מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE

Thesis for the degree
Doctor of Philosophy

חבור לשם קבלת התואר
דוקטור לפילוסופיה

By
Ilya Safro

מאת
איליה ספרו

אלגוריתמים רב-סריגים לבעיות אופטימיזציה קומבינטורית
Multilevel algorithms for combinatorial optimization problems

Published papers format

Advisors
Prof. Achi Brandt
Dr. Dorit Ron

מנחה
פרופ' אחי ברנדט
דר' דורית רון

30.06.2007

י"ד בתמוז תשס"ז

Submitted to the Scientific Council of the
Weizmann Institute of Science
Rehovot, Israel

מוגש למועצה המדעית של
מכון ויצמן למדע
רחובות, ישראל

Abstract

The Multiscale method is a class of algorithmic techniques for solving efficiently and effectively large-scale computational and optimization problems. This method was originally invented for solving elliptic partial differential equations and up to now it represents the most effective class of numerical algorithms for them. During the last two decades, there were many successful attempts to adapt the multiscale method for combinatorial optimization problems. Whereas the variety of continuous systems' multiscale algorithms turned into a separate field of applied mathematics, for combinatorial optimization problems they still have not reached an advanced stage of development, consisting in practice of a very limited number of multiscale techniques. The main goal of this dissertation is to extend the knowledge of multiscale techniques for the combinatorial optimization problems.

In the first part of this dissertation we formulate the principles of designing the multilevel algorithms for combinatorial optimization problems defined on a simple graph (or matrix) model. We present the results for a variety of linear ordering functionals (minimum linear arrangement/2-sum/bandwidth/workbound/wavefront sum). Since our algorithms were developed for practical purposes we compared them to many different heuristics: Spectral Sequencing, Optimally Oriented Decomposition Tree, Multilevel based, Simulated Annealing, Genetic Hillclimbing and other (including their combinations). In almost all cases we observed significant improvement of previous state-of-the-art results.

In the second part of this research we present a multiscale coarsening scheme for minimizing a quadratic objective functional under planar inequality constraints. The scheme is demonstrated on a graph drawing problem in which the economical space utilization demand is evolved over the desired area rather than the widely used force-directed method, which preserves the non-overlapping property of the graph vertices. The non-overlapping property is automatically almost preserved as a result of equidensity constraints defined over the entire area. This demonstrates the ability of the algorithm to be used for solving a famous VLSI placement problem.

Acknowledgements

First and foremost I would like to thank my advisors Achi Brandt and Dorit Ron who have guided this research and acquainted me with multigrid methods. It is difficult to overstate my gratitude to Dorit Ron for being a teacher, a collegue and a friend. With her scientific enthusiasm, inspiration, and her ability to explain things clearly, essentially and deeply at the same time, she helped to mould my research skills (not to mention my writing skills). I would like to express my deepest appreciation to Achi Brandt whose ideas and broad view significantly helped in finding and exploring new research directions. There is no doubt that Achi's scientific vision, his experience and his intuition have had an extremely important impact both on my scientific taste and on my approach to research.

I would like to thank the faculty members at the Weizmann Institute, together with the administration and the students, for a creation and maintenance of the unique atmosphere, in which an educational process becomes a real pleasure. During all years at the Weizmann Institute I was surrounded by knowledgeable and friendly people who helped me daily.

Most of all, I would like to thank my parents Luba and Mark, my wife Maya and my brother Daniel, for their love, patience and absolute confidence in me; my dog Uma for her permanent support under the table during even non-conventional hours of work and parrot Fourier, for trying to be quiet.

Contents

CHAPTER 1

Introduction

1.1 Preface

Combinatorial optimization [41] is a wide class of problems, the central goal of which can be usually formulated as finding the minimum or maximum of a function under certain constraints, defined on some finite domain. Many of these problems (which are of great theoretical and practical interest) are known to be NP-hard [49]. Moreover, the status of some problems is still unknown, e.g., when a function or constraint becomes too complicated (as in the case of VLSI design [31], isomorphism problems and several other well known tasks).

In this work we concentrate on developing linear time algorithms for large-scale "real world" instances. Although the question "how to define what is a real world object?" is beyond the scope of this work, it is well known that in many practical problems the structure of such objects usually differs from the ones generated randomly. Most of combinatorial optimization problems (COP) are motivated by various engineering and applied science questions. Naturally, the instances for such problems can be extremely large, thus, even if some quadratic time approximation algorithm produces the solution with a rather optimistic rigorously proven approximation ratio, in practice, it is still inapplicable.

It is more than arduous scientific task to generalize the definition of a "real world" instance. Quite often it turns to be less complicated as we significantly constrict the set of instances or problems. However, it is believable that many "real world" instances are well structured or, in other words, possess some geometry. Examples of huge classes of various well-organized objects are: finite element structures, power low distributions, trees, etc., to mention just a few.

In this work we propose various strategies for designing linear time heuristic algorithms for large-scale COP. These strategies will be demonstrated on some problems which belong to the set of one- and two-dimensional layout problems and, hence, will be empirically evaluated on "real world" benchmarks.

In many problems it is often noticeable that, notwithstanding the fact that elementary parts of the system have a very complicated (or even non-deterministic) behavior, their ensembles represent much more structured systems. The *multiscale algorithms* (MA) [14] successfully exploit these facts demonstrating a high quality performance on practical data. The multiscale computational methodology is a systematic approach to achieve efficient calculations of systems containing many degrees of freedom (like graph vertices, image pixels, discretized differential equations, particles, etc.). From the relationships between the given microscopic parts of the system (like graph, image, physical model, system of equations, etc.), the rules for the system at increasingly larger scales are derived. The idea behind every MA is to collect the relevant information regarding the system at different scales and then to obtain the solution at microscopic scale by adapting the information inherited at larger scales. Firstly, these algorithms were developed for continuous systems and their discretizations. However, during the last two decades, there were many successful attempts to adapt them for discrete optimization problems. Whereas the variety of continuous systems' MA turned into a separate field of applied mathematics, MA for COP still have not reached an advanced stage of development, consisting in practice of a very limited number of multiscale techniques. The main goal of this dissertation is to extend the knowledge of multiscale techniques for COP.

Overview of results

In the first part of this research (Chapters 2, 3 and 4) we formulate the principles of designing MA for COP defined on a simple graph model. We present numerical results for a variety of linear ordering functions (minimum p-sum, minimum bandwidth, minimum workbound and minimum wavefront sum, see [39]). The common heuristic principle is based on the idea that during coarsening each vertex may be associated to more than just one aggregate according to some "likelihood" measure rather than the usually used strict coarsening, where each coarse vertex is accumulated from small subsets of fine vertices. The uncoarsening initialization, which produces the first arrangement of the fine graph nodes, strongly relies on energy considerations (unlike usual interpolation in classical Algebraic Multigrid). This initial order is improved further by local strict minimization relaxation and possibly by employing stochasticity.

In addition, we propose two general principles that can be used for different linear ordering functionals: (1) the continuation approach in which functionals that contain an evaluation of power p are successively approximated by a sequence of similar functionals but with lower power; (2) a first approximation can be obtained from the arrangement produced by one V-cycle of the minimum 2-sum problem instead of using the popular spectral approach.

Since our algorithms were developed for practical purposes we compared them to many different heuristics : Spectral Sequencing [7, 34, 35, 50], Optimally Oriented Decomposition Tree [6], Multilevel based [68, 59], Simulated Annealing [79, 77, 78], Genetic Hillclimbing [81] and other (including their combinations). In almost all cases we observed significant improvement of the results. In particular, the results of the minimum 2-sum, the minimum bandwidth and the workbound problems were improved on the average by about 30%-40%, while the running time on graphs with up to 10^5 edges is less than one minute on a simple 1.7GHz PC. Our algorithms have proven themselves to be very stable (i.e., small standard deviations) and of high quality both as a first approximation and as more aggressive energy minimizers.

In the second part of this research (Chapter 5) we present a multiscale coarsening scheme for minimizing a quadratic objective functional under planar inequality constraints. The scheme is demonstrated on a graph drawing problem in which the economical space utilization demand is evolved over the desired area rather than the widely used force-directed method, which preserves the non-overlapping property of the graph vertices. In its current preliminary version it is only designed to provide a correction to a given solution, rather than solving the entire problem.

The running time of all algorithms is linear, thus it can be applied to very large systems. The implemented algorithms can be obtained at [90].

Thesis structure

This thesis begins with a brief background on classical geometric and algebraic multiscale algorithms for linear and nonlinear systems (Chapter 1.2). The journal articles on which this thesis is based on are presented as separate chapters.

1. Chapter 2: I. Safro, D. Ron and A. Brandt, "Graph minimum linear arrangement by multilevel weighted edge contractions", *Journal of Algorithms*, vol. 60/1, pp. 24–41, 2006.

2. Chapter 3: I. Safro, D. Ron and A. Brandt, "A multilevel algorithm for the minimum 2-sum problem", *Journal of graph algorithms and applications*, vol. 10/2, 2006.

3. Chapter 4: I. Safro, D. Ron and A. Brandt, "Multilevel algorithms for linear ordering problems", extended version Technical Report MCS07-03, Computer Science and Applied Mathematics, Weizmann Institute of Science (submitted to the *Journal of experimental algorithmics*).

4. Chapter 5: D. Ron, I. Safro and A. Brandt, "Fast multilevel solver for a 2-dimensional graph layout improvement", In preparation.

The lists of references of all papers are unified into one list that appears at the end of the thesis.

1.2 Multigrid methods background

The following very brief survey on multiscale methods is intended for a reader who needs either to refresh or to be introduced to the basic components and ideas of these methods. Much more advanced explanations, rigorous analysis and references can be found in the detailed surveys [14, 17, 107] and in the textbooks [22, 103].

1.2.1 History and intuition

The *Multiscale method*[1] is a class of algorithmic techniques for solving efficiently and efficiently large-scale computational and optimization problems. Any multivariable problem defined in some space can have an approximate description at any given length scale of that space: a continuum problem can be discretized at any given resolution; multiparticle system can be represented at any given characteristic length; etc. The multiscale algorithm recursively constructs a sequence of such descriptions at increasingly larger (coarser) scales, and combines local processing (relaxation) at each scale with various inter-scale interactions. Typically, the evolving solution on each scale recursively dictates the equations on coarser scales while supplying large-scale corrections to the solutions on finer scales. In this way, large-scale changes are effectively calculated on coarse grids, based on information previously gathered from finer grids. Various fundamental computational problems in different disciplines (physics, chemistry, engineering, etc.) use the ideas of multiscale methods.

Multigrid methods were originally introduced for solving elliptic partial differential equations (PDE) and up to now they represent the most effective class of numerical algorithms for them. The idea of multigrid methods was formulated by Fedorenko and Bakhvalov [45, 46, 4] in the '60s. However, the real power of these methods was recognized only in the early '70s by Brandt in his pioneering research [10, 11]. During the last three decades, multigrid methods were adapted and generalized for many computational tasks in various disciplines. Multigrid methods are known to be very well scalable and efficient since they can solve a system with only linear time and space complexity. Moreover, the nature of these algorithms allows relatively easy distribution of the main parts of the task among parallel machines, what makes these methods ideal for solving large-scale computational problems.

Before we turn to the description of the basic multigrid components, let us describe the general idea and intuition which lie behind these methods. Consider a classical elliptic PDE problem for which the first multigrid algorithms were successfully developed. The PDE is approximated (discretized) by a linear system of equations when its physical domain Ω is approximated (or discretized) by a mesh.

[1]The multiscale method got several additional names like *multigrid* and *multilevel*. Usually we will mention the most historically suitable name, as for example in case of a) the general discussion - the most appropriate term will be "mutiscale"; b) the classical optimization algorithms - "multigrid" and c) COP - "multilevel".

Thus, every mesh over Ω defines a linear system

$$Ax = b \ , \tag{1.1}$$

where $A \in \mathbb{R}^{n\times n}$ is symmetric positive definite and $x, b \in \mathbb{R}^n$. Let $\tilde{x}$ be an approximate solution of (1.1) and denote by $e = x - \tilde{x}$ the *error* vector and by $r = b - A\tilde{x} = Ae$ the vector of *residuals*. We use $||\cdot||$ for the l_2 norm and $||\cdot||_{/A}$ for the A-normalized l_2 norm , i.e.,

$$||e||^2 = \sum_{i=1}^{n} e_i^2 \ , \qquad ||r||^2_{/A} = \sum_{i=1}^{n} \frac{r_i^2}{\sum_{\lambda=1}^{n} a_{i\lambda}^2} \ . \tag{1.2}$$

The *direct methods* for solving such linear systems compute exact solution, but the price paid in work and storage can be prohibitive for very large linear systems. In contrast to the direct methods that attempt to solve the problem in one-shot (like solving a linear system of equations 1.1 by finding the inverse of the matrix A), *iterative methods* begin with an initial estimation for the solution and successively improve it until the solution is accurate enough. However, in order to achieve the high-quality approximation (which is very close to the exact solution), the number of iterations in these methods grows up together with the total complexity and thus become a quite ineffective tool for large-scale data (in particular, when there are no special restrictions on the problem formulation). These iterative methods are also called *relaxation* and they play a key role in the multigrid methodology. We will first describe the classical relaxation algorithms and then explain how they demonstrate the idea of multigrid.

Relaxation

Relaxation is a pointwise iterative[2] method for solving (1.1), each iteration has the form

$$\tilde{x}^{(k+1)} = T\tilde{x}^{(k)} + v, \tag{1.3}$$

where the matrix T and the vector v are chosen so that the fixed point of the equation $x = Tx + v$ is the solution to (1.1). Such a method is said to be *stationary* if T and v are constant over all iterations. One way to obtain a suitable matrix T is by *splitting*, in which A is written as $A = M - N$, where M is nonsingular. Taking $T = M^{-1}N$ and $v = M^{-1}b$, the generalized iteration will be

$$\tilde{x}^{(k+1)} = M^{-1}N\tilde{x}^{(k)} + M^{-1}b. \tag{1.4}$$

We will briefly describe the most relevant (for this work) traditional stationary schemes: Jacobi and Gauss-Seidel.

[2]The iteration number will be denoted by a superscript in parentheses.

- **Jacobi method**. The simplest choice for M in the matrix splitting is a diagonal matrix, specifically the diagonal of A. Denote by D the matrix which contains only zeros besides its diagonal copied from A. Therefore, fix $M = D$ and $N = -(L+U)$, where U and L are strict upper- and lower-triangular submatrices of A, respectively. If A has no zero diagonal entries, so that D is nonsingular, then we obtain the iterative scheme known as the Jacobi method:

  $$\tilde{x}^{(k+1)} = D^{-1}(b - (L+U)\tilde{x}^{(k)}) \quad . \tag{1.5}$$

  Thus, the pointwise $\tilde{x}$ correction will be the following:

  $$\tilde{x}_i^{(k+1)} = \frac{b_i - \sum_{j \neq i} a_{ij}\tilde{x}_j^{(k)}}{a_{ii}} \quad . \tag{1.6}$$

  The convergence speed of this method is usually slow and it requires double memory for $\tilde{x}$ to keep the last two iterations. On the other hand, Jacobi is preferable in applications when each entry of $\tilde{x}^{(k+1)}$ must be based on the entries of the previous iteration only.

- **Gauss-Seidel method**. One reason for the Jacobi slow convergence is that it does not use the latest information available, i.e., new entries are involved only after the entire sweep has been completed. The *Gauss-Seidel* method remedies this drawback by taking each new component of the solution immediately after its update. Let the splitting be $M = D + L$ and $N = -U$. Thus, the corresponding iteration and pointwise formulas will be

  $$\tilde{x}^{(k+1)} = D^{-1}(b - L\tilde{x}^{(k+1)} - U\tilde{x}^{(k)}) \quad \text{and} \tag{1.7}$$

  $$\tilde{x}_i^{(k+1)} = \frac{b_i - \sum_{j<i} a_{ij}\tilde{x}_j^{(k+1)} - \sum_{j>i} a_{ij}\tilde{x}_j^{(k)}}{a_{ii}} \quad . \tag{1.8}$$

  The updating of the unknowns must be done successively, in contrast to the Jacobi method, in which the unknowns can be updated in any order. In the view of the possible parallelization of the algorithm, this is an undoubtless advantage of the Jacobi relaxation. Besides of the faster convergence, the Gauss-Seidel method wastes less storage space than Jacobi.

All these stationary iterative methods converge $\tilde{x}^{(k)}$ to the solution $x = A^{-1}b$ if and only if the spectral radius $\rho(M^{-1}N) < 1$.

It is important to remark that: (1) a common feature of the mentioned and many other types of relaxation is that at each step some corrections to $\tilde{x}$ are calculated based on a small number of unknowns, i.e., the relaxation is a local process, which basically averages out each unknown with the values of its neighboring unknowns, e.g., (1.6) and (2) after a small number of relaxation sweeps on system (1.1), the remained error becomes smooth.

The basic observation

The basic intuition for the multigrid methods is concerned with the meaning of the relaxation. In fact, the relaxation is a process of averaging, in which the highly-oscillatory error modes are removed. A one-dimensional example of a simple relaxation (whose basic step is $x_i = (x_{i-1} + x_{i+1})/2$ which is the discretization of the 1-dimensional Poisson equation, i.e., this is a special case of Jacobi relaxation with $b_i = 0$) is shown in Figure 1.1. The initial data may contain many highly-oscillatory components that disappear while applying more and more relaxation sweeps. In other words, the relaxation is actually a "smoother" which reduces the error components of increasingly larger scales as the averaging process proceeds (see the results after 5,10 and 500 sweeps in Figure 1.1): the highly-oscillatory components are removed very fast during the first sweeps, while the smooth error modes are be removed very slow. Thus, after a small number of relaxation sweeps, there is actually no need to describe the system at the same resolution (or discretization) as has been done before the relaxation, in fact, the relaxed data can be effectively characterized by a coarser resolution with fewer variables.

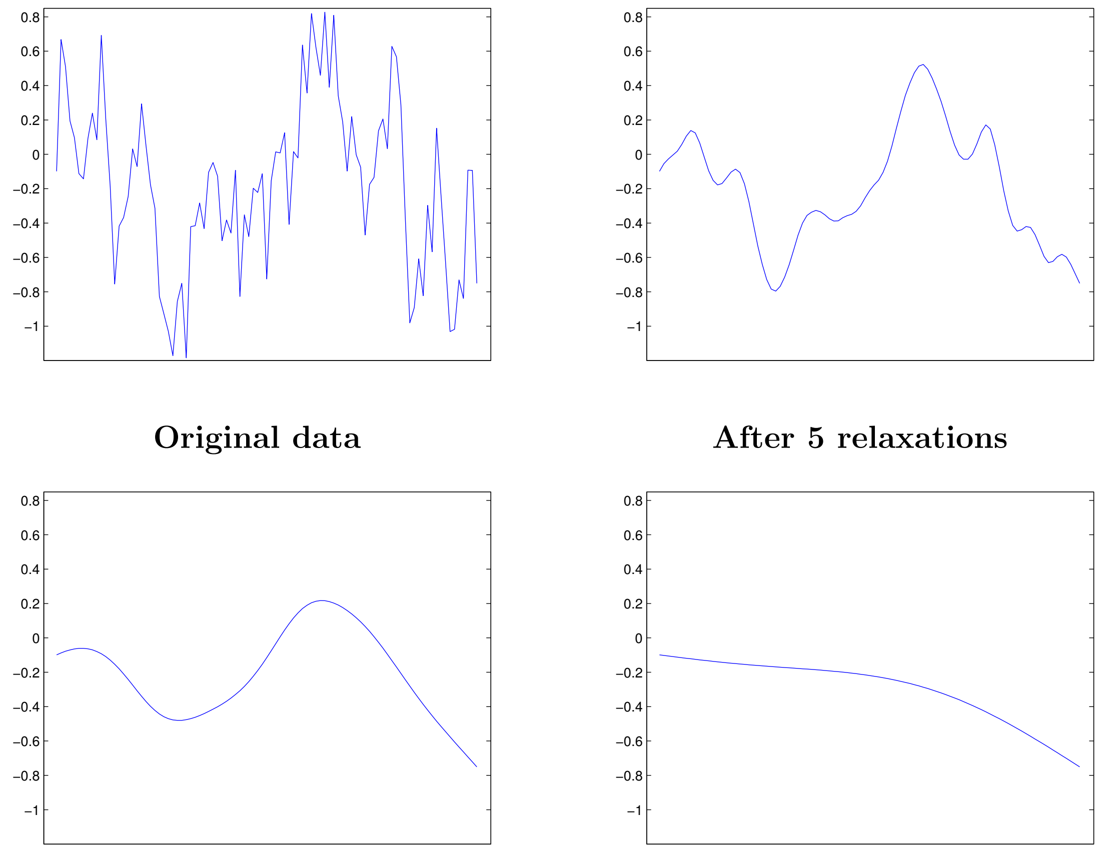
Conclusion: *A suitable relaxation can always reduce the information content of the error (by smoothing it), and quickly make it approximable by far fewer variables (which are related to the smooth error modes).*

The above conclusion is very general and holds even for quite general nonlinear systems. As a result of this conclusion, the following natural question can be asked: when should we stop the relaxation and continue by describing the system with fewer variables without loosing much information? The answer to this question is based on the fact that the convergence of the relaxation must be slow when the individual residuals do not show the true magnitude of error, i.e., when $||r||_{/A} \ll ||e||$. The converse is also true: if the convergence of a *suitable* relaxation is slow, then $|r||_{/A} = ||Ae||_{/A} \ll ||e||$ must hold. Since the deeper the condition $||Ae||_{/A} \ll ||e||$ is satisfied the more special must be the type of the error $||e||$. In practice, it is enough to perform up to 3 sweeps of relaxation in order to smooth the system.

1.2.2 The hierarchy of coarser problems

Following a small number of relaxation sweeps, the remaining error e, and hence also the solution $\tilde{x}$ itself can be approximated by a coarser system with fewer new variables $\{x_i^c\}_{i=1}^m$. A first issue in any coarsening scheme, whether for linear or nonlinear system, is how to define the coarse variables. During this work two coarsening schemes will be mentioned: "geometric multigrid" and "algebraic multigrid".

Geometric multigrid (GMG). In the classical case of this scheme, where the fine-level set x is defined on a well-structured grid, the coarse set x^c is naturally defined in terms of coarsening that grid, for example by omitting from it any other matrix row and column as in Figure 1.2. This scheme is used in Chapter 5.



Original data **After 5 relaxations**

After 10 relaxations **After 500 relaxations**

Figure 1.1: An example of a simple relaxation process.

Algebraic multigrid (AMG). Unlike the geometric multigrid coarsening (in which the coarse variables are predefined at all scales regardless of the problem input), AMG does not have a predefined set of coarse variables. The set of coarse variables is chosen as a subset of the set of fine variables V; or, more generally, each coarse variable is compounded of fractions of a small number of fine variables. This aggregation process is performed around each fine variable (which entirely contributes itself to the respective aggregate) chosen into the set C. This set must be constructed so that all other fine variables in $F = V \setminus C$ will be "strongly coupled" to C (as explained in Section 1.2.3).

Coarse equations. There are several ways to construct the coarse equations which describe the relationship between the coarse variables. An inexpensive direct derivation is available in GMG, in which the same structure of the original equations is preserved. Relatively inexpensive algebraic derivations of coarse equations are obtained via the definition of an explicit coarse-to-fine *interpolation operator* $\uparrow_c^f$ such that $x \approx \uparrow_c^f x^c$. More details are supplied in Section 1.2.3.

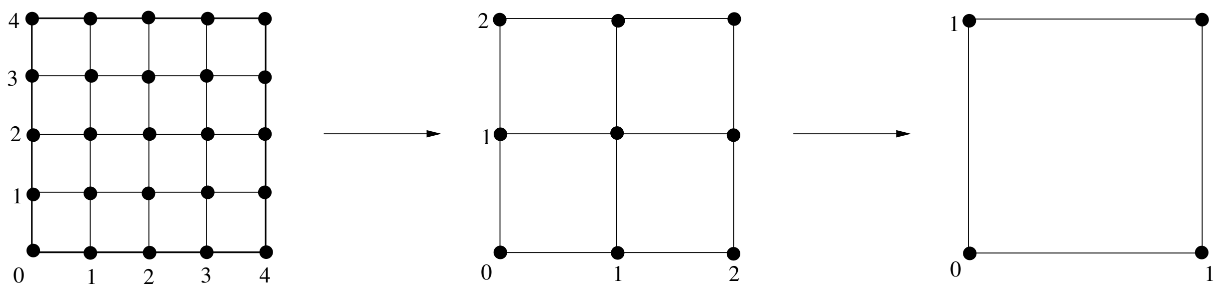


Figure 1.2: Standard geometric coarsening scheme.

Multilevel cycles. Having constructed the coarse-level variables and equations, they are then solved by a similar procedure: a small number of relaxation iterations followed by approximating the remaining error with a still coarser system. This recursively defines the multilevel cycle, which time and space complexity is comparable to that of just a small number of relaxation sweeps over the finest level. A basic rule that every part of the multiscale algorithm (interpolation operator, relaxation, etc.) must satisfy is *stationarity*, meaning that if $\tilde{x}$ is already at the desired minimum (in a case of optimization problem), the algorithm will never move away from it.

Multiscale interpretation

The behavior of any physical system can be interpreted differently while one observes it at different scales. Not rarely it is rather difficult to understand the general behavior of a system when only its elementary parts and their relationships are taken into account. At the same time, observing the ensembles of these elementary parts can make comprehension of the global picture a much easier task. For example, consider a digital representation of an image as a matrix of pixels. Staring concentratively many times at specific image regions (such that separate pixels would be distinguishable), it is almost impossible to understand the image in general. However, it is enough to look at the entire image (without special attention to separate pixels) and the general picture will be evident. Such a division of our attention into different scales is the main postulate of the multigrid methods idea. It can be presented in the following algorithmic structure. For the solution of a problem P, we define a hierarchical set of problems $P = P_0, P_1, ..., P_K$ where P_i is in some sense a coarser approximation of P_{i-1} in the range $1 \leq i \leq K$. The solution Π_i to P_i is closely related to the solution Π_{i-1} of P_{i-1}, and moreover, it is easier to solve P_i than P_{i-1}. The basic strategy is to start with finding a solution Π_K for P_K and then, level by level, construct Π_{i-1} from Π_i for each i.

The relaxation is also a most important computational part of the task. During each call, this local process will affect only a very limited number of current scale elementary components (like pixels, particles, graph vertices, etc.). At each scale the relaxation will be responsible for smoothing the respective finer scale error.

