY, 2003.
[2] G. Di Battista, P. Eades, R. Tamassia, and I. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice–Hall, Upper Saddle River, NJ, 1998.
[3] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp., 31 (1977), pp. 333–390.
[4] A. Brandt and D. Ron, *Multigrid solvers and multilevel optimization strategies*, in Multilevel Optimization in VLSICAD, J. Cong and J. R. Shinnerl, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 2003, pp. 1–69.
[5] M. Cardei and J. Wu, *Energy-efficient coverage problems in wireless ad-hoc sensor networks*, Comput. Commun., 29 (2006), pp. 413–420.
[6] T. F. Chan, J. Cong, and K. Sze, *Multilevel generalized force-directed method for circuit placement*, in Proceedings of the International Symposium on Physical Design (ISPD 2005), San Francisco, CA, 2005, pp. 185–192.
[7] Z. Drezner, *Facility Location: A Survey of Applications and Methods*, Springer, New York, 1995.
[8] P. A. Eades, *A heuristic for graph drawing*, Congr. Numer., 42 (1984), pp. 149–160.
[9] D. Harel, *On visual formalisms*, Comm. ACM, 31 (1988), pp. 514–530.
[10] A. Inger, *On the Aesthetic Layout of Higraphs*, Ph.D. thesis, The Weizmann Institute of Science, Rehovot, Israel, 2008.
[11] J. Marks, ed., *Graph Drawing*, Lecture Notes in Comput. Sci. 1984, Springer, Berlin, 2001.
[12] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. B. Srivastava, *Coverage problems in wireless ad-hoc sensor networks*, in Proceedings of IEEE INFOCOM 2001, Vol. 3, Anchorage, AK, 2001, pp. 1380–1387.
[13] G.-J. Nam and J. Cong, *Modern Circuit Placement*, Springer, New York, 2007.
[14] Peko Benchmark, http://cadlab.cs.ucla.edu/~pubbench/peko.htm (2002/2003).
[15] D. Ron, I. Safro, and A. Brandt, *Relaxation-Based Coarsening and Multiscale Graph Orga-*

nization, Technical report ANL/MCS-P1696-1009, Argonne National Laboratory, Argonne, IL, 2010.

[16] I. Safro, D. Ron, and A. Brandt, *Multilevel algorithms for linear ordering problems*, ACM J. Exp. Algorithmics, 13 (2008), pp. 1.4–1.20.

[17] U. Trottenberg, C. W. Oosterlee, and A. Schuller, *Multigrid*, Academic Press, Orlando, FL, 2001.