The main parts of a multigrid algorithm (MA) can be summarized as follows:

- **A**. Define the notion of elementary (or microscopic) component of the problem and the set of relations among them;
- **B**. Define the next (coarse) scale problem;
- **C**. Define an appropriate relaxation process which reduces the error of some approximate solution;
- **D**. Define a method to transfer information between coarse and fine scales.

Let us see how these tasks can be joined to construct the basic multigrid algorithms for quadratic optimization which is related to our COP problems. The classical multilevel approach is described bellow, while the *new* particular algorithmic ingredients for solving the COP will be presented in 1.3.

1.2.3 Correction scheme cycle

The general unconstrained quadratic optimization problem can be formulated as follows. Assume $A \in \mathbb{R}^{n\times n}$ and $x, b \in \mathbb{R}^n$. Our goal is to minimize the quadratic form

$$\frac{1}{2}x^T Ax - b^T x \quad , \tag{1.9}$$

where A is symmetric positive definite (however, this condition is not obligatory in more complicated cases). The minimizing vector x satisfies the same exact linear system as in (1.1). Thus, our discussion below will focus on fast multiscale solvers for (1.1) with the same notation.

Having the problem (1.9), the above demand **A** is accomplished – the elementary components of the problem are defined as $\{x_i\}_{i=1}^n$ and the (i,j)-th entry a_{ij} is the corresponding relationship between x_i and x_j.

Usually, while constructing the hierarchy of the coarse scales, the MA involves at different scales similarly defined (or very closely related) problems. Thus, the natural way to introduce the next coarse scale problem is to reduce the dimensionality of the original problem. The superscript index c in $A^c, x^c, \tilde{x}^c, e^c, r^c$ will refer to the similarly defined coarse scale matrix and vectors (as demanded in **B**). In other words, the coarse scale will be represented by lower dimensional matrix $A \in \mathbb{R}^{m\times m}$ and corresponding vectors in $\mathbb{R}^m$, where m is a fraction of n, typically $n/8 \le m \le n/2$. The information between the coarse and the fine scales (see demand **D**) will be transferred by interpolation and restriction operators denoted by $\uparrow_c^f\colon \mathbb{R}^m \longrightarrow \mathbb{R}^n$ and $\uparrow_f^c\colon \mathbb{R}^n \longrightarrow \mathbb{R}^m$, respectively. By transferring the information from fine to coarse scales we usually mean the creation of a new problem, while the backward process will project the solution from the coarse scale to the fine ones.

In the first part of our studies (Chapters 2, 3 and 4) the notion of microscopic component refers to the graph vertex and the relationship between two components is defined by the corresponding weighted edge. The elementary component of the second part's system (Chapter 5) will be a variable which determines the movement needed for a small subset of vertices for solving a correction to the two-dimensional graph layout problem.

As it was observed, the key role in multigrid methods is intended for the relaxation process (mentioned as demand **C**), or, more precisely for the interplay among the relaxation, the interpolation and the restriction operators. Using the terms of fine and coarse scales, the explanation of relaxation follows. At each scale it must decrease high frequency error obtained before and after transferring the information between scales. The error which is not efficiently reduced by this smoothing is typically well approximated by a next coarser system. The error on the coarse system, which is smooth on the fine level, turns out to contain again high frequency error with respect to the coarser system. The remained error can be approximated by a coarser system with much smaller number of variables while on the coarsest grid a direct solver can be applied.

We will demonstrate here one basic MA called *correction scheme.* In terms of solving (1.1) the algorithm can be summarized as follows:

```
Multigrid($A$, $\tilde{x}$, $b$)
  If $A$ is small enough
    Solve $A\tilde{x} = b$ directly
  Else
    Apply $\nu_1$ relaxation (error smoothing) sweeps on $A\tilde{x} = b$
    Define $A^c$, $e^c = 0$ and $r^c = \uparrow_f^c r$
    Multigrid($A^c$, $e^c$, $r^c$)
    Correct $\tilde{x} = \tilde{x} + \uparrow_c^f e^c$
    Apply $\nu_2$ relaxation (error smoothing) sweeps on $A\tilde{x} = b$
  Return $\tilde{x}$
```

The $\nu = \nu_1 + \nu_2$ sweeps performed in this algorithm on any of its scales are expected to reduce the corresponding error components (those visible on that grid but not on the coarser level) by the factor $\overline{\mu}^{\nu}$, where $\overline{\mu}$ is the smoothing factor, which for solving some simple PDEs can be rigorously calculated by Fourier analysis [103, 19]. Since all grids are so traversed, the cycle should heuristically reduce all error components at least by the factor $\overline{\mu}^{\nu}$. Thus $\overline{\mu}$ can serve as a practical predictor of the multigrid performance one should be able to obtain. The number of relaxation sweeps ν is also an indicator of the cycle complexity.

Coarse variables. When the geometry of the problem is known we can choose a coarser grid by eliminating points in a geometrically-regular pattern as is presented in Figure 1.2. Let us see in more details the AMG case for which we refer in Chapters 2, 3 and 4.

The procedure for choosing C, the set of coarse variables, contains only two sweeps through all variables: (0) decide if A has weak connections (a_{ij}) and possibly remove them (1) pass through all fine variables transferring to C those variables that still are not "strongly connected" to already chosen. The strength of the connection between F-points and their seeds is defined by the parameter $Q \in (0,1]$. In particular, $i \in F$ is "strongly connected" to C if

$$\frac{\sum_{j\in C} a_{ij}}{\sum_{j\in V} a_{ij}} \geq Q \quad . \tag{1.10}$$

and, therefore, should not be added to C.

Coarse equations. There are several ways to construct approximate coarse equations. Non interpolation-based derivation is usually available in GMG. As it was shown in the correction scheme algorithm, to approximate the error $e = x - \tilde{x}$ left after several relaxations, its equation $Ae = r$ is defined on the coarse level by $A^c e^c = r^c$. The right-hand side of which is calculated by $r^c = \uparrow_f^c r$, where $\uparrow_f^c$ is a fine-to-coarse transfer operator, called *restriction.* $\uparrow_f^c r$ is a coarse-grid function whose value at each point is typically a *weighted average* of values of r at neighboring fine-grid points. The matrix A^c can be obtained in a similar discretization of the same problem on a grid with larger mesh size.

The correction to $\tilde{x}$ is obtained by

$$\tilde{x} = \tilde{x} + \uparrow_c^f e^c \quad , \tag{1.11}$$

where $\uparrow_c^f$ is a coarse-to-fine interpolation matrix. At each fine-grid point the value of $\uparrow_c^f e^c$ is interpolated from values e^c at neighboring coarse-grid points. Linear interpolation can be used in many cases.

One of the most traditional approaches for derivation of the coarse equations A_c in AMG is the Galerkin operator

$$A_c = \uparrow_f^c A \uparrow_c^f \quad , \tag{1.12}$$

which projects the fine system of equations to the coarser scale. Usually, for symmetric and positive definite matrices A the restriction mapping is defined as the transpose of the interpolation $\uparrow_f^c = (\uparrow_c^f)^T$. The (i, J)-th entry of $\uparrow_c^f$ represents the strength of the connection between fine variable i and coarse variable J and $e_i = (\uparrow_c^f e^c)_i$. The entries of $\uparrow_c^f$ are called *interpolation weights* and they describe both the coarse-to-fine and fine-to-coarse variable relations.

AMG principles are widely employed for solving discretized PDEs on both structured and unstructured grids, in image processing, clustering, various graph algorithms and for many other goals. Its main advantage is that the coarsening process is fully automatic and this is the major reason for AMG flexibility in adapting itself to specific requirements of the problem and robustness despite using very simple pointwise smoothers.

Cycle variations

The scheduling of the multiscale recursive calls and returns can be defined in several forms. The simplest form of the multigrid algorithm is called a V-cycle. Starting at

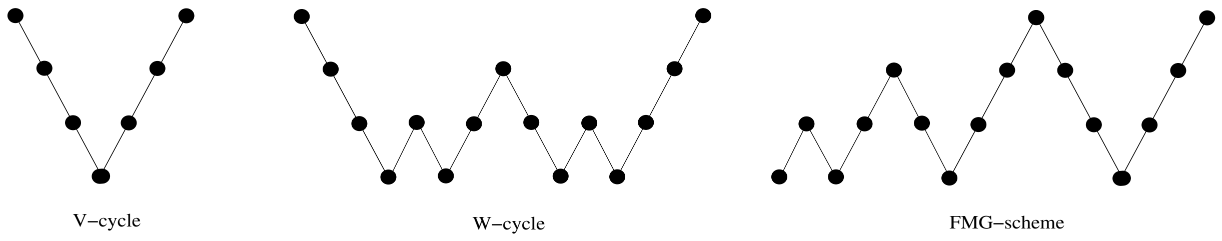


Figure 1.3: Schematic example of different cycle models. In all examples there are five coarse levels. The W-Cycle scheduling has cycle index 2. Each solid point represents the respective coarse level at some time. The leftmost top point is the start of the cycle at the finest level.

the finest grid, a number of recursive calls reduces the problem size and then comes back bringing an approximation from the next coarser level to the finer one (see Figure 1.3, V-cycle).

A more advanced form is called a *W-cycle*. It has a parameter constant (called *cycle index*), which declares how many times should the process visit the next coarse level before returning to the next finer level. We present the scheme of the W-cycle with cycle index 2 in Figure 1.3, W-cycle. The V-cycle is the special case of the W-cycle with cycle index 1.

The cycle forms described above can be applied to any first approximation given on the finest grid. Another way is to obtain the first approximation by interpolation from the solution of the next coarser grid, which has been previously calculated by a similar algorithm. The last strategy is called a *Full multigrid cycle* (FMG). A typical FMG algorithm, with one V-cycle per refinement, is shown in Figure 1.3, FMG-scheme.

1.2.4 Nonlinear systems

The most popular algorithm for nonlinear systems is called the *Full Approximation Scheme* (FAS). Denote by $N(x)$ the nonlinear fine level system

$$N(x) = b \tag{1.13}$$

and by $N^c(x^c)$ the corresponding coarse-level nonlinear system. In particular, if the system $N(x) = b$ represents a discretization of some continuum equations, then a similar discretization can be obtained for the coarse grid. If $\tilde{x}$ is the current fine-level approximation, and hence $r = b - N(\tilde{x})$ is the current residual, then a general coarsening scheme is to approximate the fine-level residual equations

$$N(x) - N(\tilde{x}) = r \tag{1.14}$$

by the nonlinear coarse system

$$N^c(x^c) - N^c(\tilde{x}^c) = \uparrow_f^c r \quad , \tag{1.15}$$

where $\uparrow_f^c$ is a fine-to-coarse transfer as defined previously for linear systems and $\tilde{x}^c = \uparrow_f^c \tilde{x}$. In this case the error is still smoothed by the relaxation process and thus approximated by the residual equations, so after obtaining a solution x^c to (1.15), it is the correction $x^c - \tilde{x}^c$ which should be interpolated, i.e., the coarse grid correction to the fine level is

$$\tilde{x}_{\text{new}} = \tilde{x} + \uparrow_c^f (x^c - \tilde{x}^c). \tag{1.16}$$

This scheme, introduced by Brandt in [11], is called Full Approximation Scheme, because it gives directly an equation in terms of the *full* solution x^c and not in terms of the correction e^c. Defining

$$\tau^c = N^c(\tilde{x}^c) - \uparrow_f^c N(\tilde{x}) \quad , \tag{1.17}$$

Eq. (1.15) can be written as $N^c(x^c) = \uparrow_f^c b + \tau^c$ and can be used recursively at all levels.

In Chapter 5 we have combined the FAS with the regular geometric multigrid. The problem formulated there represents a quadratic minimization functional under planar linear inequality constraints. These inequality constraints are coarsened by the FAS as a highly non-linear part of the problem.

1.3 New multilevel techniques for graph problems

During the last two decades there were many attempts of employing multilevel strategies for solving combinatorial optimization problems [17, 95, 64, 105, 106, 104, 65, 1]. Since the most frequent branches on which the multilevel algorithms were applied were VLSI design [27, 31, 29, 32, 26], graph optimization problems (with a special attention on the partitioning problem [33, 2, 95, 65, 87, 64, 8, 56, 66, 5, 1]), and several others, the most of the multilevel schemes were developed for a simple graph model. We will describe the most basic components of our multilevel scheme for the linear ordering problems and the main difference between our and previous approaches.

Coarsening

Coarsening variables. Typically, almost all previously developed multilevel schemes for simple graphs possess exactly the same coarsening. It is carried out by matching groups (usually pairs) of vertices together and representing each group with a single vertex in the coarsened space (e.g. matching [64, 105, 106, 104], first choice [31], etc.). The main difference between our approach to most other multilevel approaches (related to various graph optimization problems) is the coarsening scheme. While the previous approaches may be viewed as *strict* aggregation process, our coarsening is based on the AMG principles and it is actually a *weighted* aggregation in which every node may be divided into *fractions*, and different fractions belong to different aggregates. This enables more freedom in solving the coarser levels and avoids making hardened local decisions, such as edge contractions, before accumulating the relevant global information.

One of the important achievements of our work is that this coarsening is very general and it turns to be suitable for all the different functionals we have tested. This fact can be explained by the way the hierarchy of problems is constructed: variables are eliminated within the coarsening phase only and exactly when they show strong dominant connections to the remaining (non-eliminated) variables, this in turn guarantees that the solution of the eliminated variables is naturally obtained once the non-eliminated variables are solved.

Coarse equations. In our linear ordering algorithms we have used the, so called, *weighted aggregation* scheme introduced in [97] for image segmentation problems. The classical AMG interpolation matrix $\uparrow_c^f$ (of size $|V| \times |C|$) is defined by

$$(\uparrow_c^f)_{iJ} = \begin{cases} w_{ij}/\big(\sum\limits_{k\in N(i)} w_{ik}\big) & \text{for } i \in F,\ j \in N(i) \\ 1 & \text{for } i \in C,\ j = i \\ 0 & \text{otherwise} \end{cases}, \tag{1.18}$$

where $N(i)$ be a reasonably small subset of C-variables strongly connected to i. $(\uparrow_c^f)_{iJ}$ thus represents the *likelihood* of i to belong to the J-th aggregate (coarse node). The edge connecting two coarse aggregates I and J, w_{IJ}^c, is assigned with the weight $w_{IJ}^c = \sum_{k\neq l}(\uparrow_c^f)_{kI} w_{kl} (\uparrow_c^f)_{lJ}$. This coarsening scheme is used in Chapters 2, 3 and 4.

The coarsest level. Solving the appropriate functional at the coarsest level is performed by trying all possible arrangements. Since the amount of work invested at the coarsest levels is negligible compared with that of the finest levels, many solutions can in fact be kept at each level whose graph is relatively small with respect to the finest level graph.

Disaggregation

Initialization. The various algorithms thus differ in the disaggregation process which follows by projecting to a finer level the final arrangement obtained on a coarser level. The initial fine level arrangement is obtained by direct projection of the coarse variables to a fine level and by minimizing the local energy contribution of the remained fine vertices. This initial fine level arrangement is being further improved by applying different local reordering methods.

Compatible and Gauss-Seidel relaxations. *Compatible relaxation* was first developed for stochastic Monte-Carlo processes in [20]. We have used the compatible relaxation in Chapters 2, 3 and 4 as an energy minimizer immediately after the coarse-to-fine interpolation stage. The role of variables in linear ordering problems play the graph vertices' positions along a line. Having a presumably good optimized coarse linear ordering functional, the coarse vertices were fixed invariant during the Gauss-Seidel relaxation (in which *each* node minimizes its contribution to the total energy), while their fine neighbors were compatible updated each time improving their contribution to the global optimization functional. The empirical convergence and the improvement of the respective functional in almost all cases were the additional evidence of the algorithm's correctness.

Minimization and postprocessing. Since we are dealing with a discrete problem, the ordering obtained after relaxations should be improved by some discrete minimization process which "converges" to a local minimum (that is achieved by Gauss-Seidel relaxation in the continuous problems). Besides simple greedy per-vertex local improvements, we have developed a simultaneous minimization of several vertices called Window Minimization. This procedure was adapted for various functionals and plays a crucial role both as a minimizer at all levels of the algorithm and as a *postprocessing* which produces a final solution at the finest level.

Finally, our minimization and postprocessing are intensified by Simulated Annealing which is a general method to escape local minima (described below). In the multilevel framework SA is aimed at searching only for *local* changes that guarantee the preservation of large-scale solution features inherited from coarser levels. Our postprocessing can be used as an integrated part in various algorithmic frameworks (as used for the spectral approach, see Chapter 3).

1.4 Multilevel simulated annealing

A general method to escape false local minima and advance to lower costs is to replace the strict minimization by a process that still accepts each candidate change which lowers the cost, but also assigns a positive probability for accepting a candidate step which increases the cost of the arrangement. The probability assigned to a candidate step is equal to $exp(-\delta/T)$, where $\delta > 0$ measures the *increase* in the arrangement cost and $T > 0$ is a temperature-like control parameter which is gradually decreased towards zero. This process, known as *Simulated Annealing* (SA) [67], in large problems would usually need to apply *very gradual* cooling (decrease of temperatures), making it extremely slow and inefficient for obtaining global optimum.

In the multilevel framework, however, the role of SA is somewhat different. At each level it is assumed that the *global* approximate solution has been inherited from the coarser levels, and thus only *local*, small-scale improvements are needed. We have developed a multilevel simulated annealing tool for the linear ordering problems in which the search space for some improvement at every step is small enough to be very local. For that purpose, we have started at relatively high T, lowered it *substantially* at each subsequent sweep until strict minimization is employed.

Repeated heating and cooling is successively employed for better search over the local landscape. This search can be further enhanced by the introduction of a "memory" like tool consisting of an additional permutation which stores the *Best-So-Far* (BSF) observed arrangement. Henceafter, the BSF is being occasionally updated by the procedure called *Lowest Common Configuration* (LCC) [18] which enables the systematic accumulation of *sub*-permutations into it over a sequence of different arrangements, such that each BSF sub-permutation exhibits the best minimal sub-order encountered so far. The cost of the obtained BSF is at most the lowest cost of all the arrangements it has observed, and usually it is lower. The use of LCC becomes more important for larger graphs, where it is expected that the optimum of a subgraph is only weakly dependent on other subgraphs. The BSF is improved by the LCC procedure which updates parts of it taken from the new arrangements reached right after each heating-cooling procedure. All these accumulated updates are thus stored at the BSF which actually provides the current calculated minimum. The complete description of the LCC algorithm is given in Chapter 2.

CHAPTER 2

Graph minimum linear arrangement by multilevel weighted edge contractions

2.1 Introduction

The Minimum Linear Arrangement (MinLA) problem belongs to a large family of graph layout problems such as : Bandwidth, Cutwidth, Vertex Separation, Profile of Graph, Sum Cut etc. Commonly for general graphs these problems are NP-hard and their decision versions are NP-complete [48].

Originally the MinLA problem was formulated in 1964 by Harper in [54]. His aim was to design error-correcting codes with minimal average absolute errors on certain classes of graphs. The MinLA may also be motivated as a model used in VLSI design, where at the placement phase it is required to minimize the total wire length [30]. Additionally, the MinLA appears in several biological applications, graph drawing, reordering of large sparse matrices and other fields (see [39, 70, 57, 96]).

Since the MinLA has a practical significance, many heuristic algorithms were developed in order to achieve near optimal solution. Among the most successful are Spectral Sequencing [62], Optimally Oriented Decomposition Tree [6], Multilevel based [68], Simulated Annealing [79, 78], Genetic Hillclimbing [81] and some of their combinations. All these heuristics were tested on the test suite that was compounded by Petit [79], some have proven themselves superior in solution quality while other in execution time.

In this paper we present a new multilevel algorithm for the MinLA problem based on the Algebraic MultiGrid scheme (AMG) [15, 16, 12, 22, 89, 99, 100]. The main objective of a multilevel based algorithm is to create a hierarchy of problems (*coarsening*), each representing the original problem, but with fewer degrees of freedom. General multilevel techniques have been successfully applied to various areas of science (e.g. physics, chemistry, engineering, etc.) [17, 14]. AMG methods were originally developed for solving linear systems of equations resulting from the discretization of partial differential equations. Lately they have been applied to various other fields, yielding for example a novel method for image segmentation [97]. In the context of graphs it is the Laplacian matrix that represents the related set of equations. The main difference between our approach to other multilevel approaches (not necessarily related to the MinLA but also to other graph optimization problems) is the coarsening scheme. While the previous approaches may be viewed as *strict* aggregation process, the AMG coarsening is actually a *weighted* aggregation. In a strict aggregation process (also called edge contraction or matching of nodes) the nodes of the graph are blocked into small disjoint subsets, called aggregates. By contrast, in weighted aggregation each node can be divided into *fractions*, and different fractions belong to different aggregates. In both cases, these aggregates will form the nodes of the coarser level. As AMG solvers have shown, *weighted*, instead of *strict* aggregation is important in order to express the *likelihood* of nodes to belong together; these likelihoods will then accumulate at the coarser levels of the process, automatically reinforcing each other where appropriate. This enables more freedom in solving the coarser levels and avoids making hardened local decisions, such as edge contractions, before accumulating the relevant global information, while a strict aggregation may lead to inconsistency between local and global pictures.

To escape false local minima we have used the general method of Simulated Annealing (SA) [67]. By introducing a temperature like parameter, moves which increase the cost function one wants to minimize are accepted with some non-vanishing probability. These algorithms are usually extremely inefficient, since they require exponential slow temperature decrease to approach the true minimum. In the multilevel framework, however, SA is aimed at searching for *local* changes with rapid cooling at each level that guarantees the preservation of large-scale solution features inherited from coarser levels.

Our experimental results show that the Algebraic Multilevel framework can be used for the MinLA problem to obtain high quality results in linear time. Our algorithm actually provides the best results (excluding one case) compared to [6, 38, 79, 78, 39, 81, 68], while its speed (despite its unoptimized current state) is much better than the fastest algorithm [68].

The problem definition and its generalization are described in Section 2.2. The multilevel algorithm along with additional optimization techniques are presented in Section 2.3. A comparison of our results with various other works is finally summarized in Section 2.4.

2.2 Problem definition and generalization

Given a weighted graph $G = (V,\ E)$, where $V = \{1, 2, ..., n\}$, denote by w_{ij} the non-negative weight of the edge ij between nodes i and j (if $ij \notin E$ then $w_{ij} = 0$).

The purpose of the MinLA problem is to find a permutation π of the graph nodes such that the cost $\sum_{ij\in E} w_{ij}|\pi(i) - \pi(j)|$ is minimal. In the generalized form of the problem that emerges during the multilevel solver, each vertex i is assigned with a *length* (or *volume*), denoted v_i. The task now is to minimize the cost $\sum_{ij\in E} w_{ij}|x_i - x_j|$, where $x_i = \frac{v_i}{2} + \sum_{k,\pi(k)<\pi(i)} v_k$, i.e., each vertex is positioned at its center of mass capturing a segment on the real axis which equals its length (see Figure 2.1). The original form of the problem is the special case where all the lengths are equal.

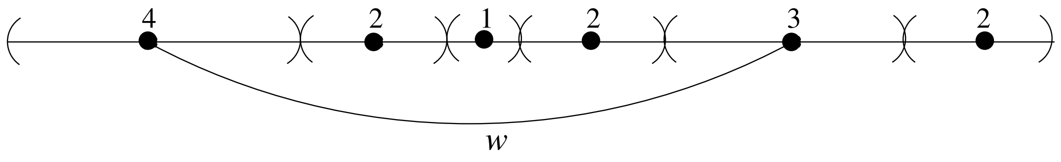


Figure 2.1: An example for the generalized form of the problem. Each node captures a segment on the real axis. Its length is written above it. If, for instance, the edge between the first node to the fifth one has weight w, then its contribution to the cost of the linear arrangement is $w \cdot 8.5$.

We will not discuss here theoretical complexity issues, such as lower and upper bounds for the solution cost. We are not interested in the worst possible cases, which are extremely non-representative. Our focus is on practical high-performance algorithm, such that in most practical cases would yield a good approximation to the optimum at low computational cost. Typically, the multilevel algorithms exhibit linear complexity, i.e., the computational cost in most practical cases is proportional to $|V| + |E|$.

2.3 The algorithm

In the multilevel framework a hierarchy of decreasing size graphs : $G_0, G_1, ..., G_k$ is constructed, see Figure 2.2. Starting from the given graph, $G_0 = G$, create by *coarsening* the sequence $G_1, ..., G_k$, then solve the coarsest level directly, and finally uncoarsen the solution back to G. This entire process is called a *V-cycle*.

As in the general AMG setting, the choice of the coarse variables (aggregates), the derivation of the coarse problem which approximates the fine one and the design of the coarse-to-fine disaggregation (uncoarsening) process are determined as described bellow.

2.3.1 Coarsening: Weighted Aggregation

Coarsening will be interpreted here as a process of *weighted aggregation* (or of weighted edge contraction) of the graph nodes to define the nodes of the next coarser graph. In a *strict* aggregation process (also called edge contraction or matching of

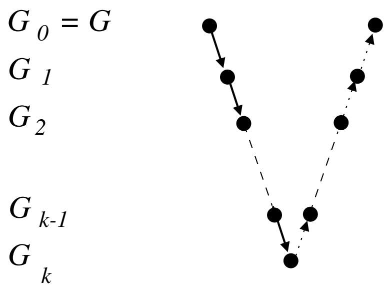


Figure 2.2: The Scheme of a V-cycle. Solid arrows stand for coarsening, dotted for uncoarsening.

nodes) the nodes are blocked in small disjoint subsets, called aggregates. Two nodes i and j would usually be blocked together (put in the same aggregate) only if their coupling is *strong*, meaning that w_{ij} is comparable to $min\{max_k w_{ik}, max_k w_{kj}\}$. In *weighted* aggregation, each node can be divided into $fractions$, and different fractions belong to different aggregates. In both cases, these aggregates will form the nodes of the *coarser level*, where they will be blocked into larger aggregates, forming the nodes of a *still coarser level*, and so on. As AMG solvers have shown, *weighted*, instead of *strict*, aggregation is important in order to express the *likelihood* of nodes to belong together; these likelihoods will then accumulate at the coarser levels of the process, automatically reinforcing each other where appropriate. Strict aggregation, by contrast, may run into a conflict between the local blocking decision and the larger-scale picture.

The construction of a coarse graph from a given one is divided into three stages: first a subset of the fine nodes is chosen to serve as the *seeds* of the aggregates (the nodes of the coarse graph), then the rules for interpolation are determined, thereby establishing the fraction of each non-seed node belonging to each aggregate, and finally the strength of the connections (or edges) between the coarse nodes is calculated.

Coarse Nodes. The algebraic representation of a graph is given by its *Laplacian* A (a $|V| \times |V|$ matrix), whose terms are defined by

$$A_{ij} = \begin{cases} -w_{ij} & \text{for } ij \in E,\ i \neq j \\ 0 & \text{for } ij \notin E,\ i \neq j \\ \sum_{k \neq i} w_{ik} & \text{for } i = j \end{cases} . \tag{2.1}$$

The construction of the set of seeds C and its complement, denoted by F, is guided by the principle that each F-node should be "strongly coupled" to C. Also, we will include in C nodes with exceptionally large volume, or nodes expected (if used as seeds) to aggregate around them exceptionally large volumes of F-nodes. We start with an empty set C, hence $F = V$, and then sequentially transfer nodes from F to C, employing the following steps.

Let $w_S(ij)$ denote the normalized weight of an edge ij with respect to the set of nodes S and to the vertex i, defined by

$$w_S(ij) \;=\; \frac{w_{ij}}{\sum\limits_{k \in S} w_{ik}} \,. \tag{2.2}$$

As a measure of how large an aggregate seeded by $i \in F$ might grow, define its *future-volume* ϑ_i by

$$\vartheta_i = v_i + \sum_{j \in F} v_j \cdot min(1, \frac{d_j}{\rho_j} \cdot w_V(ji)) \qquad , \tag{2.3}$$

where d_j is the degree of j and $\rho_j = min(r, \lceil Q \cdot d_j \rceil)$ is a rough estimate of the number of C nodes to which node j will be connected, the threshold Q (see below) and the coarse neighborhood size r (see Appendix B) being parameters. The basic idea is that each F-node will eventually be associated with only a *limited* number (the coarse neighborhood size r) of C-nodes, thus the relative connection $w_V(ji)$ of each $j \in F$ to i is usually amplified by $\frac{d_j}{\rho_j}$, as for the cases where $r < d_j$, the volume v_j is divided among less neighbors. Nodes with future-volume larger than η times the average of ϑ should automatically be included in C as most "representative". (In our tests $\eta = 2$). The insertion of additional fine nodes to C depends on a threshold Q (in our tests $Q = 0.4$) as specified by Algorithm 1. That is, a fine node i is added to C if its relative connection to C is not strong enough, i.e., smaller than Q. Also, vertices with larger values of ϑ are given higher priority to be chosen to C.

Algorithm 1: CoarseNodes(fine level $\mathcal{F}$)

```
Parameters: $Q,\ \eta$

$C \leftarrow \emptyset,\ F \leftarrow V$
Calculate $\vartheta_i$ for each $i \in F$
$C \leftarrow$ nodes with $\vartheta > \eta \cdot (average\ of\ \vartheta)$
$F \leftarrow V \setminus C$
Recalculate $\vartheta_i$ for each $i \in F$
Sort $F$ in descending order of $\vartheta$
Go through all $i \in F$ in decreasing order of $\vartheta$
  If $\left( \sum_{j \in C} w_{ij} / \sum_{j \in V} w_{ij} \right) \leq Q$ then move $i$ from $F$ to $C$
Return $C$
```

For convenience we are currently using a library $O(n \cdot \log(n))$ sorting algorithm. However, since no exact ordering is really needed, this can be replaced by a rough sort which has $O(n)$ complexity. This remark will be valid for all cases where we have used exact sort.

The Coarse Problem. The chosen set C of seeds becomes the set of coarse level nodes. Define for each $i \in F$ a coarse neighborhood $N_i = \{j \in C,\ w_{ij} \geq \alpha_i\}$, where α_i is determined by the demand that $|N_i|$ does not exceed the allowed coarse neighborhood size r chosen to control complexity. (For typical values of r consider Appendix B). The classical AMG interpolation matrix P (of size $|V| \times |C|$) is defined by

$$P_{ij} = \begin{cases} w_{N_i}(ij) & \text{for } i \in F,\ j \in N_i \\ 1 & \text{for } i \in C,\ j = i \\ 0 & \text{otherwise} \end{cases} . \tag{2.4}$$

P_{ij} thus represents the likelihood of i to belong to the j-th aggregate. The Laplacian of the coarse graph A^c can be calculated by the so called Galerkin coarsening $A^c = P^T AP$. Here, however, we follow the approximated scheme used in [97], namely, the edge connecting two coarse aggregates i and j, w^c_{ij}, is assigned with the weight $w^c_{ij} = \sum_{k \neq l} P_{ki} w_{kl} P_{lj}$. The volume of the i-th coarse aggregate is $\sum_j v_j P_{ji}$. Note that during the process of coarsening the total volume of all vertices is conserved.

2.3.2 The coarsest level

Solving the coarsest level, which consists of no more than 8 nodes (otherwise a still coarser level should be introduced for efficiency) is performed directly by simply trying all possible arrangements and choosing the minimal one.

2.3.3 Disaggregation (uncoarsening)

Having solved a coarse problem, the solution to the next-finer-level problem is initialized by first placing the seeds according to the coarse order and then adjusting all other F-nodes while aiming at the minimization of the arrangement cost. This approximation is subsequently improved by several *relaxation* sweeps, first compatible, then regular with or without additional stochastic elements, as explained below and summarized in Algorithm 2.

Initialization

Given is the linear arrangement of the coarse level aggregates in its generalized form, where the center of mass of each aggregate $j \in C$ is positioned at x_j along the real axis. We begin the initialization of the fine level arrangement by letting each seed inherit the position of its respective aggregate. Define $V' \subset V$ to be the subset of nodes that have already been placed, i.e., initially $V' = C$. Then proceed by positioning each fine node $i \in V \setminus V'$ at the coordinate y_i in which the cost of the arrangement, at that moment when i is being placed, is minimized. The sequence in which the nodes are placed is roughly in decreasing order of their *relative* connection to V', that is, the nodes which have strong connections to V' compared with their connections to V are placed first. To be precise, the coordinate y_i is located within the *minimization segment* (possibly containing a single point) defined by

$$\{y \ : \ |\sum_{y_j<y,\ j\in V'} w_{ij} - \sum_{y_j>y,\ j\in V'} w_{ij}| \quad \text{is minimal}\}, \tag{2.5}$$

which can be easily obtained by calculating the partial sums of weights along the already placed neighbors of vertex i. Note that for the case where all the w's are identical, as indeed in the original graph, y_i is just the *median* of the locations of the already placed neighbors of i, as in [68]. In the general case, y_i is placed within the minimization segment, where the sum of all left oriented adjacent edges is roughly equal to the sum of all right oriented adjacent edges, as close as possible to the end of the bigger sum and thus minimizes the cost of the arrangement with respect to i. Then $V' \leftarrow V' \cup \{i\}$ and the process continues until $V' = V$. Finally each position y_i is changed to

$$x_i \ = \ \frac{v_i}{2} + \sum_{y_k<y_i} v_k \qquad , \tag{2.6}$$

thus retaining order while taking volume (length) into account.

Relaxation

The arrangement obtained after the initialization is not likely to be accurate enough, only about 25% of the vertices end up within their minimization segment (satisfying (2.5) for $V' = V$). It should therefore be followed by several sweeps of *relaxation*, first *compatible* then *Gauss-Seidel-like*. These two types of relaxation are very similar to the above initialization. In each sweep, the nodes are scanned in their natural order, replacing their position one at a time by locally minimizing the cost of the arrangement associated with it. The compatible relaxation, motivated in [13], only improves the positions of the F-nodes according to the minimization criterion (2.5) (where $V' = V$) while keeping the positions of the seeds (C-nodes) unchanged, the Gauss-Seidel-like relaxation is similarly performed everywhere. Each such sweep is again followed by (2.6). Our tests show that by employing just a small number of relaxation sweeps the number of vertices located within their minimization segment grows to about 70%.

Strict minimizations

A simple strict minimization is a relaxation that accepts only changes which decrease the arrangement cost. Since done in the multilevel framework, it can be restricted at each level to just *local* changes, i.e., reordering small sets of neighboring nodes, which are adjacent (or relatively close) to each other at the current linear arrangement. It is easy to see that switching positions between several adjacent nodes is indeed a local operation, since the resulting new arrangement cost can be calculated only at the vicinity of the adjustment and not elsewhere.

Node-by-node minimization. We have chosen to minimize over a sequence of local changes in which the candidate positions for each vertex i, in its turn, are scanned over a segment of (maximal) length of $2k+1$, starting k positions to the left of the current location of i, ending k positions to its right (with exceptions of course at the beginning and end of the arrangement). Each of the candidate positions has its own cost and the arrangement with the minimal cost is finally chosen. Such minimization sweeps are repeated until no significant improvement in the arrangement is observed or until a given maximal allowed number of repetitions is reached. This parameter as well as k are addressed in Appendix B.

Segment minimization. We have also used another more sophisticated minimization strategy that operates on segments of subsequent nodes. In each sweep, the nodes are scanned according to their current linear arrangement, extracting *weakly* connected segments of subsequent nodes. A weakly connected segment of nodes is a segment which is connected within itself but is either completely disconnected or only slightly connected to its right and left neighbors in the arrangement. Then the position of each such segment is replaced by minimizing the cost of the arrangement associated with it. The minimization of the energy of such a segment is similar to that of a single node. Let $W = \{i_1 = \pi^{-1}(p+1), ..., i_q = \pi^{-1}(p+q)\}$ be a segment of q sequential vertices in the current arrangement, i.e., the nodes positioned at q subsequent coordinates starting at the p-th position. W will be moved to the position where the sum of all its edges to the right is as equal as possible to the sum to its left, that is, we used a generalization of the criterion (2.5), where the sums run over all $i \in W$. The sweeps are again repeated up to some maximal allowed number of iterations. This minimization has been in particular successful for meshes as is summarized in Table 2.3.

Simulated Annealing

A general method to escape false local minima and advance to lower costs is to replace the strict minimization by a process that still accepts each candidate change which lowers the cost, but also assigns a positive probability for accepting a candidate step which increases the cost of the arrangement. The probability assigned to a candidate step is equal to $exp(-\delta/T)$, where $\delta > 0$ measures the *increase* in the arrangement cost and $T > 0$ is a temperature-like control parameter which is gradually decreased towards zero. This process, known as *Simulated Annealing* (SA) [67], in large problems would usually need to apply *very gradual* cooling (decrease of temperatures), making it extremely slow and inefficient for obtaining global optimum.

In the multilevel framework, however, the role of SA is somewhat different. At each level it is assumed that the *global* arrangement of aggregates has been inherited from the coarser levels, and thus only *local*, small-scale changes are needed. For that purpose, we have started at relatively high T, lowered it *substantially* at each subsequent sweep until strict minimization is employed.

Similar to the above strict minimization, $2k+1$ candidate locations are examined for each vertex, each corresponds to moving it some distance l, $0 < |l| \leq k$. The initial temperature $T = T(|l|) > 0$ is calculated apriori for each distance l by aiming at the acceptance of a certain percent of changes (for instance 60%). In detail, the probability of moving a vertex l positions ($l = \pm 1, ..., \pm k$) is $Pr(l) = z \cdot min(1, exp(-\delta(l)/T(|l|))$, where z is a normalization factor calculated by the demand $\sum_{l=-k}^{k} Pr(l) = 1$ and $Pr(0) = z \cdot min_{l=\pm 1,...,\pm k}(1 - Pr(l)/z)$. In each additional sweep $T(|l|)$ is reduced by a factor, such as 0.6. Typically only three such cooling steps are used.

Repeated heating and cooling is successively employed for better search over the local landscape. This search can be further enhanced by the introduction of a "memory" like tool consisting of an additional permutation which stores the *Best-So-Far* (BSF) observed arrangement. Henceafter, the BSF is being occasionally updated by the procedure called *Lowest Common Configuration* (LCC) [18] which enables the systematic accumulation of *sub*-permutations into it over a sequence of different arrangements, such that each BSF sub-permutation exhibits the best minimal sub-order encountered so far. The cost of the obtained BSF is at most the lowest cost of all the arrangements it has observed, and usually it is lower. The use of LCC becomes more important for larger graphs, where it is expected that the optimum of a subgraph is only weakly dependent on other subgraphs. Thus, it is not necessary to wait until all minimal sub-permutations are *simultaneously* obtained, which may take exponential time; instead it is sufficient to obtain each such minimal sub-order just once, since henceforth it is guaranteed to appear in the BSF. In detail, the BSF (of a certain level) is initialized by the arrangement obtained at the end of the strict minimization. Then the BSF is improved by the LCC procedure which updates parts of it taken from the new arrangements reached right after each heating-cooling procedure. All these accumulated updates are thus stored at the BSF which actually provides the current calculated minimum. The complete description of the LCC algorithm is given in Appendix A.

Algorithm 2: Disaggregation(coarse level $\mathcal{C}$, fine level $\mathcal{F}$)

```
Parameters: $k_1, ..., k_8, \gamma$ (for details consider Appendix B)

Initialize $\mathcal{F}$ from $\mathcal{C}$
Apply $k_1$ sweeps of compatible relaxation on $\mathcal{F}$
Apply $k_2$ sweeps of Gauss-Seidel-like relaxation on $\mathcal{F}$
Apply at most $k_3$ sweeps of strict minimization within distance $k_4$ on $\mathcal{F}$
Apply at most $k_5$ sweeps of segment minimization on $\mathcal{F}$
Initialize $BSF \leftarrow$ current arrangement of $\mathcal{F}$
Do $k_6$ times of heating and cooling
```

```
    **Calculate** $T(|l|)$ for $l = 1, ..., k_7$
    **Do** $k_8$ steps
        **Apply** SA within distance $k_7$ on $\mathcal{F}$
        **Decrease** all $T(|l|)$ by a factor $\gamma$
    **Apply** at most $k_3$ sweeps strict minimization within distance $k_4$ on $\mathcal{F}$
    **Update** $BSF \leftarrow LCC(BSF,$ current arrangement of $\mathcal{F})$
**Return** $BSF$
```

2.3.4 Linearization and cycling

The graph Laplacian yields a good coarsening (the AMG coarsening) when the problem is associated with, or approximated by, the problem of minimizing the quadratic functional $\sum_{ij\in E} w_{ij}(x_i - x_j)^2$. A better quadratic formulation to a non-quadratic minimization problem can usually be obtained in terms of a *current approximation*, in the spirit of Newton linearization (see [17]). The main property of such an approximate quadratic formulation is *stationarity*, i.e., the quadratic formulation will reproduce the current approximation if the latter happens to be already the solution to the original (non-quadratic) problem. In the context of the MinLA, given a current approximation $\{\tilde{x_i}\}$, a stationary quadratic approximation to the generalized MinLA problem is :

$$\text{minimize} \sum_{ij\in E} \frac{w_{ij}}{|\tilde{x_i} - \tilde{x_j}|^\alpha}(x_i - x_j)^2 \text{ , with } \alpha = 1. \tag{2.7}$$

At each level of the multiscale MinLA solver, several cycles to coarser levels can thus be performed, using first the original ($\alpha = 0$) quadratization, then in following cycles gradually increasing α to 1. Using a certain value of α means here to employ newly defined weights $w_{ij}^{new} = w_{ij}/|\tilde{x_i} - \tilde{x_j}|^\alpha$ instead of the original Laplacian in forming the aggregation seeds and interpolation weights. That is, a previously obtained approximate solution is used to create weights for the next cycle. We have used this idea only partially, i.e., by performing only *complete* V-cycles (returning to coarser levels just from the finest level), with $\alpha = 0, 1/2, 1$ successively, while updating the BSF of the finest level by applying the LCC procedure at the end of each additional V-cycle. Note, however, that (2.7) is stationary only for the real-number approximation to MinLA; it is not stationary when the constraint that $\{x_i\}$ should be a permutation of $(1, ..., n)$ is added.

2.4 Results and Related Works

We have implemented and tested the algorithm using standard C++ and LEDA libraries [74] on Linux machine with 1700MHz processor. The implementation is non-parallel and not fully optimized.

We have started to test our algorithm on the benchmark provided by Petit [79]. The test suite graphs are given in Table 2.1. In Table 2.2 we present the results we have obtained for these graphs compared with other related works. In the column "**Petit**" we have extracted the *best* results reported in Petit's et al. articles [39, 38, 79, 78]. These results were usually obtained by combining spectral sequencing method with simulated annealing. In the column "**KH**" we show the results of Koren and Harel [68]. They developed a linear-time algorithm for the MinLA, based on the combination of spectral methods with the multi-scale paradigm. We present their best reported results, those obtained after 10 full V-cycles. In the column "**BEFN**" the results of Bar-Yehuda et al. [6] are given. They have developed a polynomial time algorithm (with complexity $O(|V|^{2.2})$) for computing an optimal ordering induced by a binary balanced decomposition tree as an initial solution followed by simulated annealing. Although Bar-Yehuda et al.'s results are of high quality, their algorithm cannot be used for very large inputs due to its high complexity. Finally the "**Poranen**" column includes the results obtained by the stochastic algorithm called "genetic hillclimbing" [81].

The running time of our algorithm clearly depends on several parameters. We have basically used three types of V-cycles : the "quick" V-cycle which is aimed at achieving fast performance and thus somewhat compromising the quality of the arrangement cost, the "extended" V-cycle which runs longer but succeeds in finding lower cost arrangements, and the "super" V-cycle which provides even better results but runs on the average twice slower for this test suite. The relevant parameters for all types are presented in Appendix B. The "quick" V-cycle is mostly useful for large graphs (like those in Table 4) for which it is crucial to cut down execution time. Here, for the relatively small sized graphs of Petit's benchmark, we have omitted its detailed results, since the "extended" V-cycle already runs fast enough and naturally provides better results. The column (of Table 2.2) marked by "**Ours** : extended" summarizes the best results observed out of 100 runs (using different sequences of random numbers) of three "extended" V-cycles each. The column (of Table 2) marked by "**Ours** : super" summarizes the best results observed out of 50 similar runs of three "super" V-cycles each. Excluding the first four random graphs (discussed next), it is evident that our algorithm almost always provides the best results, if not by using the "extended" V-cycle, then when applying the "super" one. Also important is the fact that the calculated standard deviation of the trials is no bigger than 1% (for both the "extended" and the "super" V-cycles) of the minimal result shown in the table and usually it is much smaller. One "quick" V-cycle gave on the average 107.3% of our best results, while three "quick" V-cycles improve it to 105.4%. One "extended" V-cycle further improved the results to 103.3% and three "extended" V-cycles to 101.5%. Extracting the best results out of only three runs (using different sequences of random numbers) of three "extended" V-cycles already gave 100.9% of the best seen costs. Since the running time of our algorithm is almost negligible for many of the graphs of Petit's test suite we present it (in Table 2.5) only for the much larger graphs given in Table 2.4 and discussed below.

Table 2.1: Petit's benchmark [79].

Graph Name	$\|V\|$	$\|E\|$	**Min/Avg/Max degree**	**Diameter**
randomA1	1000	4974	1/9.95/21	6
randomA2	1000	24738	28/49.7/72	3
randomA3	1000	49820	72/99.64/129	4
randomA4	1000	8177	4/16.35/29	4
randomG4	1000	8173	5/16.34/31	23
hc10	1024	5120	10/10/10	10
mesh33x33	1089	2112	2/3.88/4	64
bintree10	1023	1022	1/1.99/3	18
3elt	4720	13722	3/5.81/9	65
airfoil	4253	12289	3/5.78/10	65
crack	10240	30380	3/5.93/9	121
whitaker3	9800	28989	3/5.91/8	161
c1y	828	1749	2/422/304	10
c2y	980	2102	1/4.29/327	11
c3y	1327	2844	1/4.29/364	13
c4y	1366	2915	1/4.26/309	14
c5y	1202	2557	1/4.25/323	13
gd95c	62	144	2/4.65/15	11
gd96a	1076	1676	1/3.06/111	20
gd96b	111	193	2/3.47/47	18
gd96c	65	125	2/3.84/6	10
gd96d	180	228	1/2.53/27	8

Random graphs. Two kinds of random graphs were introduced in Petit's test suite : (a) Uniform random graphs $G_{n,p}$ (randomA[1,2,3,4]), where $n = 1000$ is the number of vertices and p is the probability of having an edge between any two nodes, and (b) Geometric random graph $G_{n,d}$ (randomG4) with $n = 1000$ uniformly distributed nodes in a unit square, such that each pair of nodes which have smaller distance than d are connected by an edge. It is clear that our algorithm succeeds when the graph has some *geometric* structure like in "randomG4", and unlike "randomA[1,2,3,4]". While most algorithms perform rather well for those uniform random graphs, producing results of comparable quality, the best shown results are those observed by Petit et al. using simulated annealing, which is basically a stochastic search. We have however checked that for fixed n and p, different random generated numbers create different uniform graphs which nonetheless always exhibit *similar* linear arrangement cost results. And the important point is that cost variations due to different heuristics are within variations anyway produced by random changes in the graph. Therefore, as already stated by Petit [38, 79], uniform random graphs are actually unworthy for the purpose of evaluating heuristic algorithms (see analytical explanation in [40]).

Graphs with known minimum. To further measure the quality of our heuristic, we have tested it on graphs for which the MinLA value is known. Three such examples already appear in Table 2.1, namely, the hypercube graph ("hc10"), the

Table 2.2: Comparative table of results for the test suite of Table 2.1. The obtained minimum is bolded.

Graph	**Petit**	**KH**	**BEFN**	**Poranen**	**Ours :** “extended”	**Ours :** “super”
randomA1	**867570**	909126	884261	878637	890430	888381
randomA2	**6528780**	6606174	6576912	6553553	6610933	6596081
randomA3	**14202700**	14457452	14289214	—	14349635	14303980
randomA4	**1721670**	1765217	1747143	1739317	1757119	1747822
randomG4	150940	149513	146996	142587	140240	**140211**
hc10	**523776**	**523776**	**523776**	**523776**	**523776**	**523776**
mesh33x33	31929	**31729**	33531	32178	31895	**31729**
bintree10	4069	3950	3762	3899	3707	**3696**
3elt	363204	373464	363204	383286	359977	**357329**
airfoil	285231	291794	289217	306005	275833	**272931**
crack	1491126	—	—	—	1507325	**1489266**
whitaker3	1151064	1205919	1200374	1203349	1152441	**1144476**
c1y	62936	64934	62333	62857	62545	**62262**
c2y	79429	80148	79571	80327	79200	**78822**
c3y	123548	127315	127065	125654	126111	**123514**
c4y	116140	118437	115222	119232	115935	**115131**
c5y	100264	104076	96956	99389	97840	**96899**
gd95c	**506**	509	**506**	**506**	**506**	**506**
gd96a	96342	106668	99944	101394	98042	**96249**
gd96b	**1416**	1434	1417	**1416**	**1416**	**1416**
gd96c	**519**	**519**	**519**	**519**	**519**	**519**
gd96d	2393	2393	2409	**2391**	**2391**	**2391**

lattice graph (“mesh33x33”) and the binary tree (“bintree10”) [39]. In addition, we have added four larger lattices (“mesh100x100”, “mesh200x200”, “mesh500x500”, “mesh1000x1000”) and three proper interval graphs which also have known minima [91]. The results are shown in Table 2.3. We have employed three “extended” V-cycles enhanced by the segment minimization (see Section 2.3.3). Eventhough the very particular known optimum for meshes was not fully reached, our solutions did show very similar structures and close results even for the large meshes, as is indicated by the last column of Table 2.3: The quality of our solution has not deteriorated with the growth of the mesh.

Larger graphs. Since the execution time of our algorithm is basically linear (even in its current unoptimized state) we were looking for additional large sized test cases. We have found only one paper with such results, the one by Koren and Harel [68], which is indeed the only one exhibiting linear execution time. In this test suite we have used the same “quick” and “extended” V-cycles as in Petit’s experiments. The results and running time (in minutes) are summarized in Table 2.5. Column “**KH**” presents those obtained by Koren and Harel after five full V-cycles. (We have chosen to present these results rather than those obtained after 10 V-cycles as the latter only improve the former by less than 1% but run twice as slow.) The two columns marked by “**Ours**” show the extremely fast performance and very

Table 2.3: Comparative table of results for graphs with known minimum.

Graph	$\|V\|$	$\|E\|$	Our cost	Optimal cost	Our/Optimal
mesh33x33	1089	2112	31720	31680	1.001
mesh100x100	10000	19800	880234	868820	1.013
mesh200x200	40000	79600	7028594	6923320	1.015
mesh500x500	250000	49900	109972299	107916916	1.019
mesh1000x1000	1000000	1998000	879287403	862634024	1.019
bintree10	1023	1022	3696	3696	1
hc10	1024	5120	523776	523776	1
Proper Interval Graph I	200	3213	30766	30766	1
Proper Interval Graph II	500	14784	250241	250241	1
Proper Interval Graph III	1000	45393	1.19709e+06	1.19709e+06	1

good results of our algorithms: our single "quick" V-cycle runs (on the average) less than 20% of the running time of Koren and Harel's algorithm and improves their results by 8.3%, while our three "extended" V-cycles run ($\sim$ 3.5 times) slower but exhibits results which are 12% better. Each cost presents the *average* result obtained over 10 runs of different sequences of random numbers, for which we have measured a standard deviation not larger than 2%. (Note that stochastisity enters not only in the SA procedure but also in the given initial order of nodes which affects the coarsening procedure given by Algorithm 1.) Additional tests show that three "quick" V-cycles already improve over "KH" by 10%, and that dropping the LCC procedure (within the SA process) from the runs of three "extended" V-cycles has worsen those results by about 1%. This last result demonstrates the ability of the LCC to further extract better minima. We found that the "super" V-cycle is unuseful here since it does not show any significant improvement of results, while the increase in running time (because of the growing degree of the coarse graphs and additional SA) makes it unusable for practical purposes, especially for the largest graphs.

Table 2.4: KH large graphs test suite.

Graph	$\|V\|$	$\|E\|$	Degree min/max
tooth	78136	452591	3/39
ocean	143437	409593	1/6
mrngA	257000	505048	2/4
rotor	99617	662431	5/125
598	110971	741934	5/26
144	144649	1074393	4/26
m14b	214765	1679018	4/40
mrngB	1017253	2015714	2/4
auto	448695	3314611	4/37

Table 2.5: Comparative table of results for large graphs. The obtained minimum is bolded.

Graph	**KH** : 5 V-cycles cost/time	**Ours** : "quick" single V-cycle cost/time	Improvement (Ours÷KH) cost/time	**Ours** : "extended" 3 V-cycles cost/time	Improvement (Ours÷KH) cost/time
tooth	255.465.042/10.5	237.353.161/1.2	0.929/0.114	**227.639.682**/27	0.891/2.571
ocean	141.732.687/13.5	131.968.513/3.2	0.931/0.237	**118.882.522**/72	0.839/5.333
mrngA	348.448.986/23.5	319.286.767/6	0.916/0.255	**305.560.971**/90	0.877/3.830
rotor	247.583.742/16.5	230.618.627/1.9	0.931/0.115	**221.832.991**/42	0.896/2.545
598	340.886.008/19	287.702.639/3	0.844/0.158	**281.033.967**/57	0.824/3.000
144	772.846.779/28.5	764.706.283/4.4	0.989/0.154	**745.701.842**/84	0.965/2.947
m14b	1.004.606.217/40	877.930.925/6.8	0.877/0.170	**857.743.008**/130	0.854/3.250
mrngB	3.558.254.373/98	3.377.861.206/38	0.949/0.388	**3.254.023.540**/520	0.914/5.306
auto	4.501.150.138/100	3.986.693.232/18	0.886/0.180	**3.871.472.605**/340	0.860/3.400
Average			0.917/0.197		0.880/3.576

2.5 Conclusions

We have presented a new multilevel algorithm for the MinLA problem for general graphs. The algorithm is based on the general principle that during coarsening each vertex may be associated to more than just one aggregate according to some "likelihood" measure. The uncoarsening initialization, which produces the first arrangement of the fine graph nodes, strongly relies on energy considerations (unlike usual interpolation in classical AMG). This initial order is further improved by local strict minimization relaxation and possibly by employing stochasticity, i.e., simulated annealing. The running time of the algorithm is linear, thus it can be applied to very large graphs.

We have basically used three types of V-cycles: the "quick", "extended" and "super". The "extended" V-cycle includes SA, which is enhanced by the LCC procedure, and spends more time on local minimization. The "super" V-cycle is aimed at achieving even better results for small and medium sized graphs. The "quick" one runs very fast and provides results which are at most about 11% (on the average 4%) off the better results obtained by the "extended" and "super" V-cycles. Due to stochastic elements, different results are observed for different sequences of random numbers; however, all our tests show that this variability is not larger than 2%.

Our main conclusion is that the average and the best results of our "extended" and "super" V-cycles are almost always better than the results of completely stochastic heuristics (simulated annealing, genetic hillclimbing, etc.), the Fiedler vector multilevel algorithm and the binary balanced decomposition tree algorithm. For uniform random graphs it is clear that some results obtained by stochastic heuristics outperform ours. This is because our algorithm succeeds when the graph has non-stochastic structure, i.e., in more intuitive words it has "some geometry". We recommend our multilevel algorithm as a general practical method for solving the

Minimum Linear Arrangement problem. The implemented algorithm can be obtained at *http://www.wisdom.weizmann.ac.il/∼safro/minla.*

Acknowledgments

This research was supported by a Grant from the German-Israeli Foundation for Scientific Research and Development (G.I.F.), Research Grant Agreement No. I-718-135.6/2001, and by the Carl F. Gauss Minerva Center for Scientific Computation at the Weizmann Institute of Science.

Appendix A: Lowest Common Configuration (LCC)

The algorithm presented below was originally designed for the Traveling Salesman Problem [86]. Given two arrangements of the graph nodes $\varphi = (\pi_1^{-1}(1), \pi_1^{-1}(2), ..., \pi_1^{-1}(n))$ and $\psi = (\pi_2^{-1}(1), \pi_2^{-1}(2), ..., \pi_2^{-1}(n))$, their LCC, denoted $LCC(\varphi, \psi)$, is a third linear arrangement whose cost is lower than (or at most equals to) the costs of both φ and ψ, produced as follows.

Define as a *common sub-permutation* (CSP) of φ and ψ any subset S for which, for certain i and j, the following two requirements hold :

1. $S = \{\varphi(i), \varphi(i+1), ..., \varphi(i+|S|-1)\} = \{\psi(j), \psi(j+1), ..., \psi(j+|S|-1)\}$
2. $\{\varphi(i), \varphi(i+|S|-1)\} = \{\psi(j), \psi(j+|S|-1)\}$.

That is, the subset S appears as a consecutive sequence of nodes in both φ and ψ, possibly in different orders, but with common ends.

$LCC(\varphi, \psi)$ is constructed by first finding all the CSPs S of φ and ψ, and then choosing for each S the suborder from either φ or ψ, whichever yields the lower cost arrangement. It is not straightforward to find all CSPs of given φ and ψ, especially if the complexity of that subroutine is required not to dominate the entire complexity of the multilevel solver. We have constructed an algorithm which finds all CSPs in nearly linear time. The algorithm is based on the following observations.

Consider a pair of consecutive suborders (one is taken from φ and the other from ψ) whose ends are common and lengths are equal. Such a pair of suborders is *suspected* of being a CSP (SCSP). Our aim is to find all SCSPs which with very high probability are indeed CSPs.

Attach to each vertex j some marking M_j, a real number. Construct for φ the vector $(SM)^\varphi$ of the partial sums of these markings $(SM)_i^\varphi = \sum_{l=1}^{i} M_{\varphi(l)}$. Similarly, construct $(SM)^\psi$ for ψ. Let $\varphi(i), \varphi(i+1), ..., \varphi(i+k)$ and $\psi(j), \psi(j+1), ..., \psi(j+k)$ be a SCSP. If the SCSP is also a CSP then the following holds:

$$(SM)_{i+k}^\varphi - (SM)_i^\varphi \;=\; (SM)_{j+k}^\psi - (SM)_j^\psi \; . \qquad (2.8)$$

The opposite is, however, not always true : (2.8) may hold for such a SCSP even when the suborders are *not* permutations of each other. Consequently the markings should be chosen so that this ambiguity will practically never happen. It is enough for example to choose M_i to be a random number between 0 and 1 taken to some power p. Clearly, the probability that (2.8) holds while the SCSP is not a CSP is extremely low.

Equation (2.8) can also be written as

$$(SM)_i^{\varphi} - (SM)_j^{\psi} \; = \; (SM)_{i+k}^{\varphi} - (SM)_{j+k}^{\psi} \; . \tag{2.9}$$

If $\varphi(i) = \psi(j) = l$ and $\varphi(i+k) = \psi(j+k) = m$, say, and if we define for every vertex l the "assignment"

$$A_l = (SM)_{\pi_1(l)}^{\varphi} - (SM)_{\pi_2(l)}^{\psi}, \tag{2.10}$$

Equation (2.9) implies that if $A_l = A_m$, then with very high probability l and m are ends of a CSP. Such pairs of vertices can easily be found by sorting the list of assignments. The final construction of the LCC follows by choosing the lower cost suborder for each CSP, in ascending order of the length of the CSPs, thus treating successfully even the rarely occurring situation of nested CSPs. All cases where $\varphi(i) = \psi(j+k)$ and $\varphi(i+k) = \psi(j)$ can also be found by repeating the above procedure while reversing the order of either φ or ψ, however in all our tests we have not found an indication of the importance of this additional procedure.

Appendix B: Parameters

In order to control the running time of the algorithm it is important to decrease the total number of edges of the constructed coarse graphs. This is achieved by the following two parameters: the maximum allowed coarse neighborhood size r, which restricts the allowed size $|N_i|$ of the coarse neighborhood of a vertex $i \in F$ by deleting the weakest w_{ij}, $j \in C$; and the edge filtering ϵ threshold, which deletes every *relatively* weak edge ij (from the created coarse graph) if both $w_{ij} < \epsilon \cdot s_i$ and $w_{ij} < \epsilon \cdot s_j$, where $s_i = \sum_k w_{ik}$.

These two parameters and five others which control the uncoarsening procedure (see Algorithm 2) are presented in Table 2.6 for the "quick", "extended" and "super" V-cycles we have used. The last two parameters (of Algorithm 2) were constantly chosen to be $k_8 = 4$ and $\gamma = 0.6$.

It is however important to mention that these parameters are the ones used only for the finest levels. As the coarse graphs become much smaller they are accordingly increased. This hardly affects the entire running time of the algorithm but systematically improves the obtained results. In the last column of Table 2.6 we specifically describe the increase introduced for each parameter as a function of level L, which usually depends on the ratio $R = max(1, |E_0|/|E_L|)$ measuring the relative decrease of the number of edges at level L compared with the original graph.

Table 2.6: The parameters used for the "quick", "extended" and "super" V-cycles. (* used only to obtain the results of Table 2.3)

Parameter	"quick" V-cycle	"extended" V-cycle	"super" V-cycle	The increase for level L
The coarse neighborhood size (r)	6	10	20	$+\log(R)$
The edge filtering threshold (ϵ)	0.01	0.005	0.001	$\cdot 0.9^{\log(R)}$
The number of sweeps of Compatible relaxation (k_1)	3	10	10	$+2 \cdot L$
The number of sweeps of Gauss-Seidel relaxation (k_2)	3	10	30	$+2 \cdot L$
The maximal number of sweeps of node-by-node minimization (k_3)	30	30	30	–
k_4 used in the node-by-node minimization	1	10	20	$+\log(\sqrt{R})$
The maximal number of sweeps of segment minimization (k_5)	0	0 (30*)	0	–
The number of heating and cooling in SA (k_6)	0	3	20	$\cdot \log(R)$
k_7 used in the SA	0	5	10	$+\log(\sqrt{R})$

CHAPTER 3

A multilevel algorithm for the minimum 2-sum problem

3.1 Introduction

The minimum p-sum problem (M2sP) belongs to a large family of graph layout problems such as : Bandwidth, Cutwidth, Vertex Separation, Profile of a Graph, Sum Cut, etc. The M2sP appears in several applications for solving problems in the large sparse matrix computation, such as finding the minimum linear arrangement [92, 68] or the bandwidth [94]. The M2sP is also closely related to the problem of calculating the envelope size of a symmetric matrix or more precisely, to the amount of work needed in the Cholesky factorization of such a matrix [50]. In addition, the M2sP may be motivated as a model used in VLSI design, where at the placement phase it is chosen to minimize the total squared wire length [30]. Commonly for general graphs (or matrices) these problems are NP-hard and their decision versions are NP-complete [48]. The NP-completeness of the M2sP is proved in [50].

The M2sP becomes a simple quadratic optimization problem with a *known* solution, due to Hall [51], if the restriction on the solution coordinates is relaxed, i.e., the coordinates need not be all integers, as in the case where all vertices are considered to have equal unity volume (see Section 3.2). Hall has shown in [51] that the eigenvector v_2 which corresponds to the second smallest eigenvalue of the Laplacian of the graph (provided the graph is connected), is the best nontrivial solution to this unrestricted form of the M2sP (subject to some normalization of the solution). Arrangement of the graph vertices according to v_2 is a well known, quite successful heuristic, usually called the *spectral approach*, used for many ordering problems like the minimum linear arrangement [68], partitioning [55, 83, 82, 98], envelope reduction of sparse matrices [7], etc.

George and Pothen [50] have studied the M2sP as they used it for establishing results for the envelope reduction of matrices. They tried to evaluate the quality of the approximation for the M2sP by the spectral approach in a quantitative manner. While for some finite element graphs they indeed got close results, for general graphs the gap was profound. They suggested that this gap can be reduced by applying some local reordering (postprocessing) to the obtained results of the spectral approach.

The fact that the solution for the M2sP with real variables is extensively used as a first approximation to other graph layout problems, brings up the idea that a good solution to the *discrete* M2sP can serve as well, if not better. The first question, of course, is how well the spectral approach solves the M2sP itself, that is, how well the solution with real variables approximates the discrete setting. This question is extensively tested in our paper. In addition, since the M2sP already penalizes long distances sufficiently strongly to practically prohibit very non-uniform distribution of distances, which is typical to many other layout problems, and since its formulation is nothing but a quadratic functional, it may be considered the simplest yet central among other layout problems. As such, M2sP can and should be used (in various ways, e.g., serve as a first approximation [94]) to help solving other problems. This requires, of course, having an efficient algorithm at hand, which is exactly the purpose of our research.

In this paper we present a new multilevel algorithm for the minimum p-sum problem based on the Algebraic MultiGrid scheme (AMG) [15, 16, 12, 22, 89, 99, 100]. The main objective of a multilevel based algorithm is to create a hierarchy of problems, each representing the original problem, but with fewer degrees of freedom. General multilevel techniques have been successfully applied to various areas of science (e.g. physics, chemistry, engineering, etc.) [14, 17]. AMG methods were originally developed for solving linear systems of equations resulting from the discretization of partial differential equations. Lately they have been applied to various other fields, yielding for example novel methods for image segmentation [97] and for the linear arrangement problem [92]. In the context of graphs it is the Laplacian matrix that represents the related set of equations. The main difference between our approach to other multilevel approaches (related to various graph optimization problems, e.g., [64]) is the coarsening scheme. While the previous approaches may be viewed as *strict* aggregation process (in which the nodes are simply blocked together into small groups and the edges are defined by the straightforward sum of the existing edges between these groups), the AMG coarsening is actually a *weighted* aggregation: each node may be divided into $fractions$, and different fractions belong to different aggregates. This enables more freedom in solving the coarser levels and avoids making hardened local decisions (such as the edge contractions made when strict aggregation is employed) before accumulating the relevant global information. The aggregation process we use here is similar to the one used for solving the minimum linear arrangement problem [92]. This part of the algorithm may, in principle, be general to many other graph layout problems (e.g., [87]), as it mainly creates the hierarchy of graphs. The diverse algorithmic ingredients, however, emerge during the disaggregation.

In the disaggregation step, the final arrangement obtained on a coarser level is projected to a finer level. This initial fine level arrangement is being further improved by applying various local reordering methods. In this article we introduce an algorithm for the strict minimization, called Window Minimization, which is based on the *simultaneous* reordering of several vertices. Then our postprocessing is intensified by Simulated Annealing (SA) [67] which is a general method to escape local minima. In the multilevel framework SA only aims at searching for *local* changes that guarantee the preservation of large-scale solution features inherited from coarser levels.

The power and robustness of our multilevel algorithm was proven by intensive experimental comparison with the spectral approach. Without the postprocessing the multilevel results are much better than the spectral ones by an average of 34.4%. After applying the same postprocessing (without simulated annealing) to both the multilevel and the spectral first approximations, the gap between the two frameworks was significantly reduced, but still the multilevel results are better by an average of 4.7%. Different parts of the postprocessing were enabled step-by-step in order to show the gradual improvement of the results. However, not only the results of the multilevel algorithm are better, but while our algorithm performs in linear time, the spectral approach is sensitive to the obtained accuracy; the trade-off between the complexity and the high accuracy of the calculation of v_2 is discussed in Section 3.4. Our experiments show that the Algebraic Multilevel approach can be used as a first approximation for the M2sP to obtain high quality results in linear time, while the postprocessing can actually improve these results and can serve as a tool for improving any first approximation of the M2sP obtained by other methods. The implemented algorithm can be obtained at *http://www.wisdom.weizmann.ac.il/∼safro/min2sum.*

The problem definition and its generalization are described in Section 3.2. The multilevel algorithm along with additional optimization techniques are presented in Section 3.3. A comparison of our results with the spectral approach is finally summarized in Section 3.4.

3.2 Problem definition and generalization

Given a weighted graph $G = (V,\ E)$, where $V = \{1, 2, ..., n\}$, denote by w_{ij} the non-negative weight of the edge ij between nodes i and j (if $ij \notin E$ then $w_{ij} = 0$). The purpose of the minimum p-sum problem is to find a permutation π of the graph nodes such that the cost $\sigma_2(G, \pi) = \sum_{ij} \big(w_{ij}(\pi(i) - \pi(j))^2\big)$ is minimal. In the generalized form of the problem that emerges during the multilevel solver, each vertex i is assigned a *volume* (or *length*), denoted v_i. The task now is to minimize the cost $\sigma_2(G, x, \pi) = \sum_{ij} \big(w_{ij}(x_i - x_j)^2\big)$, where $x_i = \frac{v_i}{2} + \sum_{k,\pi(k)<\pi(i)} v_k$, i.e., each vertex is positioned at its center of mass capturing a segment on the real axis which equals its length. Note that the difference $x_i - x_j$ contains the volumes of the vertices between i and j and half the volumes of i and j. The original form of the problem and the general form with equal-volume vertices are both minimized by the same permutation.

We are not interested in the worst possible cases, which are often very artificial. Our focus is on practical high-performance algorithm that will yield (in most practical cases) a good approximation to the optimum at low computational cost. Typically, the multilevel algorithms exhibit linear complexity, i.e., the computational cost in most practical cases is proportional to $|V| + |E|$.

3.3 The algorithm

In the multilevel framework a hierarchy of decreasing size graphs : $G_0, G_1, ..., G_k$ is constructed. Starting from the given graph, $G_0 = G$, create by *coarsening* the sequence $G_1, ..., G_k$, then solve the coarsest level directly, and finally uncoarsen the solution back to G. This entire process is called a *V-cycle*.

As in the general AMG setting, the choice of the coarse variables (aggregates), the derivation of the coarse problem which approximates the fine one and the design of the coarse-to-fine disaggregation (uncoarsening) process are all determined automatically as described below.

3.3.1 Coarsening: Weighted Aggregation

The coarsening used here is similar to the process we have used in solving the minimum linear arrangement problem [92]. However, for the completeness of this article, we briefly repeat it.

The coarsening is interpreted as a process of *weighted aggregation* of the graph nodes to define the nodes of the next coarser graph. In *weighted* aggregation each node can be divided into *fractions*, and different fractions belong to different aggregates. The construction of a coarse graph from a given one is divided into three stages: first a subset of the fine nodes is chosen to serve as the *seeds* of the aggregates (the nodes of the coarse graph), then the rules for interpolation are determined, thereby establishing the fraction of each non-seed node belonging to each aggregate, and finally the strength of the connections (or edges) between the coarse nodes is calculated.

Coarse Nodes. The construction of the set of seeds C and its complement, denoted by F, is guided by the principle that each F-node should be "strongly coupled" to C. Also, we will include in C nodes with exceptionally large volume, or nodes expected (if used as seeds) to aggregate around them exceptionally large volumes of F-nodes. To achieve these objectives, we start with an empty set C, hence $F = V$, and then sequentially transfer nodes from F to C, employing the following steps. As a measure of how large an aggregate seeded by $i \in F$ might grow, define its *future-volume* ϑ_i by

$$\vartheta_i = v_i + \sum_{j \in V} v_j \cdot \frac{w_{ji}}{\sum\limits_{k \in V} w_{jk}} \quad . \tag{3.1}$$

Nodes with future-volume larger than η times the average of the ϑ_i's are first transferred to C as most "representative". (In our tests $\eta = 2$). The insertion of additional fine nodes to C depends on a threshold Q (in our tests $Q = 0.4$) as specified by Algorithm 1. That is, a fine node i is added to C if its relative connection to C is not strong enough, i.e., smaller than Q. Also, vertices with larger values of ϑ_i are given higher priority to be chosen to belong to C.

Algorithm 1: CoarseNodes($Parameters : Q,\ \eta$)

```
$C \leftarrow \emptyset,\ F \leftarrow V$
**Calculate** $\vartheta_i$ for each $i \in F$, and their average $\overline{\vartheta}$
$C \leftarrow$ nodes $i$ with $\vartheta_i > \eta \cdot \overline{\vartheta}$
$F \leftarrow V \setminus C$
**Sort** $F$ in descending order of $\vartheta_i$
**Go** through all $i \in F$ in descending order of $\vartheta_i$
  **If** $\left( \sum_{j \in C} w_{ij} / \sum_{j \in V} w_{ij} \right) \leq Q$ **then** move $i$ from $F$ to $C$
**Return** $C$
```

For convenience we are currently using a library $O(n \log(n))$ sorting algorithm. However, since no exact ordering is really needed, this can be replaced by a rough bucketing sort which has $O(n)$ complexity. We have actually implemented a simple bucketing sort and compared its results with the exact sort ones. Since no significant differences were observed, we may state that the exact sorting does not play a role in the algorithm. The only important task is to identify the vertices with exceptionally large future-volume, which can easily be achieved by $O(n)$ procedure.

The Coarse Problem. Each node in the chosen set C becomes the seed of an aggregate that will constitute one coarse level node. Define for each $i \in F$ a coarse neighborhood $N_i = \{j \in C,\ w_{ij} \geq \alpha_i\}$, where α_i is determined by the demand that $|N_i|$ does not exceed the allowed coarse neighborhood size r chosen to control complexity. (For typical values of r consider the Appendix). The classical AMG interpolation matrix P (of size $|V| \times |C|$) is defined by

$$P_{ij} = \begin{cases} w_{ij} / \sum\limits_{k \in N_i} w_{ik} & \text{for } i \in F,\ j \in N_i \\ 1 & \text{for } i \in C,\ j = i \\ 0 & \text{otherwise} \quad . \end{cases} \tag{3.2}$$

P_{ij} thus represents the likelihood of i to belong to the j-th aggregate. Let $I(k)$ be the ordinal number in the coarse graph of the node that represents the aggregate around a seed whose ordinal number at the fine level is k. Following the weighted aggregation scheme used in [97], the edge connecting two coarse aggregates, $p = I(i)$ and $q = I(j)$, is assigned with the weight $w_{pq}^{(coarse)} = \sum_{k \neq l} P_{ki} w_{kl} P_{lj}$. The volume of the i-th coarse aggregate is $\sum_j v_j P_{ji}$. Note that during the process of coarsening the total volume of all vertices is conserved.

Solving the coarsest level, which consists of no more than 8 nodes (otherwise a still coarser level would be introduced for efficiency) is performed directly by simply trying all possible arrangements.

3.3.2 Disaggregation (uncoarsening)

Having solved a coarse problem, the solution to the next-finer-level problem is initialized by first placing the seeds according to the coarse order and then adjusting all other F-nodes while aiming at the minimization of the quadratic arrangement cost. This approximation is subsequently improved by several *relaxation* (local reordering) sweeps, first compatible, then regular with or without additional stochastic elements, as explained below and summarized in Algorithm 3.

Initialization

Given is the arrangement of the coarse level aggregates in its generalized form, where the center of mass of each aggregate $j \in C$ is positioned at $x_{I(j)}$ along the real axis. We begin the initialization of the fine level arrangement by letting each seed $j \in C$ inherit the position of its respective aggregate: $y_j = x_{I(j)}$. At each stage of the initialization procedure, define $V' \subset V$ to be the subset of nodes that have already been placed, so we start with $V' = C$. Then proceed by positioning each fine node $i \in V \setminus V'$ at the coordinate y_i in which the cost of the arrangement, at that moment when i is being placed, is minimized. The sequence in which the nodes are placed is roughly in decreasing order of their *relative* connection to V', that is, the nodes which have strong connections to V' compared with their connections to V are placed first. To be precise, the coordinate y_i is located at its minimum (volumes are not taken into account)

$$y_i = \frac{\sum_{j \in V'} y_j w_{ij}}{\sum_{j \in V'} w_{ij}}. \tag{3.3}$$

Then $V' \leftarrow V' \cup \{i\}$ and the process continues until $V' = V$. Finally each position y_i is changed to

$$x_i = \frac{v_i}{2} + \sum_{y_k < y_i} v_k \qquad , \tag{3.4}$$

thus retaining order while taking volume (length) into account.

Relaxation

The arrangement obtained after the initialization is a first feasible solution for M2sP which is then improved by employing several sweeps of *relaxation*, first *compatible* then *Gauss-Seidel*-like (GS). These two types of relaxation are very similar to the above initialization: The compatible relaxation, motivated in [13], improves the positions of (only) the F-nodes according to the minimization criterion (3.3) (where $V' = V$) while keeping the positions of the seeds (C-nodes) unchanged. The GS relaxation is similarly performed, but for *all* nodes (including C). Each such sweep is again followed by (3.4).

Window Minimization

The cost of the arrangement can be further reduced by *strict minimization*, a sequence of rearrangements that accepts only changes which decrease the arrangement cost. Since done in the multilevel framework, it can be restricted at each level to just *local* changes, i.e., reordering small sets of neighboring nodes, which are adjacent (or relatively close) to each other at the current arrangement. It is easy to see that switching positions between several adjacent nodes is indeed a local operation, since the resulting new arrangement cost can be calculated only at the vicinity of the adjustment and not elsewhere. Such a node by node minimization was applied in our algorithm for the Minimum Linear Arrangement problem (see [92]). This method may also be used for M2sP. However, we would like to propose a more advanced method of local minimization, called *Window Minimization* (WM), which is suitable for both the multilevel and the spectral approach frameworks. The difference between WM and simple node by node minimization is that WM *simultaneously* minimizes the arrangement cost of several nodes. Given a current approximation $\tilde{x}$ to the arrangement of the graph, denote by δ_i a *correction* to $\tilde{x}_i$. Let $\mathfrak{W} = \{i_1 = \pi^{-1}(p+1), ..., i_q = \pi^{-1}(p+q)\}$ be a *window* of q sequential vertices in the current arrangement, i.e., the nodes positioned at q subsequent coordinates $\tilde{x}_{i_1}, ..., \tilde{x}_{i_q}$. The local energy minimization problem associated with a given window $\mathfrak{W}$ can be formulated as follows :

$$\text{minimize } \sigma_2(\mathfrak{W}, \tilde{x}, \pi, \delta) = \sum_{i,j\in\mathfrak{W}} w_{ij}(\tilde{x}_i+\delta_i-\tilde{x}_j-\delta_j)^2 + \sum_{i\in\mathfrak{W},\ j\notin\mathfrak{W}} w_{ij}(\tilde{x}_i+\delta_i-\tilde{x}_j)^2. \quad (3.5)$$

To prevent the possible convergence of many coordinates to one point, and, more precisely, to express the aim of having $\{x_i + \delta_i\}_{i\in\mathfrak{W}}$ an approximate permutation of $\{x_i\}_{i\in\mathfrak{W}}$, we have added the following constraints

$$\sum_{i\in\mathfrak{W}} (\tilde{x}_i + \delta_i)^m v_i = \sum_{i\in\mathfrak{W}} \tilde{x}_i^{\,m} v_i \ , \quad m = 1, 2$$

where for $m = 2$ we have neglected the quadratic term in δ_i. Note that the sums $\sum_{i\in\mathfrak{W}} \tilde{x}_i^{\,m} v_i$ for $m = 1, 2$ are invariant under permutations. Using Lagrange multipliers, the final formulation of the window minimization problem is :

$$\text{minimize } \sigma_2(\mathfrak{W}, \tilde{x}, \pi, \delta, \lambda_1, \lambda_2) = \sigma_2(\mathfrak{W}, \tilde{x}, \pi, \delta) + \lambda_1 \sum_{i \in \mathfrak{W}} \delta_i v_i + \lambda_2 \sum_{i \in \mathfrak{W}} \delta_i v_i \tilde{x}_i \quad , \quad (3.6)$$

under the second and third constraints of (3.7) below, yielding the following system of equations:

$$\begin{cases} \sum_{j \in \mathfrak{W}} w_{ij}(\delta_i - \delta_j) + \delta_i \sum_{j \notin \mathfrak{W}} w_{ij} + \lambda_1 v_i + \lambda_2 v_i \tilde{x}_i = \sum_j w_{ij}(\tilde{x}_j - \tilde{x}_i) \quad i \in \{1 \ldots q\} \\ \sum_i \delta_i v_i = 0 \\ \sum_i \delta_i v_i \tilde{x}_i = 0 \quad . \end{cases} \quad (3.7)$$

Usually in a correct multilevel framework, the changes δ_i are supposed to be relatively small since the global approximation for the arrangement is inherited from the coarser levels. Their smallness is effected by the very restriction of the minimization to one window at a time. Because of the continuous formulation (3.7) of the problem within a window, the solution will almost always tend to move the vertices away from their initial ordering (adapted from the discrete arrangement). These changes may introduce some overlap between the vertices, but at the same time decrease the energy cost (3.6). It is thus expected that the δ_i's will not vanish even at the global minimum. After solving the system (3.7), every vertex $i \in \mathfrak{W}$ is thus positioned at $y_i = \tilde{x}_i + \delta_i$. Feasibility with respect to the volumes of the nodes is retained by applying (3.4). Since the size and location of $\mathfrak{W}$ are quite arbitrary, the energy cost of the new sub-arrangement is further improved by GS relaxation sweeps applied to an *enlarged* $\mathfrak{W}$, where, say 5% of the window's size at each end (if possible) are added to $\mathfrak{W}$. As usual, each sweep is followed by (3.4). The final obtained energy cost is then compared with the one prior to all the window changes, the minimum of the two is accepted, updating $\tilde{x}$. We have observed actual decrease in the energy cost in about 5% of the windows.

A sweep of WM with a given window size q consists of a sequence of overlapping windows, starting from the first node in the current arrangement and stepping by $\lfloor \frac{q}{2} \rfloor$ for each additional window. One such sweep is employed for every given q, while a small number of different q's is used (in our tests there never was a need for more than 6). Our experiments show that the algorithm is robust to changes in the chosen q's; for complete details consider $WinSizes$ in the Appendix. Note, however, that q should be small enough to still guarantee linear execution time of the entire algorithm. The WM is summarized in Algorithm 2.

Algorithm 2: WindowMinimization(graph G, current order $\tilde{x}$)

```
  Parameters:  $WinSizes$, $k_2$ (for chosen values, consider the Appendix)

  For each $q \in WinSizes$
  **For** $i = 1$ **To** $|V|$ **Step** $i = i + \lfloor \frac{q}{2} \rfloor$
      $\mathfrak{W} = \{\pi^{-1}(i), ..., \pi^{-1}(i + q - 1)\}$
      **Solve** the system of equations (3.7)
      **Apply** $k_2$ sweeps of GS relaxation on the enlarged $\mathfrak{W}$ with $\tilde{x} + \delta$
      $\tilde{x} \leftarrow \tilde{x} + \delta$ if the cost of the arrangement was decreased
**Return** $\tilde{x}$
```

Simulated Annealing

A general method to escape false local minima and advance to lower costs is to replace the strict minimization by a process that still accepts each candidate change which lowers the cost, but also assigns a positive probability for accepting a candidate step which increases the cost of the arrangement. The probability assigned to a candidate step is equal to $exp(-\Delta/T)$, where $\Delta > 0$ measures the *increase* in the arrangement cost and $T > 0$ is a temperature-like control parameter which is gradually decreased toward zero. This process, known as *Simulated Annealing* (SA) [67], in large problems would usually need to apply *very gradual* cooling (decrease of temperatures), making it extremely slow and inefficient for approaching the global optimum.

In the multilevel framework, however, the role of SA is somewhat different. At each level it is assumed that the *global* arrangement of aggregates has been inherited from the coarser levels, and thus only *local*, small-scale changes are needed. For that purpose, we have started at relatively high T, lowered it *substantially* at each subsequent sweep, until window minimization is employed.

In particular, $2k + 1$ candidate locations are examined for each vertex, each corresponds to moving it some distance l, $0 < |l| \leq k$. The initial temperature $T = T(|l|) > 0$ is calculated apriori for each distance l by aiming at the acceptance of a certain percent of changes (for instance 60%). In detail, the probability of moving a vertex l positions ($l = \pm 1, ..., \pm k$) is $Pr(l) = z \cdot min(1, exp(-\Delta(l)/T(|l|))$, where z is a normalization factor calculated by the demands $\sum_{l=-k}^{k} Pr(l) = 1$ and $Pr(0) = z \cdot min_{l=\pm 1,...,\pm k}(1 - Pr(l)/z)$. In each additional sweep $T(|l|)$ is reduced by a factor, such as 0.6. Typically only three such cooling steps are used.

Repeated heating and cooling is successively employed for better search over the local landscape. This search is further enhanced by the introduction of a "memory"-like tool consisting of an additional permutation which stores the *Best-So-Far* (BSF) observed arrangement, which is being occasionally updated by a procedure called *Lowest Common Configuration* (LCC) [18]. LCC enables the systematic accumulation of *sub*-permutations over a sequence of different arrangements, such that each BSF sub-permutation exhibits the best (minimal) sub-order encountered so far. The cost of the obtained BSF is at most the lowest cost of all the arrangements it has observed, and usually it is lower. The use of LCC becomes more important for larger graphs, where it is expected that the optimum of a subgraph is only weakly dependent on other subgraphs. Due to the LCC procedure, it is not necessary to wait in the stochastic annealing process until all minimal sub-permutations are *simultaneously* obtained, which may take exponential time; instead it is sufficient to obtain each such minimal sub-order just once, since henceforth it is guaranteed to appear in the BSF. In detail, the BSF (of a certain level) is initialized by the arrangement obtained at the end of the strict minimization. Then the BSF is improved by the LCC procedure which updates parts of it taken from the new arrangements reached right after each heating-cooling cycle. All these accumulated updates are thus stored at the BSF, which thus represents the current calculated minimum. The complete description of the LCC algorithm is given in [92].

The entire disaggregation procedure is summarized below in Algorithm 3. The Algorithm is divided into two parts: the first approximation and the postprocessing corresponding to the results supported later.

Algorithm 3: Disaggregation(coarse level $\mathcal{C}$, fine level $\mathcal{F}$)

```
Parameters: $k_1, ..., k_5, \gamma$ (for chosen values consider the Appendix.)

FIRST APPROXIMATION :
  Initialize $\mathcal{F}$ from $\mathcal{C}$
  Apply $k_1$ sweeps of compatible relaxation on $\mathcal{F}$
POSTPROCESSING :
  Apply $k_2$ sweeps of GS relaxation on $\mathcal{F}$
  Apply Window Minimization on $\mathcal{F}$
  Initialize $BSF \leftarrow$ current arrangement of $\mathcal{F}$
  Do $k_3$ cycles of heating and cooling
      Calculate $T(|l|)$ for $l = 1, ..., k_4$
      Do $k_5$ steps
          Apply SA within distance $k_4$ on $\mathcal{F}$
          Decrease all $T(|l|)$ by a factor $\gamma$
      Apply Window Minimization on $\mathcal{F}$
      $BSF = LCC(BSF,$ current arrangement of $\mathcal{F})$
Return $BSF$
```

3.4 Results and Related Works

We have implemented and tested the algorithm using standard C++, LAPACK++ [84] and LEDA libraries [74] on Linux 2.4GHz machine. The implementation is non-parallel and has not been optimized. The results (order costs and running times) should only be considered qualitatively and can certainly be further improved by more advanced implementation.

We have found only one article [50] with an implemented algorithm and numerical results for M2sP. The algorithm is based on the spectral approach. Since this test suite is relatively small to provide enough information regarding M2sP, we have launched a new, much larger test suite which consists of 66 graphs from different areas [37, 77], see Table 3.1. These graphs are divided into two groups according to their size : the results for the smaller ones are introduced in Tables 3.2 and 3.3, while those for the larger ones in Table 3.4. For all the graphs in Tables 3.2 and 3.3 we compare our results with those of the spectral approach. The numbers in columns 4-5 (marked by "**ML**" and "**ML+GS**") and 7-11 are in percentage above the cost energy presented at the column "**Quick**" (e.g., the 0.8 appearing for the first graph gd96c in column "**ML**" means that the initial cost energy is $3455 \cdot 1.008$). The first approximation obtained by the multilevel V-cycle, i.e., the arrangement obtained right after applying the compatible relaxation at the finest level is introduced in the column "**ML**" of Tables 3.2 and 3.3. We run the algorithm 100 times (using the parameters specified in the Appendix for the "**Quick**" V-cycle), each starts from a different permutation of the nodes. The best obtained results are presented here. The means of the 100 runs are worse than the corresponding "**Quick**"-values by an average of 0.51%, while the standard deviation (around the means) is 0.69% on the average. The "**ML**" results should be compared to the spectral approach results at column "**SP**" obtained by calculating the second eigenvector of the Laplacian[1] of the graph using MATLAB routine. It shows that the "**ML**" algorithm provides much better results, shorter by an average of +34.4% (excluding the statistics of bintree10, in which the improvement is much larger). Only in one case, the 10-dimensional hypercube, the spectral approach provided a lower cost of -2.8%. However, not only the obtained results are much worse. Even if the spectral method leads to the correct order, the calculations must be performed with very high accuracy. In fact, the precision of the second eigenvector coordinates must be at least $O(\log |V|)$ and usually much better. This is not a trivial task while one uses some approximation algorithm. Our results for the spectral approach were thus obtained with 16-digits precision of an exact algorithm. The experiments with lower precision or with approximation algorithms gave much poorer results. For example, for the three graphs bcspwr10, airfoil1 and bcsstk38 the spectral costs with 5-digits precision were 4.40E+07, 4.49E+07 and 1.20E+10 respectively, while increasing the precision to 7-digits gave 1.64E+07, 1.93E+07 and 4.40E+09. The complexity of an exact calculation of the second smallest eigenvector is $O(|V|^3)$ while the multilevel algorithm is linear in the number of edges.

We have next tested the outcome of our postprocessing on both initial sets of

[1]The algebraic representation of a graph is given by its *Laplacian* A (a $|V| \times |V|$ matrix), whose terms are defined by

$$A_{ij} = \begin{cases} -w_{ij} & \text{for } ij \in E,\ i \neq j \\ 0 & \text{for } ij \notin E,\ i \neq j \\ \sum_{k\neq i} w_{ik} & \text{for } i = j \end{cases} . \tag{3.8}$$

results. Most significant improvement was introduced by applying the Gauss-Seidel-like relaxation, as can be seen in Tables 3.2 and 3.3 column "**ML+GS**" for the multilevel algorithm and "**+GS**" for the spectral approach. The gap between the two has been reduced, but the spectral approach still provides worse results on the average by 7.1%. Next we have applied the window minimization which concludes our, so called "**Quick**" V-cycle. Comparing columns "**Quick**" with the corresponding "**+WM**" shows that the multilevel results remain better, the spectral ones are worse by an average of 4.7%.

Finally, we introduced randomness by applying Simulated Annealing. In the multilevel framework, the SA enters at all levels of the V-cycle. We refer to this version as the "**Extended**" V-cycle (its complete parameters are given in the Appendix). While the "**Quick**" V-cycle is aimed at achieving fast performance, the "**Extended**" V-cycle runs longer but succeeds in finding lower cost arrangements on the average by 1%. The means of the 100 runs of the "**Extended**" V-cycles are worse than the corresponding "**Quick**"-values by an average of 0.49% and the average of the calculated standard deviations (around the means for 100 runs) of the "**Extended**" V-cycle is 0.66%. We may conclude that the "**Extended**" V-cycle is not really needed. Almost identical results are already obtained by the "**Quick**" V-cycle, where the improvement is neither significant nor consistent, i.e., it is just within the typical standard deviation. In column "**+SA**" of Tables 3.2 and 3.3 we present the results obtained after adding SA to the spectral approach followed by the above postprocessing. The improvement is again of only 1%. Our last test was to run a very long SA after the postprocessing with the spectral approach, aiming at achieving comparable amount of work to 100 "**Extended**" V-cycles. These results are given in column "**HSA**". While improvement is naturally observed, the results on the average remain worse by about 2%, while for 6 graphs out of 37 it is worse by more than 5%.

In addition, we present the spectral lower bounds [50] for the smaller graphs (see Tables 3.2 and 3.3, column "**LB**") and calculate the gap (see Tables 3.2 and 3.3, column "$\mathbf{\Delta_{Quick}}$") between our results and the spectral lower bound. In spite of the fact that it is impossible to judge which costs are closer to the real minima, we may state that no significant indications of the existence of these lower costs were observed: in 16 out of 37 graphs our results were within 25% of the lower bounds, but on the average they were 75.4% longer.

To enrich our test suite, we present in Table 3.4 our "**Quick**" V-cycle results for additional 29 relatively large graphs. No spectral approach results are provided since we were not able to run (on the computers available to us) the MATLAB routine and calculate the needed eigenvector. Each result is again the best observed out of 100 runs, for which the means for 100 runs are worse than the corresponding "**Quick**"-values by an average of 0.55% and the average standard deviation is 0.47%.

The running time of the algorithm is presented in Table 3.6. In column "$\mathbf{T_{Quick}}$" we present the running time of one V-cycle which corresponds to the "**Quick**" column in Tables 3.2, 3.3 and 3.4. The running time of the suggested postprocessing added after the spectral ordering was measured for the graphs from Tables 3.2, 3.3 and we show it in column "$\mathbf{T_{Post}}$". The running time of the postprocessing corresponds to the values introduced in the "**+SA**" column of Tables 3.2 and 3.3. The dash notation ("-") corresponds to the graphs from Table 3.4 that were too large for usual MATLAB spectral calculation routines. In both cases the running time is measured in minutes. The star notation ("*") can be interpreted as "less than one second".

3.5 Conclusions

We have presented a multilevel algorithm for the minimum p-sum problem for general graphs. The algorithm is based on the general principle that during coarsening each vertex may be associated to more than just one aggregate according to some "likelihood" measure. The uncoarsening initialization, which produces the first arrangement of the fine graph nodes, strongly relies on energy considerations (unlike usual interpolation in classical AMG). This initial order is further improved by Gauss-Seidel-like relaxation, window minimization and possibly by employing randomness, i.e., simulated annealing. The running time of the algorithm is linear, thus it can be applied to very large graphs.

We have compared our results to those obtained by the spectral approach. The calculation of the second eigenvector of the Laplacian of the graph has to be of high accuracy to provide reasonable results. Such a direct computation is of complexity $O(|V|^3)$. Still, the obtained results are much worse than the initial results obtained by our multilevel V-cycle by 34.4% on the average for the smaller sized test suite. In addition, we have applied postprocessing to both initial arrangements. The Gauss-Seidel-like relaxation improves both results most significantly. The window minimization further reduces the arrangement cost for some graphs. The final results show that the multilevel framework achieves better results of 4.7% on the average. Finally, we have added stochastisity to both algorithms. Both results were improved by about 1%. We have also tried to apply a very long SA to the final results of the postprocessing of the spectral approach. Many results have been further improved, however, some graphs (6 out of 37) still present results higher by more than 5%.

Our main conclusion is that the average and the best results of our V-cycles are better than the results of the spectral approach. We recommend our multilevel algorithm as a general practical method for solving the minimum p-sum problem and as a fast and high-quality method for obtaining first approximation for it. The implemented algorithm can be obtained at *http://www.wisdom.weizmann.ac.il/∼safro/min2sum*.

Appendix: Parameters

In order to control the running time of the algorithm it is important to decrease the total number of edges of the constructed coarse graphs. This is achieved by the following two parameters: the maximum allowed coarse neighborhood size r, which restricts the allowed size $|N_i|$ of the coarse neighborhood of a vertex $i \in F$ by deleting the weakest w_{ij}, $j \in C$; and the edge filtering ϵ threshold, which deletes every *relatively* weak edge ij (from the created coarse graph) if both $w_{ij} < \epsilon \cdot s_i$ and $w_{ij} < \epsilon \cdot s_j$, where $s_i = \sum_k w_{ik}$.

These two parameters and five others which control the uncoarsening procedure (see Algorithm 3) are presented in Table 3.5 for the "**Quick**" and "**Extended**" V-cycles we have used. The last two parameters within the SA (of Algorithm 3) were constantly chosen to be $k_5 = 4$ and $\gamma = 0.6$.

It is however important to mention that these parameters are the ones used only for the finest levels. As the coarse graphs become much smaller they are accordingly increased. This hardly affects the entire running time of the algorithm but systematically improves the obtained results. In the last column of Table 3.5 we specifically describe the increase introduced for each parameter as a function of level L, which usually depends on the ratio $R = max(1, |E_0|/|E_L|)$ measuring the relative decrease of the number of edges at level L compared with the original graph.

We tested many options for the window sizes in Algorithm 2. Usually these sizes were relatively small and very robust to changes. In our implementation we used $WinSizes = \{5, 10, 15, 20, 25, 30\}$, however similar results were obtained with other sets of windows, for example, $WinSizes = \{5, 9, 17, 23, 29\}$.

Acknowledgments

This research was supported by a Grant from the German-Israeli Foundation for Scientific Research and Development (G.I.F.), Research Grant Agreement No. I-718-135.6/2001, and by the Carl F. Gauss Minerva Center for Scientific Computation at the Weizmann Institute of Science.

Table 3.1: Benchmark for the minimum 2-sum problem.

Graph	**|V|**	**|E|**	**Graph**	**|V|**	**|E|**
gd96c	65	125	**nasa1824**	1824	18692
gd95c	62	144	**randomA2**	1000	24738
gd96b	111	193	**nasa2146**	2146	35052
gd96d	180	228	**bcsstk13**	2003	40940
dwt245	245	608	**whitaker3**	9800	28989
bintree10	1023	1022	**zcrack**	10240	30380
bus685	685	1282	**shuttleeddy**	10429	46585
bus1138	1138	1458	**randomA3**	1000	49820
gd96a	1096	1676	**nasa4704**	4704	50026
can445	445	1682	**bcsstk24**	3562	78174
c1y	828	1749	**bcsstk38**	8032	173714
c2y	980	2102	**finan512**	74752	261120
bcspwr08	1624	2213	**bcsstk33**	8738	291583
bcspwr09	1723	2394	**bcsstk29**	13830	302424
c5y	1202	2557	**ocean**	143437	409593
jagmesh1	936	2664	**tooth**	78136	452591
c3y	1327	2844	**mrng1**	257000	505048
c4y	1366	2915	**bcsstk37**	25503	557737
dwt918	918	3233	**msc23052**	23052	559817
dwt1007	1007	3784	**bcsstk36**	23052	560044
jagmesh9	1349	3876	**bcsstk31**	35586	572913
can838	838	4586	**msc10848**	10848	609464
randomA1	1000	4974	**ferotor**	99617	662431
hc10	1024	5120	**bcsstk35**	30237	709963
can1054	1054	5571	**598a**	110971	741934
can1072	1072	5686	**bcsstk32**	44609	985046
randomG4	1000	8173	**bcsstk30**	28924	1007284
randomA4	1000	8177	**144**	144649	1074393
bcspwr10	5300	8271	**ct20stif**	52329	1273983
bcsstm13	649	9949	**m14b**	214765	1679018
dwt2680	2680	11173	**mrng2**	1017253	2015714
airfoil1	4253	12289	**auto**	448695	3314611
bcsstk12	1423	16342	**pwtk**	217918	5653257

Table 3.2: Results (small graphs).

Graph	**LB**	$\Delta_{\textbf{Quick}}$	**ML**	**ML+GS**	**Quick**	**SP**	**+GS**	**+WM**	**+SA**	**HSA**
gd96c	1.25E+03	176.9	0.8	0.3	3.45500E+03	46.0	5.8	0.0	0.0	0.0
gd95c	1.13E+03	232.9	0.8	0.0	3.75500E+03	26.2	0.3	0.0	0.0	0.0
gd96b	2.43E+03	684.8	8.7	0.0	1.90860E+04	53.7	1.3	1.1	1.0	1.0
gd96d	3.74E+04	46.3	8.2	1.0	5.47390E+04	87.0	1.0	0.2	0.1	0.0
dwt245	4.34E+04	45.8	2.9	0.4	6.32810E+04	80.4	2.6	0.8	0.8	0.0
bintree10	8.85E+04	53.2	11.2	0.0	1.35656E+05	16394.2	38.8	11.3	10.3	6.4
bus685	1.43E+05	50.8	9.6	0.2	2.15744E+05	44.9	9.8	7.0	7.0	6.6
bus1138	4.00E+05	38.0	6.7	0.4	5.52111E+05	76.5	7.4	1.8	0.9	0.4
gd96a	2.83E+06	430.0	13.6	0.1	1.49741E+07	124.4	31.3	25.2	21.9	15.9
can445	1.57E+06	5.0	1.2	0.0	1.65431E+06	6.0	0.8	0.6	0.6	0.6
c1y	5.56E+06	41.4	8.2	0.0	7.86685E+06	121.1	5.6	4.6	4.3	4.2
c2y	8.74E+06	22.7	7.3	0.0	1.07286E+07	61.2	0.9	0.6	0.3	0.3
bcspwr08	7.97E+05	17.8	5.6	0.4	9.39437E+05	48.6	13.0	11.2	9.1	1.9
bcspwr09	7.91E+05	28.7	5.8	0.5	1.01801E+06	71.8	25.5	22.5	22.0	11.3
c5y	1.16E+07	21.2	7.2	0.0	1.39958E+07	130.5	10.8	9.6	8.4	6.4
jagmesh1	8.27E+05	5.1	2.5	0.1	8.68459E+05	14.2	12.6	12.2	11.8	1.1
c3y	1.55E+07	27.0	7.0	0.0	1.97321E+07	142.7	7.5	2.5	0.8	0.7
c4y	1.44E+07	15.0	8.2	0.0	1.66028E+07	51.8	2.1	1.4	0.2	0.0
dwt918	5.46E+05	51.0	5.1	0.1	8.25233E+05	11.5	1.4	0.9	0.3	0.0
dwt1007	8.86E+05	15.9	1.4	0.0	1.02750E+06	4.3	2.2	1.9	1.7	0.0
(continued)										

Table 3.3: Results (small graphs) - continuation. The average results are calculated for all the 37 graphs of Tables 3.2 and 3.3.

Graph	**LB**	Δ_{Quick}	**ML**	**ML+GS**	**Quick**	**SP**	**+GS**	**+WM**	**+SA**	**HSA**
jagmesh9	1.10E+06	27.4	5.2	0.2	1.39541E+06	10.3	6.3	4.5	1.6	0.9
can838	7.02E+06	5.8	0.4	0.0	7.43012E+06	1.8	0.1	0.1	0.0	0.0
randomA1	7.03E+07	321.8	34.9	1.8	2.96618E+08	49.7	18.2	9.7	4.9	1.1
hc10	1.79E+08	0.0	3.6	0.0	1.78957E+08	0.8	0.1	0.1	0.0	0.0
can1054	5.79E+06	9.9	0.2	0.1	6.36257E+06	2.0	0.1	0.1	0.0	0.0
can1072	8.17E+06	6.5	3.6	0.0	8.70400E+06	3.6	0.1	0.0	0.0	0.0
randomG4	7.33E+06	5.0	6.9	0.0	7.70221E+06	7.8	1.1	0.8	0.6	0.1
randomA4	3.01E+08	125.6	18.3	4.3	6.78008E+08	31.2	15.3	5.7	0.9	0.4
bcspwr10	1.19E+07	15.0	10.5	0.2	1.37238E+07	19.0	4.9	4.1	3.0	2.3
bcsstm13	2.23E+07	77.2	0.5	0.0	3.94573E+07	31.7	0.8	0.6	0.3	0.0
dwt2680	7.31E+06	25.6	4.2	0.0	9.18901E+06	5.2	0.3	0.1	0.0	0.0
airfoil1	1.18E+07	37.9	8.9	0.1	1.63343E+07	18.4	7.2	5.9	2.7	1.1
bcsstk12	1.71E+07	20.6	7.7	0.1	2.06281E+07	19.2	11.9	10.2	6.8	5.9
nasa1824	1.37E+08	3.1	5.8	0.0	1.41216E+08	24.0	7.9	4.2	1.1	0.3
randomA2	2.21E+09	33.5	12.8	4.1	2.95112E+09	12.9	5.1	0.3	0.1	0.0
nasa2146	1.11E+08	11.3	5.3	0.1	1.23584E+08	6.6	4.3	4.2	4.0	2.1
bcsstk13	4.35E+08	54.4	3.4	0.0	6.71461E+08	40.2	14.8	9.3	3.0	2.7
AVERAGE		75.4	6.9	0.4		41.3	7.5	4.7	3.5	2.0

Table 3.4: Results (large graphs).

Graph	**Quick**	**Graph**	**Quick**
whitaker3	6.53774E+07	**msc23052**	6.58277E+10
zcrack	1.36390E+08	**bcsstk36**	6.58053E+10
shuttleeddy	1.36200E+08	**bcsstk31**	7.45410E+10
randomA3	6.63612E+09	**msc10848**	5.95150E+10
nasa4704	7.54695E+08	**ferotor**	2.67776E+11
bcsstk24	9.06089E+08	**bcsstk35**	7.51880E+10
bcsstk38	3.87606E+09	**598a**	3.85388E+11
finan512	1.00967E+10	**bcsstk32**	1.46284E+11
bcsstk33	2.97010E+10	**bcsstk30**	5.11256E+10
bcsstk29	1.06444E+10	**144**	1.55347E+12
ocean	1.16999E+11	**ct20stif**	6.77425E+11
tooth	3.38761E+11	**m14b**	1.67209E+12
mrng1	6.69398E+11	**mrng2**	1.93775E+13
bcsstk37	6.77934E+10	**auto**	1.33598E+13
		pwtk	2.25527E+12

Table 3.5: The parameters used for the "**Quick**" and "**Extended**" V-cycles.

Parameter	"**Quick**" V-cycle	"**Extended**" V-cycle	The increase for level L
The coarse neighborhood size (r)	10	10	$+log(R)$
The edge filtering threshold (ϵ)	0.001	0.001	$\cdot 0.9^{log(R)}$
Compatible relaxation sweeps (k_1)	5	10	$+2 \cdot L$
GS relaxation sweeps (k_2)	5	10	$+2 \cdot L$
Heating and cooling in SA (k_3)	0	3	$\cdot log(R)$
k_4 used in the SA	0	5	$+log(\sqrt{R})$

Table 3.6: Running time

Graph	$\mathbf{T_{Quick}}$	$\mathbf{T_{Post}}$	**Graph**	$\mathbf{T_{Quick}}$	$\mathbf{T_{Post}}$
gd96c	*	*	**nasa1824**	0.05	0.17
gd95c	*	*	**randomA2**	0.26	0.27
gd96b	*	*	**nasa2146**	0.1	0.38
gd96d	*	*	**bcsstk13**	0.1	0.44
dwt245	*	0.01	**whitaker3**	0.15	-
bintree10	*	0.02	**zcrack**	0.13	-
bus685	*	0.02	**shuttleeddy**	0.15	-
bus1138	0.05	0.04	**randomA3**	0.42	-
gd96a	0.04	0.05	**nasa4704**	0.11	-
can445	*	0.02	**bcsstk24**	0.14	-
c1y	*	0.02	**bcsstk38**	0.3	-
c2y	0.02	0.03	**finan512**	1.2	-
bcspwr08	0.04	0.05	**bcsstk33**	0.65	-
bcspwr09	0.05	0.06	**bcsstk29**	0.6	-
c5y	0.02	0.06	**ocean**	7.5	-
jagmesh1	0.04	0.04	**tooth**	2.5	-
c3y	0.02	0.05	**mrng1**	12.3	-
c4y	0.02	0.05	**bcsstk37**	1.3	-
dwt918	*	0.04	**msc23052**	1.25	-
dwt1007	*	0.04	**bcsstk36**	1.3	-
jagmesh9	0.05	0.07	**bcsstk31**	1.9	-
can838	*	0.04	**msc10848**	1.1	-
randomA1	0.1	0.11	**ferotor**	4.7	-
hc10	0.065	0.06	**bcsstk35**	1.6	-
can1054	*	0.07	**598a**	6.4	-
can1072	*	0.05	**bcsstk32**	2.9	-
randomG4	0.02	0.07	**bcsstk30**	2	-
randomA4	0.15	0.18	**144**	10.3	-
bcspwr10	0.09	0.17	**ct20stif**	4.7	-
bcsstm13	0.04	0.14	**m14b**	17.5	-
dwt2680	0.05	0.12	**mrng2**	143	-
airfoil1	0.06	0.19	**auto**	64.3	-
bcsstk12	0.05	0.17	**pwtk**	20	-

CHAPTER 4

Multilevel algorithms for linear ordering problems

4.1 Introduction

The objective of the class of linear ordering problems is to minimize different functionals that map the set of the graph vertices onto $(1, 2, ..., n)$. This class contains many graph (or matrix) layout problems such as : the minimum p-sum, the workbound reduction, the wavefront, the envelope size, etc. Some problems, such as finding the minimum linear arrangement [92] or the bandwidth [70], appear in many applications for solving problems in the large sparse matrix computation. Some other are closely related to the problem of calculating the envelope size of a symmetric matrix or, more precisely, to the amount of work needed in the Cholesky factorization of such a matrix [50]. Linear ordering problems may also be motivated as a model used in VLSI design [30] and may be used in several biological applications, graph drawing and other fields (see [39, 70, 57, 96]). Commonly for general graphs (or matrices) these problems are NP-hard and their decision versions are NP-complete [48].

Since these problems have a practical significance, many heuristic algorithms were developed in order to achieve near optimal solution. Among the most successful are spectral sequencing [62], optimally oriented decomposition tree [6], multilevel based [68, 59], simulated annealing [79] and others. Some of these algorithms have proven themselves superior in solution quality while others in execution time.

One of the most popular and exploitable methods designed to achieve a suitable linear ordering for different problems is the spectral sequencing (SS) [62]. This approach consists of ordering the graph vertices according to the sorted coordinates of the second eigenvector of the graph Laplacian. The heuristic argumentation of SS is based on the fact that the *continuous* version of the minimum 2-sum problem can be solved by this method to the optimum [62]. In practice, for the (discrete) minimum 2-sum it was shown in [93] that the direct application of SS (without additional reinforcement postprocessing) on "real world instances" does not achieve good enough results, while the lower bounds based on SS are very far from the best known ordering costs. Rather poor results of the *exact* SS were presented in [35] for the minimum bandwidth problem. Better results were shown there by using different *approximated* SS, i.e., by calculating the second eigenvector *less* precisely. In fact, they have tested 19 algorithms (17 of which are different versions of SS) and presented the best achieved results among all. In Section 4.4 we show the significant improvement achieved by our algorithm over all those algorithms, on the average our results were better by 34%.

In this paper we present a general framework of multilevel algorithms especially designed for linear ordering problems. Our strategy is based on the Algebraic Multi-Grid scheme (AMG) [15, 16, 12, 22, 89, 99, 100]. While in previous works we have developed and tested special multilevel algorithms for solving the minimum linear arrangement problem [92] and the minimum 2-sum problem [93], in this article we demonstrate how the building blocks of the general multilevel approach can be used in various ways to make it suitable for solving more involved functionals. In particular, we present two algorithms : we show how the *bandwidth* of a graph can be approximated by a continuation approach in which a sequence of increasingly p-sum problems are involved until p is large enough to be considered infinite for practical purposes; in addition, we use the minimum 2-sum problem result as a first approximation for the *workbound* reduction problem, which is then improved by a postprocessing of local minimizations with actual use of the workbound functional. In fact, we propose to use the ordering obtained by the minimum 2-sum problem as a first approximation for other linear ordering problems, as demonstrated for the *wavefront reduction* problem.

The main objective of a multilevel based algorithm is to create a hierarchy of problems, each representing the original problem, but with fewer degrees of freedom. General multilevel techniques have been successfully applied to various areas of science (e.g. physics, chemistry, engineering, etc.) [14, 17]. AMG methods were originally developed for solving linear systems of equations resulting from the discretization of partial differential equations. Lately they have been applied to various other fields, yielding for example novel methods for image segmentation [97] and for the linear arrangement problem [92]. In the context of graphs it is the Laplacian matrix that represents the related set of equations. The main difference between our approach to most other multilevel approaches (related to various graph optimization problems) is the coarsening scheme. While the previous approaches may be viewed as *strict* aggregation process, the AMG coarsening is actually a *weighted* aggregation : each node may be divided into *fractions*, and different fractions belong to different aggregates. This enables more freedom in solving the coarser levels and avoids making hardened local decisions, such as edge contractions, before accumulating the relevant global information.

One of the important achievements of our work is the general coarsening that turns to be suitable for all the different functionals we have tested. This fact can be explained by the way the hierarchy of problems is constructed: variables are eliminated within the coarsening phase only and exactly when they show strong dominant connections to the remaining (non-eliminated) variables, this in turn guarantees that the solution of the eliminated variables is naturally obtained once the non-eliminated variables are solved. The various algorithms thus differ in the disaggregation process which follows by projecting to a finer level the final arrangement obtained on a coarser level. This initial fine level arrangement is being further improved by applying different local reordering methods. We have developed a simultaneous minimization of several vertices called Window Minimization. In its basic application (for the 2-sum problem [93]) it involves the minimization of a quadratic form. Here we show how to quadratize other functionals. Also, we suggest the use of numerical calculation rather than analytic, for instance, in calculating derivatives. Finally, our postprocessing is intensified by Simulated Annealing (SA) [67] which is a general method to escape local minima. In the multilevel framework SA is aimed at searching only for *local* changes that guarantee the preservation of large-scale solution features inherited from coarser levels.

We will not discuss here theoretical complexity issues, such as lower and upper bounds for the solution cost. We are not interested in worst possible scenarios nor in random instances. Our focus is on practical high-performance and low computational cost algorithms that will outperform existing algorithms by providing better results in less running time. For that purpose we used a known benchmark [37] from which we took graphs of various origins and sizes including very large instances. Our multilevel algorithm exhibit linear complexity, i.e., the computational cost is proportional to $|V| + |E|$.

We compared the results obtained by our multilevel algorithms with many previously described algorithms. In this paper we present the results of the bandwidth problem and the workbound problem and show that our results are on the average better than previous ones by about 30%, while the running time for graphs with about 10^4 nodes and 10^5 edges is less than one minute. In general, our experimental results show that the AMG framework can be used for linear ordering problems to obtain high quality results in linear time while using the exact same set of parameters. The implemented algorithm can be downloaded from [90].

The various functionals and their generalizations are described in Sec. 2. The multilevel algorithm along with additional optimization techniques are presented in Sec. 3. A comparison of our results with other works is finally summarized in Sec. 4.

4.2 Definitions and generalizations

Given a weighted graph $G = (V,\ E)$, where $V = \{1, 2, ..., n\}$, denote by w_{ij} the non-negative weight of the edge ij between nodes i and j; if $ij \notin E$ then $w_{ij} = 0$.

Let π be a bijection

$$\pi : V \longrightarrow (1, 2, ..., n) \quad .$$

The purpose of linear ordering problems is to minimize some functional over all possible permutations π. The following functional should be minimized for the minimum p-sum problem[1] :

$$\sigma_p(G,\pi) = \sum_{ij} w_{ij} |\pi(i) - \pi(j)|^p \quad . \tag{4.1}$$

In the generalized form of the problem that emerges during the multilevel solver, each vertex i is assigned with a *volume* (or *length*), denoted v_i. The task now is to minimize the cost $\sigma_p(G,x) = \sum_{ij} w_{ij}|x_i - x_j|^p$, where $x_i = \frac{v_i}{2} + \sum_{k,\pi(k)<\pi(i)} v_k$, i.e., each vertex is positioned at its center of mass capturing a segment on the real axis which equals its length. The original form of the problem is the special case where all the volumes are equal. In particular, we would like to concentrate on the minimum bandwidth problem which seeks a linear layout that minimizes the maximal stretched edge, i.e., $bw(G) = \min_\pi \max_{ij} w_{ij}|\pi(i)-\pi(j)|$. The minimization functional of the bandwidth problem can be formulated in term of $\sigma_p(G,\pi)$:

$$bw(G,\pi) = \lim_{p\to\infty} (\sigma_p(G,\pi))^{1/p} \quad . \tag{4.2}$$

The minimization functional of the workbound reduction problem is defined as

$$wb(G,\pi) = \sum_i \max_{\substack{j \\ \pi(j)<\pi(i)}} w_{ij}(\pi(i) - \pi(j))^2 \quad . \tag{4.3}$$

The generalized form of this problem is similar to the above derivation, and the max function may be approximated by

$$wb(G,x) = \sum_i \max_{j:x_j<x_i} w_{ij}(x_i - x_j)^2 \approx \sum_i \Big(\sum_{j:x_j<x_i} w_{ij}(x_i - x_j)^p \Big)^{2/p} \quad . \tag{4.4}$$

4.3 The algorithm

In the multilevel framework a hierarchy of decreasing size graphs : $G_0, G_1, ..., G_k$ is constructed. Starting from the given graph, $G_0 = G$, create by recursive *coarsening* the sequence $G_1, ..., G_k$, then solve the coarsest level G_k directly, and finally uncoarsen the solution back to G. This entire process is called a *V-cycle*. As in the general AMG setting, the choice of the coarse variables (aggregates), the derivation of the coarse problem which approximates the fine one and the design of the coarse-to-fine disaggregation (uncoarsening) process are all determined automatically, as described below.

[1]We use this definition for simplicity, while the usual definition of the functional is $\sigma_p(G,\pi) = (\sum_{ij} w_{ij}|\pi(i) - \pi(j)|^p)^{1/p}$, which yields of course the same minimization problem.

4.3.1 Coarsening: Weighted Aggregation

The coarsening used here is similar to the process we have used in solving the minimum linear arrangement and the minimum 2-sum problems [92, 93]. For completeness we briefly repeat its description. The coarsening is interpreted as a process of *weighted aggregation* of the graph nodes to define the nodes of the next coarser graph. In a *strict* aggregation process (also called edge contraction or matching of vertices) the nodes are blocked in small disjoint subsets, called aggregates. Two nodes i and j would usually be blocked together (put in the same aggregate) if their coupling is *strong*, meaning that w_{ij} is comparable to $min\{max_k w_{ik}, max_k w_{kj}\}$. In *weighted* aggregation, each node can be divided into $fractions$, and different fractions belong to different aggregates. In both cases, these aggregates will form the nodes of the *coarser level*, where they will be blocked into larger aggregates, forming the nodes of a *still coarser level*, and so on. As AMG solvers have shown, *weighted*, instead of *strict*, aggregation is important in order to express the *likelihood* of nodes to belong together; these likelihoods will then accumulate at the coarser levels of the process, indicating tendencies of larger scale aggregates to be associated to each other (see [100] for a deep explanation). Strict aggregation, by contrast, may run into a conflict between the local blocking decision and the larger-scale picture.

The construction of a coarse graph from a given one is divided into three stages: first a subset of the fine nodes is chosen to serve as the *seeds* of the aggregates (which become the nodes of the coarse graph), then the rules for interpolation are determined, thereby establishing the fraction of each non-seed node belonging to each aggregate, and finally the strength (or weight) of the connections (or edges) between the coarse nodes is calculated.

Coarse Nodes. The construction of the set of seeds C and its complement, denoted by F, is guided by the principle that each F-node should be "strongly coupled" to C. Also, we will include in C nodes with exceptionally large volume, or nodes expected (if used as seeds) to aggregate around them exceptionally large volumes of F-nodes. To achieve these objectives, we start with an empty set C, hence $F = V$, and then sequentially transfer nodes from F to C until all remaining $i \in F$ satisfy

$$\sum_{j \in C} w_{ij} / \sum_{j \in V} w_{ij} \geq Q \quad ,$$

where Q is a parameter; $Q = 0.4$ is used in the reported experiments.

Note that it is thus guaranteed that every F-node has strong dominant connections to the C-nodes which are uniquely associated to the coarse aggregates. This in turn means that once an order of the desired functional is obtained among the aggregates, an initial order of a finer level naturally follows (see Section 4.3.3). This reasoning explains why the same coarsening is successful for the various functionals.

The Coarse Problem. Each node in the chosen set C becomes the seed of an aggregate that will constitute one coarse level node. Define for each $i \in F$ a coarse neighborhood $N_i = \{j \in C,\ w_{ij} \geq \alpha_i\}$, where α_i is determined by the demand that $|N_i|$ does not exceed the allowed coarse neighborhood size r chosen to control complexity. (For typical values of r consider the Appendix). The classical AMG interpolation matrix P (of size $|V| \times |C|$) is defined by

$$P_{ij} = \begin{cases} w_{ij} / \sum\limits_{k \in N_i} w_{ik} & \text{for } i \in F,\ j \in N_i \\ 1 & \text{for } i \in C,\ j = i \\ 0 & \text{otherwise} \quad . \end{cases} \tag{4.5}$$

P_{ij} thus represents the likelihood of i to belong to the j-th aggregate. Let $I(k)$ be the ordinal number in the coarse graph of the node that represents the aggregate around a seed whose ordinal number at the fine level is k. Following the weighted aggregation scheme used in [97], the edge connecting two coarse aggregates, $p = I(i)$ and $q = I(j)$, is assigned with the weight $w_{pq}^{(coarse)} = \sum_{k \neq l} P_{ki} w_{kl} P_{lj}$. The volume of the i-th coarse aggregate is $\sum_j v_j P_{ji}$. Note that during the process of coarsening the total volume of all vertices is conserved.

4.3.2 The coarsest level

Minimizing the appropriate functional at the coarsest level, which consists of no more than 8 nodes (otherwise a still coarser level would be introduced for efficiency) is performed directly by simply trying all possible arrangements. Since the amount of work invested at the coarsest levels is small compared with that of the finest level, many solutions can in fact be kept at each level whose graph is small relative to G. In principle, this number depends on the amount of work associated with the graph parameters of that level. In particular, a large number of solutions is chosen at the coarsest level; they are chosen so that they all enjoy a relatively low energy cost and are mutually significantly different from each other. Each is then propagated to the next finer level and being optimized there. The best solutions are chosen using the same criteria, and so on. This variety of solutions enlarges the range of the search by either extracting different best solutions or combining them using LCC [92].

Since we wanted to measure the standard deviation for our algorithm, we have run it a few times for each of the given graphs by starting with a different permutation of the nodes of G (see Section 4.4.2). Experiments show that the variety of solutions generated thus is similar to those obtained by a single run with multitude of solutions at the coarsest levels, thus it became less important to also use the later. Still this approach has proven to work well for [87].

4.3.3 Disaggregation (uncoarsening)

While the same identical coarsening procedure was used for the minimization of all our functionals, the uncoarsening only shares the same basic structure, but the actual implementation varies from one functional to another. Having solved a coarse problem, the solution to the next-finer-level problem is initialized by first placing the seeds according to the coarse order and then adjusting all other F-nodes while aiming at the minimization of the arrangement cost. This first approximation is subsequently improved by several *relaxation* sweeps, first compatible, then regular (explained below). Then, the arrangement is improved by strict minimization, possibly with added stochasticity. These are the local reordering processes which either accept only changes that decrease the arrangement cost (strict minimization) or might also accept steps which increase the cost (with some probability) in order to escape false local minima (simulated annealing). The entire scheme is explained below and summarized in Algorithm 2.

Before we turn to the details of these common stages of the disaggregation process, let us describe the particular structure we have used for the minimum p-sum problem. The disaggregation scheme for the minimization of $\sigma_p(G, x)$ is based on *continuation* in the parameter p, such that $p = 2$ is used to exactly solve the coarsest level, and then, at each subsequent finer level, p is increased (e.g. by two). Thus, every level l (other than the coarsest) minimizes $\sigma_p(G_l, x)$ by initialization from $\sigma_{p-2}(G_{l+1}, x)$. Except that in cases where the desired p is already reached on one of the coarse levels, no further continuation is employed beyond that level. Our experiments show that the results are not sensitive to small changes in the continuation of p, e.g., solving the coarsest level with $p = 4$, or increasing p by four. In case where p should tend to infinity (as for the bandwidth (4.2)), the increase of p is continued also at the end of the V-cycle in a *postprocessing* procedure.

Initialization of the next finer level

Given is the arrangement of the coarse level aggregates in its generalized form, where the center of mass of each aggregate $j \in C$ is positioned at $x_{I(j)}$ along the real axis. We begin the initialization of the fine level arrangement by letting each seed $j \in C$ inherit the position of its respective aggregate: $y_j = x_{I(j)}$. At each stage of the initialization procedure, define $V' \subset V$ to be the subset of nodes that have already been placed, so we start with $V' = C$. Then proceed by positioning each fine node $i \in V \setminus V'$ at the coordinate y_i in which the cost of the arrangement, at that moment when i is being placed, is minimized. The sequence in which the nodes are placed is roughly in decreasing order of their *relative* connection to V', that is, the nodes which have strong connections to V' compared with their connections to V are placed first. To be precise, for the minimum p-sum problem the coordinate y_i is located at its minimum (volumes are not taken into account)

- if $p = 1$ then $y_i \in \{y \ : \quad |\sum_{y_j<y,\ j\in V'} w_{ij} - \sum_{y_j>y,\ j\in V'} w_{ij}| \quad \text{is minimal}\}$, i.e., y_i is within the median segment,
- if $p = 2$ then $y_i = \frac{\sum_{j\in V'} y_j w_{ij}}{\sum_{j\in V'} w_{ij}}$, i.e., y_i is placed at the weighted average position of y_j, $j \in V'$, to which y_i is connected,
- for a general (even) p the location of y_i has to minimize $\sum_{j\in V'} w_{ij}(y_i - y_j)^p$.

This is achieved numerically by several steps of Newton-Rhapson method starting at the $p = 2$ solution.

Then $V' \leftarrow V' \cup \{i\}$ and the process continues until $V' = V$. Finally each position y_i is changed to

$$x_i = \frac{v_i}{2} + \sum_{y_k < y_i} v_k \quad , \tag{4.6}$$

thus retaining order while taking volume (length) into account.

Relaxation

The arrangement obtained after the initialization is a first feasible solution for the minimum p-sum problem which is then improved by employing several sweeps of *relaxation*, first *compatible* then *Gauss-Seidel-like*. These two types of relaxation are very similar to the above initialization: The compatible relaxation, motivated in [13], improves the positions of the F-nodes one by one according to the minimization criteria above (where $V' = V$) while keeping the positions of the seeds (C-nodes) unchanged. The Gauss-Seidel-like relaxation is similarly performed, but for *all* nodes (including C). Each such sweep is again followed by (4.6).

Window Minimization

The cost of the arrangement can be further reduced by *strict minimization*, i.e., a sequence of rearrangement that accepts only changes which decrease the arrangement cost. Since done in the multilevel framework, it can be restricted at each level to just *local* changes, i.e., reordering small sets of neighboring nodes, which are adjacent (or relatively close) to each other at the current arrangement. It is easy to see that switching positions between several adjacent nodes is inexpensive, since the resulting new arrangement cost can be calculated only at the vicinity of the adjustment and not elsewhere. Such a node by node minimization was applied in our algorithm for the Minimum Linear Arrangement problem (1-sum problem, see [92]). This method may also be used for any functional. However, for the minimum 2-sum problem we have introduced a more advanced method of local minimization, called *Window Minimization* (WM), which is suitable not only for the multilevel framework but can also be used as local postprocessing relaxation in other frameworks (like the spectral approach). The difference between WM and simple node by node minimization is that WM *simultaneously* minimizes the arrangement cost of a small number of nodes (e.g., 5 to 20).

We first describe the basic WM involving the quadratic form for $p = 2$ [93], then possible generalizations are presented. Given a current approximation $\tilde{x}$ to the arrangement of the graph, denote by δ_i a *correction* to $\tilde{x}_i$. Let $\mathfrak{W} = \{i_1 = \pi^{-1}(s+1), ..., i_q = \pi^{-1}(s+q)\}$ be a *window*, i.e., q successive vertices in the current arrangement, positioned at $\tilde{x}_{i_1}, ..., \tilde{x}_{i_q}$. The local minimization problem of the $p = 2$

functional associated with a given window $\mathfrak{W}$ can be formulated as follows :

$$\text{minimize } \sigma_2(\mathfrak{W},\tilde{x},\delta) = \sum_{i,j\in\mathfrak{W}} w_{ij}(\tilde{x}_i+\delta_i-\tilde{x}_j-\delta_j)^2 + \sum_{\substack{i\in\mathfrak{W}\\ j\notin\mathfrak{W}}} w_{ij}(\tilde{x}_i+\delta_i-\tilde{x}_j)^2. \quad (4.7)$$

To prevent the possible convergence of many coordinates to one point, and, more precisely, to express the aim of having $\{x_i+\delta_i\}_{i\in\mathfrak{W}}$ an approximate permutation of $\{x_i\}_{i\in\mathfrak{W}}$ one should add constraints of the form

$$\sum_{i\in\mathfrak{W}}(\tilde{x}_i+\delta_i)^m v_i \;=\; \sum_{i\in\mathfrak{W}}\tilde{x}_i{}^m v_i\ , \quad m=1,2 \qquad (4.8)$$

where for $m=2$ we have neglected the quadratic term in δ_i. Note that the sums $\sum_{i\in\mathfrak{W}}\tilde{x}_i{}^m v_i$ for $m=1,2$ are invariant under permutations. Using Lagrange multipliers, the final formulation of the WM for $p=2$ is :

$$\text{minimize } \sigma_2(\mathfrak{W},\tilde{x},\delta,\lambda_1,\lambda_2) = \sigma_2(\mathfrak{W},\tilde{x},\delta) + \lambda_1\sum_{i\in\mathfrak{W}}\delta_i v_i + \lambda_2\sum_{i\in\mathfrak{W}}\delta_i v_i\tilde{x}_i \quad , \qquad (4.9)$$

under the second and third constraints of (4.10) below, yielding the following system of equations:

$$\begin{cases} \sum_{j\in\mathfrak{W}} w_{ij}(\delta_i-\delta_j) + \delta_i\sum_{j\notin\mathfrak{W}} w_{ij} + \lambda_1 v_i + \lambda_2 v_i\tilde{x}_i = \sum_j w_{ij}(\tilde{x}_j-\tilde{x}_i) & \text{for } i=1,\dots,q \\ \sum_i \delta_i v_i = 0 \\ \sum_i \delta_i v_i\tilde{x}_i = 0 \quad . \end{cases} \qquad (4.10)$$

Usually in a correct multilevel framework, the changes δ_i are supposed to be relatively small since the global approximation for the arrangement is inherited from the coarser levels. Their smallness is effected by the very restriction of the minimization to one window at a time. After solving the system (4.10), every vertex $i\in\mathfrak{W}$ is thus positioned at $y_i=\tilde{x}_i+\delta_i$. Feasibility with respect to the volumes of the nodes is retained by applying (4.6). Since the size and location of $\mathfrak{W}$ are quiet arbitrary, the energy cost of the new sub-arrangement can be further improved by Gauss-Seidel-like relaxation sweeps applied to an *enlarged* window $\mathfrak{W}$, where, say 5% of the window's size at each end (if possible) are added to $\mathfrak{W}$. As usual, each sweep is followed by (4.6). The final obtained energy cost is then compared with the one prior to all the window changes, the minimum of the two is accepted, updating $\tilde{x}$.

A sweep of WM with a given window size q consists of a sequence of overlapping windows, starting from the first node in the current arrangement and stepping by $\lfloor\frac{q}{2}\rfloor$ for each additional window. One such sweep is employed for every given q, while a small number of different q's is used (for actual values see Sections 4.4.2, 4.4.3 and the Appendix). Our experiments show that the algorithm is robust to changes in the chosen q's. Note that due to the multiscale framework, only bounded values of q need be used, which guarantees linear execution time of the entire algorithm. The WM is summarized in Algorithm 1.

```
Algorithm 1: WindowMinimization(graph $G$, current order $\tilde{x}$, window length $q$)
  Parameter: $k_1$ (see the Appendix)
  For $i = 1$ To $|V| - q + 1$ Step $i = i + \lfloor \frac{q}{2} \rfloor$
    $\mathfrak{W} = \{\pi^{-1}(i), ..., \pi^{-1}(i + q - 1)\}$
    Solve the system of equations (4.10)
    Apply $k_1$ sweeps of Gauss-Seidel-like relaxation on the enlarged $\mathfrak{W}$ with $\tilde{x} + \delta$
    $\tilde{x} \leftarrow \tilde{x} + \delta$ if the cost of the arrangement was decreased
  Return $\tilde{x}$
```

The use of WM for non-quadratic functional is achieved by quadratization. For $p > 2$, define $\widehat{w}_{ij} = w_{ij}(\tilde{x_i} - \tilde{x_j})^{p-2}$ and the WM follows by substituting w_{ij} with $\widehat{w}_{ij}$ in (4.7) and (4.10). For the bandwidth problem, where p should tend to infinity, additional WM sweeps with further increasing of p are employed at the end of the V-cycle as a postprocessing procedure. More details are provided in Section 4.4.2.

A more involved example is the workbound reduction problem. Using (4.4), the respective functional for $\mathfrak{W}$ can be approximated by

$$wb(\mathfrak{W}, \tilde{x}, \delta) \approx \sum_{i \in \mathfrak{W}} \Big(\sum_{\substack{j \in \mathfrak{W} \\ \tilde{x}_j < \tilde{x}_i}} w_{ij}(\tilde{x_i} + \delta_i - \tilde{x_j} - \delta_j)^p + \sum_{\substack{j \notin \mathfrak{W} \\ \tilde{x}_j < \tilde{x}_i}} w_{ij}(\tilde{x_i} + \delta_i - \tilde{x_j})^p \Big)^{2/p} = wb_p(\mathfrak{W}, \tilde{x}, \delta), \tag{4.11}$$

where p should tend to infinity so that the longest edges become dominant as desired. The quadratization of (4.11) is achieved by Taylor expansion up to the third term as follows

$$wb_p(\mathfrak{W}, \tilde{x}, \delta) \approx wb_p(\mathfrak{W}, \tilde{x}, \underline{0}) + \sum_{i \in \mathfrak{W}} \frac{\partial wb_p}{\partial \delta_i}(\mathfrak{W}, \tilde{x}, \underline{0})\delta_i + \sum_{i,j \in \mathfrak{W}} \frac{\partial^2 wb_p}{\partial \delta_i \partial \delta_j}(\mathfrak{W}, \tilde{x}, \underline{0})\delta_i \delta_j. \tag{4.12}$$

Thus, the system of equations to be solved is composed of q equations of the form $\frac{\partial wb_p}{\partial \delta_i} = 0$ and constraints (4.8). In our experiments, this minimization was applied only as a postprocessing procedure right after completing the V-cycle for $\sigma_2(G)$. Each i-th iteration of WM was done with sequentially growing even power parameter p. Since the involved analytic derivatives of (4.12) are rather lengthy, it is easier and more efficient to use *numerical* derivatives.

Simulated Annealing

A general method to escape false local minima and advance to lower costs is to replace the strict minimization by a process that still accepts each candidate change which lowers the cost, but also assigns a positive probability for accepting a candidate step which increases the cost of the arrangement. The probability assigned to a candidate step is equal to $exp(-\Delta/T)$, where $\Delta > 0$ measures the *increase* in the arrangement cost and $T > 0$ is a temperature-like control parameter which is gradually decreased toward zero. This process, known as *Simulated Annealing* (SA) [67], in large problems would usually need to apply *very gradual* cooling (decrease of temperatures), making it extremely slow and inefficient for approaching the global optimum.

In the multilevel framework, however, the role of SA is somewhat different. At each level it is assumed that the *global* arrangement of aggregates has been inherited from the coarser levels, and thus only *local*, small-scale changes are needed. For that purpose, we have started at relatively high T, lowered it *substantially* at each subsequent sweep, until window minimization is employed.

Repeated heating and cooling is successively employed for better search over the local landscape. This search is further enhanced by the introduction of a "memory"-like tool consisting of an additional permutation which stores the *Best-So-Far* (BSF) observed arrangement, which is being occasionally updated by a procedure called *Lowest Common Configuration* (LCC) [18]. LCC enables the systematic accumulation of *sub*-permutations over a sequence of different arrangements, such that each BSF sub-permutation exhibits the best (minimal) sub-order encountered so far. The complete description of the SA and LCC algorithms is given in [92].

The entire disaggregation procedure for the minimum p-sum problem is summarized below in Algorithm 2.

```
**Algorithm 2:** Disaggregation(coarse level $\mathcal{C}$, fine level $\mathcal{F}$)
  **Parameters:** $k_2, k_3$ (see the Appendix)
    **Decide** on the appropriate power $p$
    **Initialize** $\mathcal{F}$ from $\mathcal{C}$
    **Apply** $k_2$ sweeps of compatible relaxation on $\mathcal{F}$
    **Apply** $k_3$ sweeps of Gauss-Seidel-like relaxation on $\mathcal{F}$
    **Apply** Window Minimization on $\mathcal{F}$
    **Apply** SA on $\mathcal{F}$
    **If** $\mathcal{F}$ is the finest level **add** postprocessing of minimization
  **Return** the linear order of $\mathcal{F}$
```

4.4 Results and Related Work

We have implemented and tested the algorithm using standard C++, LAPACK++ [84] and LEDA libraries [74] on Linux machine. The implementation is non-parallel and not fully optimized.

4.4.1 Previous work

The Minimum linear arrangement [92]. We have tested our algorithm on the benchmarks provided by Petit [79] and Koren [68]. Most successful competitive heuristics were : Spectral Sequencing, Optimally Oriented Decomposition Tree, Multilevel based, Simulated Annealing, Genetic Hillclimbing and some of their combinations. The test suite provided in [79] contains rather small graphs for which our algorithm gave the best costs (in almost all cases) in comparison to all previously listed heuristics. The running time was so negligible, that comparison was meaningless. The most interesting result was the comparison of our AMG-like algorithm with the combination of spectral and multilevel approaches [68] on very large graphs (introduced there). The fast version of our algorithm which run only a fifth of the time of [68] exhibited an average improvement of 7%. Our slower but more evolved version improved the costs of [68] by 12%. Other heuristics were not tested on this suite, because of their higher than linear complexity. For complete list of results see [92].
The Minimum 2-sum [93]. We have found only one article [50] with an implemented algorithm and numerical results for the minimum 2-sum problem. The algorithm is based on the spectral approach. Since their test suite is relatively small to provide enough information regarding the problem, we have launched a new, much larger test suite and compared our results to the spectral approach. Our multilevel algorithm without any minimization at the finest level provided much better results (better by an average of 31.4%) than the spectral one. Finally, the minimization process applied after both strategies has proven itself to be good enough for both of the approaches and almost equalized the results. For complete list of results see [93].

4.4.2 Bandwidth

There are many different theoretical and practical results for the bandwidth problem, e.g., [24, 80, 25, 42], to mention just a few. However, only a small number allow tests on large inputs within a reasonable execution time, e.g., [7, 34, 35]. Since we believe that a fair comparison of two heuristics should include final results as well as running times, and since our algorithm is able to deal with very large instances, we have thus chosen to test it on the test suites of [7, 34, 35] which include large enough graphs to make the picture complete. These graphs are presented at the leftmost three columns of Table 4.1.

We compare our results to the best results achieved in [7, 34, 35], presented at column "bk_∞" of Table 4.1. These results are the best obtained by testing many (e.g., 19 in [35]) *different* algorithms, most of which are versions of the spectral approach. That is, ordering the graph vertices according to the sorted coordinates of the second eigenvector of the graph's *Laplacian* A (a $|V| \times |V|$ matrix), whose terms are defined by

$$A_{ij} = \begin{cases} -w_{ij} & \text{for } ij \in E,\ i \neq j \\ 0 & \text{for } ij \notin E,\ i \neq j \\ \sum_{k \neq i} w_{ik} & \text{for } i = j \quad . \end{cases} \tag{4.13}$$

Our results (columns "M_5", "M_{10}" and "M_{200}") are given as ratios to theirs, i.e., to column "bk_∞". "M_5" introduces the results obtained by one V-cycle with five WM at all levels (with $q = 5, 10, 15, 20, 25$, see Algorithm 1). Note that on the finest level p is increased by two from one window size to another. We run the algorithm one hundred times, each starts from a different permutation of the nodes. The best obtained results show an improvement of about 23% over "bk_∞". The means of the one hundred runs are worse than the corresponding "M_5"-values by an average of 7%, while the standard deviation (around the means) is 4.7% on the average. We have next tested the outcome of our algorithm with enlarged number of WM. The V-cycle corresponding to "M_{10}" uses ten WM at all levels (with window sizes 5 to 50 and increased p only at the finest level) and results with an improvement of 26%, while "M_{200}" has the same ten WM at each coarse level and 200 iterations at the finest, where p is increased by two every four iterations of window sizes 5, 10, 20 and 40. (In fact, even though p in (4.2) should tend to infinity, in practice, the minimization process has almost not progressed after $p \approx 100$.) The "M_{200}" shows improvement of 34% on the average over "bk_∞". In these two versions, the *means* of the hundred runs are worse than the corresponding "M_{10}"("M_{200}")-values by an average of 6(4)%, while the standard deviation (around the means) is 3.6(2.3)% on the average.

We have finally tested our algorithm on the five random graphs appearing in the benchmark [79]. We compare a *single* run of our V-cycles with the results of the exact spectral method and with those of the Cuthill-McKee permutation [36] which was checked also in [35]. The results are summarized in Table 4.2 showing a clear advantage to our multilevel approach even for those obviously unstructured random graphs.

4.4.3 Workbound reduction

Continuing the comparison of multilevel and spectral frameworks started in [93], we present our results for the workbound reduction problem compared to the best known values from [34, 35]. The test suite graphs are the same as in the bandwidth problem. In the second part of Table 1 we present the results we have obtained for these graphs. In column "bk_{wb}" we have extracted the *best* results reported in [34, 35]. These results were obtained by several modifications of the spectral sequencing method. Then the results for two types of V-cycles (ten executions for each V-cycle) are presented: the "$\sigma_2(G)$" V-cycle which is aimed at achieving fast performance and thus somewhat compromising the quality of the arrangement cost by simply approximating the workbound only with the $\sigma_2(G)$ solution; and the "$\sigma_2(G)$+WM"

V-cycle which starts with the $\sigma_2(G)$ solution and then applies a postprocessing of 20 additional iterations with increased p of WM (of sizes 5,10,15,20,25,5,10,...) using (4.4) and then ten sweeps of node by node minimization using (4.3). The latter version runs longer but succeeds in finding lower cost arrangements. Our results are presented in the form of ratio between our cost and the best known values from [34, 35]. On the average they exhibit 18% improvement for $\sigma_2(G)$ and 31% when the postprocessing is added. The *means* of the ten runs are worse than the corresponding "$\sigma_2(G)$"("$\sigma_2(G)$+WM")-values by an average of 2.5(1.5)%, while the standard deviation (around the means) is 1(0.5)% on the average.

Finally, we have also tried to add stochasticity by implementing the SA process. Here as well as for the bandwidth problem we obtained no significant improvement, i.e., no more than the observed variance. Still, as was shown in [92], SA can be extremely important in other problems.

4.4.4 Additional experiments

We have tried to use the minimum 2-sum as a first approximation also for the bandwidth as it was done for the workbound. However, this attempt was unsuccessful. The nature of the bandwidth functional is somewhat different than other p-sum problems or the workbound. It deals with the minimization of only several concrete edges, those which are the longest, while in the p-sum and workbound it is necessary to minimize many edges, at least one per node.

As an additional preliminary experiment aimed at checking whether the minimum 2-sum may indeed provide a good first approximation for another functional, we tested it for the *wavefront reduction* problem defined by

$$wf(G,\pi) = \left(\frac{\sum_i |f_i|^2}{n}\right)^{1/2} \quad , \tag{4.14}$$

where $f_i = adj(\{\pi^{-1}(1),...,\pi^{-1}(i)\}) \bigcup \{\pi^{-1}(i)\}$ and $adj(X) = \bigcup_{j\in X}\{k : kj \in E\}\backslash X$. We have compared our results with those of [59] obtained by a multilevel algorithm. We have just *evaluated* for 15 graphs the wavefront functional on the arrangement produced by the V-cycle with $p = 2$ and obtained similar results to those presented in [59]. We emphasize that these results are *prior* to any postprocessing which would involve minimization with the particular wavefront functional.

4.5 Conclusions

We have presented a variety of multilevel algorithms for the class of linear ordering problems for general graphs. These algorithms are based on the general principle that during coarsening each vertex may be associated to more than just one aggregate according to some "likelihood" measure. The uncoarsening initialization, which produces the first arrangement of the fine graph nodes, strongly relies on

Table 4.1: Results.

Graph	$\vert V\vert$	$\vert E\vert$	bk_∞	M_5	T_{M_5}	M_{10}	M_{200}	bk_{wb}	σ_2	σ_2+WM
3dtube	4.5E+04	1.6E+06	2334	0.89	11.00	0.87	0.81	1.48E+11	1.04	0.99
add20	2.4E+03	5.4E+03	711	0.60	0.03	0.55	0.50	9.78E+07	0.39	0.20
add32	5.0E+03	7.4E+03	669	0.03	0.05	0.03	0.03	1.67E+07	0.02	0.01
barth	6.7E+03	2.0E+04	200	0.76	0.12	0.72	0.64	4.09E+07	0.99	0.88
barth4	6.0E+03	1.7E+04	213	0.60	0.10	0.58	0.55	3.23E+07	0.76	0.71
barth5	1.6E+04	4.6E+04	370	0.65	0.30	0.63	0.57	1.89E+08	0.93	0.88
bcspwr08	1.6E+03	2.2E+03	131	0.63	0.03	0.63	0.53	1.10E+06	0.76	0.64
bcspwr09	1.7E+03	2.4E+03	123	0.68	0.07	0.65	0.57	1.18E+06	0.76	0.63
bcspwr10	5.3E+03	8.3E+03	288	0.68	0.18	0.63	0.52	1.43E+07	0.85	0.71
bcsstk12	1.4E+03	1.6E+04	109	0.61	0.10	0.61	0.57	4.29E+06	0.87	0.83
bcsstk13	2.0E+03	4.1E+04	546	0.69	0.20	0.64	0.60	1.63E+08	0.80	0.58
bcsstk24	3.6E+03	7.8E+04	227	0.79	0.29	0.80	0.79	7.10E+07	1.01	1.00
bcsstk29	1.4E+04	3.0E+05	838	0.68	1.48	0.67	0.63	1.09E+09	0.85	0.78
bcsstk30	2.9E+04	1.0E+06	2512	0.50	3.15	0.48	0.43	4.32E+09	0.91	0.67
bcsstk31	3.6E+04	5.7E+05	1104	1.14	3.50	1.03	0.78	1.97E+10	0.60	0.51
bcsstk32	4.5E+04	9.9E+05	2339	0.97	4.50	0.87	0.71	2.83E+10	0.61	0.47
bcsstk33	8.7E+03	2.9E+05	519	1.12	1.55	1.03	0.99	1.93E+09	0.98	0.87
bcsstk35	3.0E+04	7.1E+05	1764	0.69	3.16	0.66	0.55	1.00E+10	0.74	0.62
bcsstk36	2.3E+04	5.6E+05	1474	0.70	2.71	0.67	0.57	8.52E+09	0.74	0.66
bcsstk37	2.6E+04	5.6E+05	1373	0.75	3.06	0.70	0.59	1.45E+10	0.49	0.44
bcsstk38	8.0E+03	1.7E+05	669	0.64	0.60	0.58	0.55	4.52E+08	0.84	0.69
bcsstm13	6.5E+02	9.9E+03	171	0.62	0.06	0.62	0.60	6.50E+06	0.89	0.78
blckhole	2.1E+03	6.4E+03	105	1.15	0.13	1.11	0.96	8.91E+06	0.98	0.85
bus1138	1.1E+03	1.5E+03	106	0.61	0.06	0.59	0.51	5.52E+05	0.85	0.69
bus685	6.9E+02	1.3E+03	83	0.47	0.05	0.46	0.42	2.28E+05	0.82	0.70
can1054	1.1E+03	5.6E+03	121	0.74	0.06	0.73	0.67	2.59E+06	1.00	0.67
can1072	1.1E+03	5.7E+03	159	0.81	0.06	0.78	0.74	4.08E+06	0.90	0.55
can445	4.5E+02	1.7E+03	78	0.76	0.02	0.74	0.71	9.12E+05	0.93	0.80
can838	8.4E+02	4.6E+03	126	0.77	0.03	0.75	0.71	2.80E+06	0.98	0.66
ct20stif	5.2E+04	1.3E+06	3187	1.30	6.40	1.26	0.80	1.94E+11	0.38	0.29
dwt1007	1.0E+03	3.8E+03	38	0.76	0.07	0.76	0.74	4.63E+05	0.98	0.94
dwt2680	2.7E+03	1.1E+04	65	0.97	0.16	0.95	0.86	3.74E+06	1.00	0.94
dwt918	9.2E+02	3.2E+03	50	0.72	0.06	0.70	0.68	4.55E+05	0.92	0.85
ex27	9.7E+02	2.0E+04	128	0.96	0.05	0.96	0.95	5.81E+06	1.01	0.77
finan512	7.5E+04	2.6E+05	1331	0.91	2.65	0.87	0.84	6.19E+09	0.87	0.64
gearbox	1.5E+05	4.5E+06	6271	0.68	26.00	0.86	0.65	1.36E+12	0.57	0.42
gupta3	1.7E+04	4.7E+06	12535	0.70	68.00	0.70	0.66	3.26E+11	1.11	0.99
jagmesh1	9.4E+02	2.7E+03	27	1.19	0.04	1.19	1.11	5.38E+05	1.04	1.00
jagmesh9	1.3E+03	3.9E+03	40	0.98	0.08	0.98	0.98	9.82E+05	0.90	0.87
memplus	1.8E+04	4.2E+04	5747	0.85	0.16	0.81	0.59	7.48E+10	0.57	0.15
msc10848	1.1E+04	6.1E+05	1349	0.78	1.50	0.73	0.64	3.08E+09	0.96	0.62
msc23052	2.3E+04	5.6E+05	1524	0.70	2.14	0.64	0.56	8.00E+09	0.78	0.69
nasa1824	1.8E+03	1.9E+04	205	0.80	0.14	0.77	0.73	2.68E+07	1.03	0.93
nasa4704	4.7E+03	5.0E+04	348	0.67	0.39	0.64	0.60	1.36E+08	0.96	0.91
pwt	3.7E+04	1.4E+05	339	0.92	1.20	0.88	0.76	7.51E+08	0.93	0.89
pwtk	2.2E+05	5.7E+06	2190	0.89	31.00	0.86	0.77	2.27E+11	0.67	0.66
shuttleeddy	1.0E+04	4.7E+04	177	0.72	0.56	0.70	0.67	6.46E+07	0.83	0.74
skirt1	1.3E+04	9.2E+04	309	0.60	0.50	0.57	0.50	1.73E+08	0.31	0.26
sstmodel	2.7E+03	9.7E+03	83	0.92	0.13	0.90	0.81	4.72E+06	0.82	0.74
twotone	1.2E+05	9.4E+05	19538	0.77	16.00	0.74	0.67	4.43E+12	0.76	0.65
vibrobox	1.2E+04	1.7E+05	3961	0.60	1.80	0.56	0.46	2.70E+10	0.90	0.58
AVERAGE				**0.77**		**0.74**	**0.66**		**0.82**	**0.69**

Table 4.2: Results for random graphs.

Graph	$\lvert V\rvert$	$\lvert E\rvert$	**Spectral**	**Cuthill-McKee**	M_5	M_{10}	M_{200}
randomA1	1.0E+03	5.0E+03	828	0.80	0.65	0.59	0.55
randomA2	1.0E+03	2.5E+04	969	0.92	0.91	0.88	0.84
randomA3	1.0E+03	5.0E+04	985	0.95	0.95	0.94	0.90
randomA4	1.0E+03	8.2E+03	855	0.89	0.83	0.75	0.69
randomG4	1.0E+03	8.2E+03	143	0.71	0.54	0.51	0.50

energy considerations (unlike usual interpolation in classical AMG). This initial order is further improved by Gauss-Seidel-like relaxation, window minimization and possibly by employing stochasticity, i.e., simulated annealing. The running time of the algorithms is linear, thus it can be applied to very large graphs. In addition, we have proposed two general principles that can be used for different functionals : (1) a first approximation can be obtained from the arrangement produced by one V-cycle of the minimum 2-sum problem instead of using the very popular spectral approach; (2) the continuation approach in which functionals that contain an evaluation of power p are successively approximated by a sequence of similar but with lower power functionals.

Since our algorithms were developed for practical purposes we compared them to many different heuristics : Spectral Sequencing, Optimally Oriented Decomposition Tree, Multilevel based, Simulated Annealing, Genetic Hillclimbing and other. In almost all cases we observed significant improvement of the results by tens and sometimes by hundreds percents. Our algorithms have proven themselves to be very stable (i.e., small standard deviations) and of high quality both as a first approximation (using "light" V-cycles) and as more aggressive energy minimizers (with more "heavy" postprocessing).

We recommend our multilevel algorithms as a general practical method for solving linear ordering problems and as a fast and high-quality method for obtaining first approximation for them. The implemented algorithm can be obtained at [90].

Appendix: Parameters

In order to control the running time of the algorithm it is important to decrease the total number of edges of the constructed coarse graphs. This is achieved by the following two parameters: the maximum allowed coarse neighborhood size r, which restricts the allowed size $|N_i|$ of the coarse neighborhood of a vertex $i \in F$ by deleting the weakest w_{ij}, $j \in C$; and the edge filtering threshold ϵ, which deletes every *relatively* weak edge ij (from the created coarse graph) for which both $w_{ij} < \epsilon \cdot s_i$ and $w_{ij} < \epsilon \cdot s_j$, where $s_i = \sum_k w_{ik}$.

The specific values of r and ϵ along with those of the three parameters controlling Algorithms 1 and 2 are presented in Table 4.3. Note that these parameters are the ones used only for the finest levels. As the coarse graphs become much smaller they are accordingly increased. This hardly affects the entire running time of the

Table 4.3: The parameters used in the V-cycle.

Parameter	"Value"	The increase for level L
The coarse neighborhood size (r)	10	$+log(R)$
The edge filtering threshold (ϵ)	0.001	$\cdot 0.9^{log(R)}$
k_1 used in the WM	5	$+log(\sqrt{R})$
The number of sweeps of Compatible relaxation (k_2)	10	$+2 \cdot L$
The number of sweeps of Gauss-Seidel relaxation (k_3)	10	$+2 \cdot L$

algorithm but systematically improves the obtained results. In the last column of Table 4.3 we specifically describe the increase introduced for each parameter as a function of level L, which usually depends on the ratio $R = max(1, |E_0|/|E_L|)$ measuring the relative decrease of the number of edges at level L compared with the original graph.

We tested many options for the window sizes in Algorithm 1. Usually these sizes were relatively small and very robust to changes. In our implementation we used $WinSizes = \{5, 10, 15, 20, 25, 30\}$, however similar results were obtained with other sets of windows, for example, $WinSizes = \{5, 9, 17, 23, 29\}$.

Acknowledgments

This research was supported by Carl F. Gauss Minerva Center for Scientific Computation at the Weizmann Institute of Science; The Israel Science Foundation Grant no. 295/01; The German-Israel Foundation for scientific research and development (GIF) Grant no. I-718-135.6/2001; European Commission Project IST-2002-506766 Aim Shape.

CHAPTER 5

Two-dimensional layout problems

The problems we will address in this section are as follows. We want to find an optimal layout of a set of two-dimensional objects such that (a) the total length of the connections between these objects will be minimal (b) the two-dimensional space will be well utilized and (c) the overlapping between objects will be as little as possible. This class of problems can be modeled by a graph in which every vertex has a predefined area and each edge has a predefined weight. While the first and the third conditions usually have a single meaning, the second requirement can be concreted in different ways. We will precisely define the space utilization demand in Section 5.4. Intuitively explaining, the second requirement will imply the space use whose lack can cause various annoying inconsistencies between the sizes of space and objects (see Figure 5).

While the minimization problem can be solved relatively easy, the space utilization and non-overlapping conditions in this class of problems significantly increase the complexity of the algorithms dealing with it. There are two usual strategies to satisfy the non-overlapping condition that lead to economical space utilization: force directed methods and penalty functions. The force directed methods [47, 43] are based on a simulation of a related physical model known as the *spring embedder*. Spring embedder algorithms simulate a physical model where objects and connections represent various forces and the result is a drawing representing a configuration of some, possibly local, minimum energy. The penalty function strategy [73, 9] dictates to reinforce the minimization function with a penalty function which pushes away too close neighbor objects by adding a large enough penalty term to the energy.

These methods can have very different natures and formulations. Some of them have proven themselves in solution quality while other in execution time. However, there are two main disadvantages regarding both of them: (a) if the number of

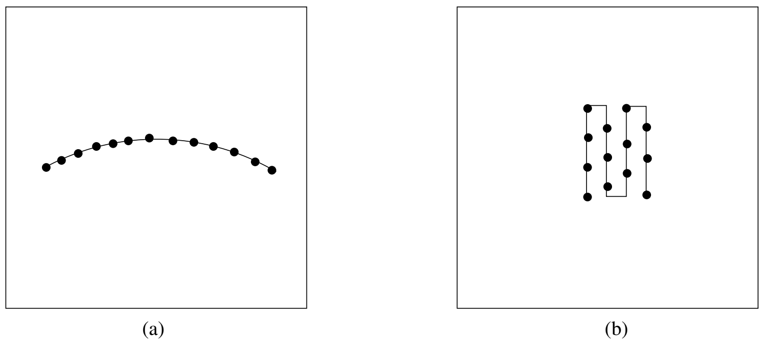


Figure 5.1: Usually, the graph "snake" is presented by most of the graph drawing algorithms as a line or chord (like the left hand part of the figure). In this case, when the number of nodes is quite big, the viewable number of nodes must be very small and the space will be utilized very inefficiently. Moreover, if the sizes of space and vertices are fixed such a layout may be practically impossible. One of the possible efficient space utilization for the graph "snake" is presented at the right hand part of the figure.

penalty components (or spring forces definitions) is too big (for example all pairs of nodes) then the complexity necessarily becomes quadratic in the number of objects; (b) otherwise, when too many components are not taken into account there can be many unforeseen violations. Thus, in terms of graph model $G = (V, E)$, in case (a) both methods are of the complexity proportional to $O(|V||E|)$ or $O(|V|^2)$, which makes the applications on large graphs very difficult.

In this section we propose a linear time strategy for the two-dimensional layout problem which compactly utilizes a given space. The almost non-overlapping condition will be achieved as a "side effect" of the distribution of the objects over the space instead of mentioning the non-overlapping rule (or force) explicitly for the objects. The strategy is scalable and thus may be applied to large instances. We will put a grid on the area over which the minimization has to be performed and will demand from every square to contain not more than some amount of material, e.g., no more than its area. The number of variables (which influences the running time of the algorithm) is dependent only on the grid size.

There are many areas in which such algorithms could find themselves very useful. We will mention here two of them: graph drawing and VLSI design.

5.1 Graph drawing

Graph drawing addresses the problem of constructing geometric representation of graphs, and has important applications to many computer technologies. There are many different demands for graph drawing problems like "draw a graph with" (a) minimum number of edge crossings; (b) minimum total edge length; (c) predefined angular resolution, etc. (for a complete survey, see [9]). For some of the applications two questions play extremely important role when the vertices have different geometric representations (like non-uniform sized circles, rectangles or other forms). The first question is concerning the non-overlapping constraint over all pairs of the vertices, while the second refers to the correct utilization of the space (paper, drawing window, etc.), since, it is important to be able to indeed draw the graphs within its given frames. We should stress that the ability to achieve a compact picture is of great importance, since area-efficient drawings are essential in practical visualization applications where screen space is one of the most valuable commodities. Today, the fastest algorithms for drawing graphs do not take into account the space utilization condition. The most popular strategy which does address these questions is the previously mentioned force directed method [43]. However, its running time is still quadratic.

Another extension that includes graph drawing aspects is a representation of higraphs. Higraphs, a combination and extension of graphs and Euler/Venn diagrams, were defined by Harel in [53]. Higraphs extend the basic structure of graph and hypergraphs to allow vertices to describe inclusion relationships. Adjacency of such vertices is used to denote set-theoretic Cartesian product. 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 [52]. For small-sized higraphs it has indeed yielded nice results, but due to its high complexity, it poses efficiency problems when used for larger higraphs.

5.2 The placement problem

The electronics industry has achieved a phenomenal growth over the last two decades, mainly due to the rapid advances in integration technologies, 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 VLSI design starts from the algorithm that describes the behavior of the target chip. First, the corresponding architecture of the system is defined. It is mapped onto the chip surface by *floorplanning*. The next design evolution in the behavioral domain defines finite state machines which are structurally implemented with functional modules such as registers and arithmetic logic units. These modules are then geometrically placed onto the chip surface using special tools for *automatic module placement* followed by *routing*, with a goal of minimizing the interconnects area and signal delays [31]. For a current most leading approaches in this area see [88, 102, 28, 61, 63, 76, 72, 85, 69, 71, 21, 44, 101, 60, 58, 23] and for a most recent survey on the placement techniques see [75].

The automatic placement stage consists of two main stages : *the global placement* and *the detailed placement.* The global placement assigns the modules to the chip region so as to minimize a cost function based on the wirelength and assures that the modules are distributed evenly among the entire chip region. To ensure that the placement solution can be legalized to the detailed (i.e., overlap-free) placement without significant cell movements, the global placement result should satisfy certain area density constraints.

The global placement is one of the most challenging problems during VLSI layout synthesis. 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.

Let us formulate one possible simplified (which is still very challenging) objective function for global placement. The circuit is described by the set of modules $\mathcal{M} = \{\mu_1, ..., \mu_N\}$ and nets $\mathcal{N} = \{\nu_1, ..., \nu_N\}$. Every module μ_i occupies a non-zero area and the coordinates of the module at the chip is interpreted as the coordinates of its center of mass. Each net ν connects a subset of the modules $\mathcal{M}_\nu$. Modules and nets are represented by nodes and hyperedges, respectively, in the following hypergraph model. Given a fixed rectangular region $\mathcal{R}$ (chip area) the modules should be positioned inside this region. When the modules are positioned in $\mathcal{R}$, each net $\nu \in \mathcal{N}$ is assigned a "bounding-box" wirelength estimation $l(\nu)$ which is equal to the half-perimeter of the smallest rectangle circumscribing its modules :

$$l(\nu) = (x_{max}(\nu) - x_{min}(\nu)) + (y_{max}(\nu) - y_{min}(\nu)) \quad , \tag{5.1}$$

where $x_{max}(\nu)$ denotes the maximum x-coordinate of any side of any module in net ν ($x_{min}(\nu)$, $y_{max}(\nu)$ and $y_{min}(\nu)$ are defined similarly). In fact, this estimate is the lower bound of the real wirelength. The corresponding wirelength estimation l for the entire circuit is obtained by direct summation over all nets :

$$l = \sum_{\nu \in \mathcal{N}} l(\nu) \quad . \tag{5.2}$$

A more precise calculation of the wirelength can only be made after routing.

In order to simplify our task we will work with the graph model of the circuit. The fact that a net may connect more than two cells implies that a graph model of the circuit is inaccurate. However, graph approximations such as the clique model we describe here are often used effectively for many purposes. In our model, the circuit will be represented as a graph $G = (V, E)$, where each node in V corresponds to a unique module in $\mathcal{M}$, i.e., $V = \mathcal{M}$, while each net $\nu \in \mathcal{N}$ generates a clique C_ν with the set of nodes $\mathcal{M}_\nu$ and the set of edges E_ν defined as

$$E_\nu = \{ij \ : \ i, j \in \mathcal{M}_\nu \text{ and } i \neq j\} \quad .$$

Thus, each edge may be generated by several cliques more than once. The weight of an edge ij generated by k cliques C_ν, $1 \le \nu \le k$, is defined as $w_{ij} = \sum_{\nu=1}^{k} 1/(|\mathcal{M}_\nu| - 1)$. The desired position of the node i is denoted by (x_i, y_i). With these definitions we formulate the quadratic objective function Φ :

$$\Phi = \sum_{ij \in E} w_{ij} \left[(x_i - x_j)^2 + (y_i - y_j)^2 \right] ,$$

which should be minimized subject to the set of appropriate constraints expressing various goals such as the non-overlapping conditions, predefined possible layout regions for certain modules, forbidden regions, etc. In our application, the set of constraints will be responsible for uniformly fill the chip area, which will result with the almost non-overlapping side effect. This quadratic function is used in many placement methods. One of the reasons for using the quadratic objective function has been that it is continuously differentiable and it can be minimized by solving a system of linear equations.

5.3 General scheme

The overall algorithm for solving a simple two-dimensional layout problem consists of two main parts : a) the *exterior* V-cycle for the repeated coarsening of the instance graph (bold circles and lines, Figure 5.2) and b) the procedure for *improving* the current layout inherited from a coarser level (dashed lines that start at each empty circle, Figure 5.2). The coarsening part of the exterior V-cycle is of similar nature as the weighted aggregation described in [87, 92, 93, 94]. The correction routine consists of an iterative process of approximating the non-linear problem, each being solved by an *interior* V-cycle. The general algorithm which has been developed by D. Ron and A. Brandt serves as a basis for my project. The interior V-cycle from (b) (which is the goal of my project) is intended for the correction of the initial approximation for the 2D-layout problem which is inherited from the coarser graph. The new algorithm presented here involves a different kind of constraints, i.e., inequality constraints rather than equality constraints as explained below. In order to achieve a uniform distribution of the vertices (explained below as equidensity constraints) and simultaneous smooth corrections for the node locations (corrections that preserve the basic structure inherited from the coarse graphs), we will discretize the given drawing area and then will define a set of variables for both vertical and horizontal corrections of the nodes locations. Then, each node will be moved according to the relative influence of its neighbor corrections and a new layout will be obtained.

5.4 Problem formulation

Given an initial approximation of the G_i-th layout at level i, which is placed within the given drawing domain (assumed to be rectangular), we will first put a grid

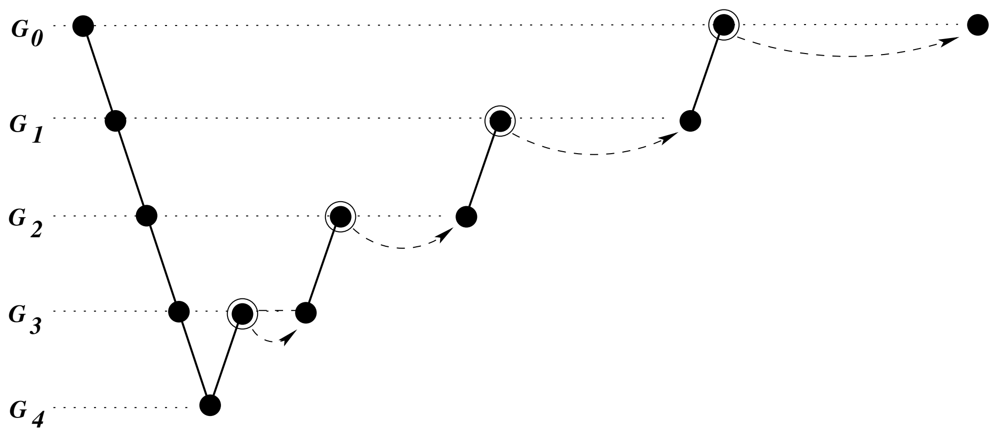


Figure 5.2: An example of the general scheme.

over that domain and define a new problem (bellow in (5.3)). The discretization is performed by a grid $\mathcal{G}$ with the set of *points* $\mathcal{P}(\mathcal{G})$, where $|\mathcal{P}(\mathcal{G})| = N_{\mathcal{G},x} \cdot N_{\mathcal{G},y}$ points and x, y correspond to the respective axes (see Figure 5.3). Denote by $\mathcal{S}(\mathcal{G})$ the set of *squares* defined by $\mathcal{G}$, clearly, $|\mathcal{S}(\mathcal{G})| = (N_{\mathcal{G},x} - 1) \cdot (N_{\mathcal{G},y} - 1)$. The points in $\mathcal{P}(\mathcal{G})$ are counted sequentially from 0 at $(0,0)$-coordinate until $|\mathcal{P}(\mathcal{G})| - 1$ at the $(N_{\mathcal{G},x}, N_{\mathcal{G},y})$-coordinate. For every point $p \in \mathcal{P}(\mathcal{G})$ we define two variables U_p and V_p which correspond to the horizontal and vertical corrections of the (up to 4) neighboring squares of $\mathcal{G}$, respectively. For example, node j depicted in Figure 5.3 will get the horizontal correction $\alpha_{12,j}U_{12} + \alpha_{13,j}U_{13} + \alpha_{17,j}U_{17} + \alpha_{18,j}U_{18}$, where $\alpha_{12,j}$, $\alpha_{13,j}$, $\alpha_{17,j}$ and $\alpha_{18,j}$ are the bilinear interpolation coefficients (explained later).

Let us formulate the minimization problem. For a node i we define the set of four closest points (the corners) in $\mathcal{P}(\mathcal{G})$ by $c(i) = \{\mathrm{rt}(i), \mathrm{rb}(i), \mathrm{lt}(i), \mathrm{lb}(i)\}$ (i.e., right-top, right-bottom, left-top, left-bottom). Similarly, $\overline{c}(s) = \{\overline{\mathrm{rt}}(s), \overline{\mathrm{rb}}(s), \overline{\mathrm{lt}}(s), \overline{\mathrm{lb}}(s)\}$ are the corner points of a square $s \in \mathcal{S}(\mathcal{G})$. The 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

$$\mathfrak{E}(U,V) =$$
$$\sum_{ij \in E} w_{ij} \left[\left(\tilde{x}_i + \sum_{p \in c(i)} \alpha_{pi} U_p - \tilde{x}_j - \sum_{p \in c(j)} \alpha_{pj} U_p \right)^2 + \left(\tilde{y}_i + \sum_{p \in c(i)} \alpha_{pi} V_p - \tilde{y}_j - \sum_{p \in c(j)} \alpha_{pj} V_p \right)^2 \right] \tag{5.3}$$

where α_{pi} are the bilinear interpolation coefficients. For example, in Figure 5.4 for vertex i in square s these coefficients are defined as

$$\alpha_{pi} = \begin{cases} \frac{d_1 e_2}{(d_1+d_2)(e_1+e_2)} & \text{if } p = \mathrm{rt}(i) \\ \frac{d_1 e_1}{(d_1+d_2)(e_1+e_2)} & \text{if } p = \mathrm{rb}(i) \\ \frac{d_2 e_2}{(d_1+d_2)(e_1+e_2)} & \text{if } p = \mathrm{lt}(i) \\ \frac{d_2 e_1}{(d_1+d_2)(e_1+e_2)} & \text{if } p = \mathrm{lb}(i) \end{cases} .$$

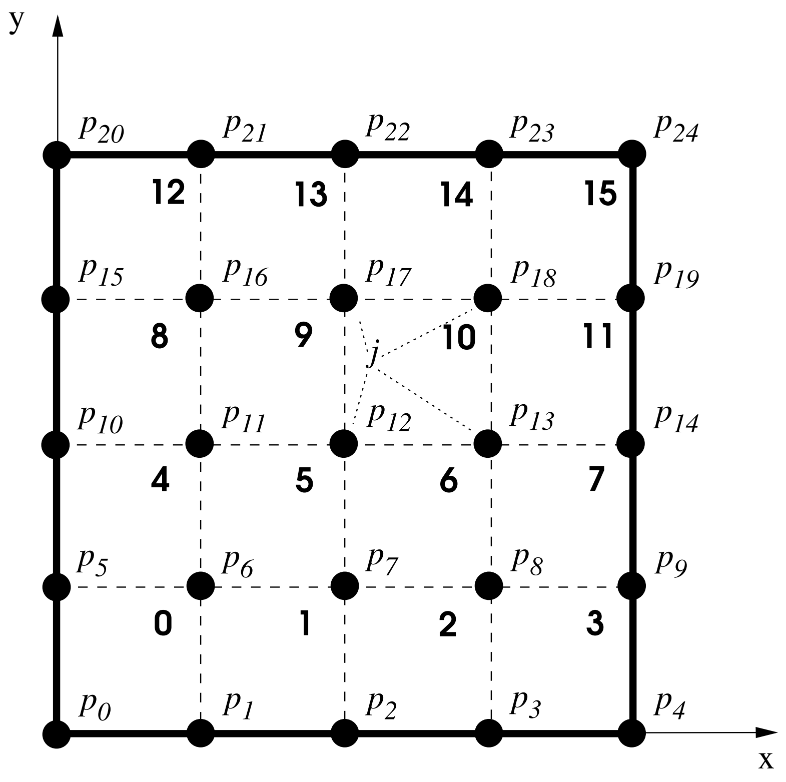


Figure 5.3: An example of a grid $\mathcal{G}$ with $N_{\mathcal{G},x} = N_{\mathcal{G},y} = 5$. The grid points and squares are labeled by p_i and bold numbers, respectively.

Function $\mathfrak{E}$ should be minimized subject to a set of constraints (otherwise, all nodes will coincide to a single point). The first type of constraints is called the *equidensity* constraints, i.e., the constraints which will ensure the uniform infill of the drawing area by the vertices. These constraints will replace the natural non-overlapping constraints which prevent the vertices from superpositioning.

The equality version of the equidensity constraints has been recently formulated and developed by D. Ron and A. Brandt. This project consists of the *inequality* version of the constraints and represents an extension of their ideas and methods. For the completeness of this chapter we will present both the equality and the inequality types of the equidensity constraints. Let $v(i)$ be the area of vertex i. Denote the area of square s by $\mathcal{A}(s) = w_s h_s$, where h_s and w_s are the height and width of square s, respectively. Denote by $\Upsilon(s)$ the total area of the vertices in grid square s, i.e., $\Upsilon(s) = \sum_{i \in V_s} \mathfrak{a}(i,s)$, where V_s is the set of nodes inside s and $\mathfrak{a}(i,s)$ is the partial area of vertex i coincide with s, see Figure 5.5. Similarly, for the right (left, top, bottom) *neighbor* square of s the total area and total area of vertices within will be denoted by $\mathcal{A}_{r(l,t,b)}(s)$ and $\Upsilon_{r(l,t,b)}(s)$, respectively. The entire

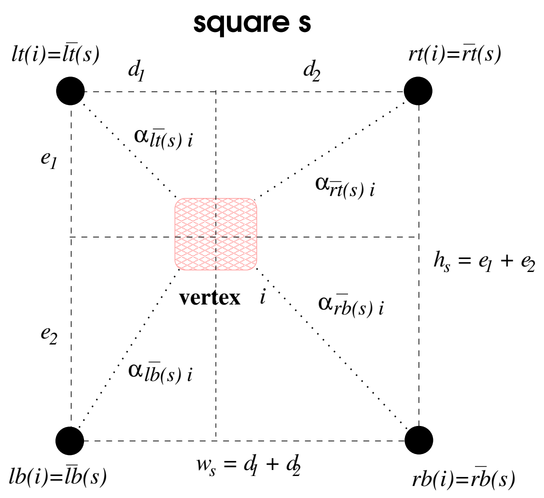


Figure 5.4: Bilinear interpolation.

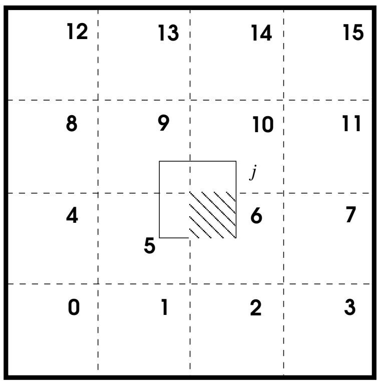


Figure 5.5: An example of a vertex that belongs to more than one square: the vertex j is shared among squares 5, 6, 9 and 10. The filled part of j is $\mathfrak{a}(j,6)$.

constraint for a square s will be

$$\begin{aligned}\mathfrak{eqd}(s) = \frac{\Upsilon(s)+\Upsilon_r(s)}{\mathcal{A}(s)+\mathcal{A}_r(s)} h_s \frac{U_{\mathrm{rt}(s)}+U_{\mathrm{rb}(s)}}{2} - \frac{\Upsilon(s)+\Upsilon_l(s)}{\mathcal{A}(s)+\mathcal{A}_l(s)} h_s \frac{U_{\mathrm{lt}(s)}+U_{\mathrm{lb}(s)}}{2} - \\ \frac{\Upsilon(s)+\Upsilon_t(s)}{\mathcal{A}(s)+\mathcal{A}_t(s)} w_s \frac{V_{\mathrm{rt}(s)}+V_{\mathrm{lt}(s)}}{2} + \frac{\Upsilon(s)+\Upsilon_b(s)}{\mathcal{A}(s)+\mathcal{A}_b(s)} w_s \frac{V_{\mathrm{rb}(s)}+V_{\mathrm{lb}(s)}}{2} + \Upsilon(s) \le \mathcal{A}(s) \ . \end{aligned} \tag{5.4}$$

First four summands at the left hand side correspond to the approximate amount of flow through the "virtual" borders between squares and its neighbor squares as shown in Figure 5.6. The main demand of the equidensity constraint is that the total area of the vertices in square s will be less than or equal to $\mathcal{A}(s)$.

The constraints of the second kind are the boundary conditions. They forbid flow across the boundary, i.e., it nullifies all boundary U_p and V_p values that correspond to flow across the boundary.

We have started to develop the algorithm with equality constraints, i.e., when the flow through the square plus its current amount of the material is equal to its area. Given a current approximation $\tilde{x}$, the entire quadratic problem is the following:

$$\begin{array}{ll} \textbf{minimize} & \mathfrak{E}(U,V) \\ \textbf{subject to} & \forall s \in \mathcal{S}(\mathcal{G})\ \mathfrak{eqD}(s) = \mathcal{A}(s); \\ & \text{if } p \in \{0,\ \dots\ , N_{\mathcal{G},x}-1\} \cup \{|\mathcal{P}(\mathcal{G})| - N_{\mathcal{G},x},\ \dots\ , |\mathcal{P}(\mathcal{G})|-1\} \text{ then } V_p = 0; \\ & \text{if } p (\text{mod } N_{\mathcal{G},x}) = 0 \text{ or } (p+1)(\text{mod } N_{\mathcal{G},x}) = 0 \text{ then } U_p = 0. \end{array} \tag{5.5}$$

We will simplify the formulation of (5.5) by the concatenation of the two vectors U and V into one $\mathbf{U} = [\{U_i\}_{i=0}^{|\mathcal{P}(\mathcal{G})|-1} \mid \{V_i\}_{i=0}^{|\mathcal{P}(\mathcal{G})|-1}]$ and its set of indexes will be denoted by $\mathcal{P}_\mathbf{U}$. From now on, we will refer to $\mathfrak{E}$ and $\mathfrak{eqD}$ in their previous meaning but with the overall vector $\mathbf{U}$, so

$$\mathfrak{E}(\mathbf{U}) = \sum_{i,j\in\mathcal{P}_\mathbf{U}} q_{ij}\mathbf{U}_i\mathbf{U}_j + \sum_{i\in\mathcal{P}_\mathbf{U}} g_i\mathbf{U}_i + C \quad ,$$

where C is a constant and q_{ij}, g_i are the coefficients calculated directly from the previous definition (5.3) of $\mathfrak{E}$. Similarly $\mathfrak{eqD}$ should be redefined. Denote by λ_s, $s \in \mathcal{S}(\mathcal{G})$ the Lagrange multiplier corresponding to the equidensity constraint of square s. Let $\mathcal{B}(\mathcal{G}) = \mathcal{B}_U(\mathcal{G}) \cup \mathcal{B}_V(\mathcal{G})$ be the set of indexes of $\mathbf{U}$ that correspond to U_p and V_p defined in the boundary constraints in (5.5), where

$$\begin{array}{lll} \mathcal{B}_V(\mathcal{G}) = & \{p & :\ p \in [0\dots N_{\mathcal{G},x}-1] \cup [|\mathcal{P}(\mathcal{G})| - N_{\mathcal{G},x}\dots|\mathcal{P}(\mathcal{G})|-1]\} \\ \mathcal{B}_U(\mathcal{G}) = & \{p + |\mathcal{P}(\mathcal{G})| & :\ p(\text{mod } N_{\mathcal{G},x}) = 0 \text{ or } (p+1)(\text{mod } N_{\mathcal{G},x}) = 0,\ 0 \le p < |\mathcal{P}(\mathcal{G})|\}. \end{array}$$

Then the Lagrangian minimization functional is

$$\mathfrak{L}(\mathbf{U},\lambda,\zeta) = \mathfrak{E}(\mathbf{U}) + \sum_{s\in\mathcal{S}(\mathcal{G})} \lambda_s(\mathfrak{eqD}(s) - \mathcal{A}(s)) + \sum_{k\in\mathcal{B}(\mathcal{G})} \zeta_k\mathbf{U}_k \quad , \tag{5.6}$$

where ζ_k, $k \in \mathcal{B}(\mathcal{G})$, are the respective Lagrange multipliers. So, we are looking for a critical point of the Lagrangian function, which is expressed by the system of linear equations

$$\nabla\mathfrak{L}(\mathbf{U},\lambda,\zeta) = \begin{bmatrix} \nabla_\mathbf{U}\mathfrak{L}(\mathbf{U},\lambda,\zeta) \\ \nabla_\lambda\mathfrak{L}(\mathbf{U},\lambda,\zeta) \\ \nabla_\zeta\mathfrak{L}(\mathbf{U},\lambda,\zeta) \end{bmatrix} = 0 \quad . \tag{5.7}$$

There are several reasons that may cause the singularity of (5.7). The first kind of singularity can appear from possible empty squares. This can be treated by adding a summand to (5.6) which minimizes the total sum of all corrections $\beta\sum_i \mathbf{U}_i^2$, i.e., adds a 2β-term to the diagonal of $\nabla_\mathbf{U}\mathfrak{L}$. This will prevent the inclusion of zero-rows in $\nabla_\mathbf{U}\mathfrak{L}$, while possibly also bound the size of each correction in the solver below. Even if there is no singularity of the first kind the rank of $\nabla\mathfrak{L}(\mathbf{U},\lambda,\eta)$ is always less than its size by 1. This kind of singularity arises from the equations of equidensity constraints in (5.7). Their sum always equals zero. This is due to the fact that under

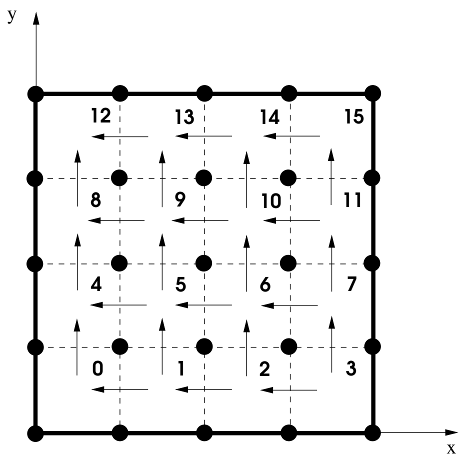


Figure 5.6: Directions of material circulation for equidensity constraints.

the boundary constraints the total amount of in-flows is always equal to the total amount of out-flows. In fact, the Lagrange summand for the equidensity constraints $\sum_{s\in\mathcal{S}(\mathcal{G})} \lambda_s(\mathfrak{eqd}(s) - \mathcal{A}(s))$ could be replaced by

$$\sum_{s\in\mathcal{S}(\mathcal{G})} (\lambda_s + K)(\mathfrak{eqd}(s) - \mathcal{A}(s))$$

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

$$K \sum_{s\in\mathcal{S}(\mathcal{G})} (\mathfrak{eqd}(s) - \mathcal{A}(s)) = 0 \ .$$

Thus, important are not the values of λ_s but only their *differences*, and the singularity can be treated by an additional constraint, say $\sum_s \lambda_s = 0$. The additional term for $\mathfrak{L}(\mathbf{U}, \lambda, \zeta)$ is $\eta \sum_s \lambda_s$, where η is a "pseudo-Lagrange" multiplier. The following proposition motivates the non-singularity of $\mathfrak{L}$ with $\sum_s \lambda_s = 0$.

Lemma 5.4.1. *Given a symmetric matrix $A_{n\times n}$. If $rank(A) = n-1$ then $rank(B) = n+1$, where B is the block matrix*

$$B = \left(\begin{array}{c|c} A_{n\times n} & x \\ \hline x^T & 0 \end{array}\right),$$

$x \in N_A \setminus \{0\}$.

Proof. Let $y = [y' \ y_{n+1}]^T$ be some vector in $\mathbb{R}^{n+1}$, where y' are the first n components of y. We will prove that if $By = 0$ then $y = 0$. The vector By can be written in the following block form

$$By = \left(\begin{array}{c} Ay' + y_{n+1}x \\ \hline x^T y' \end{array}\right).$$

Assume by negation that if y is the solution of $By = 0$ then not all components of y vanish (otherwise $rank(B) = n + 1$). The following sequence obliges $y_n + 1$ to be zero:

$$Ay' + y_{n+1}x = 0 \Rightarrow Ay' = -y_{n+1}x \Rightarrow x^T Ay' = -y_{n+1}x^T x \Rightarrow 0y' = 0 = -y_{n+1}x^T x\,.$$

If $y_{n+1} = 0$ then necessarily $Ay' = 0$. However, when $y' \in N_A \setminus \{0\}$ (we omit the case when $y' = 0$) then $y' = \alpha x$, where $\alpha \neq 0$. Thus,

$$By = \begin{pmatrix} \alpha Ax + y_{n+1}x \\ \hline \alpha x^T x \end{pmatrix}$$

and since $\alpha x^T x \neq 0$ we obtain a contradiction with $By = 0$. $\square$

Let us finally redefine the pseudo-Lagrangian functional $\mathfrak{L}$ for our problem

$$\mathfrak{L}(\mathbf{U}, \lambda, \eta) = \sum_{i,j \in \mathcal{P}_\mathbf{U}} q_{ij}\mathbf{U}_i\mathbf{U}_j + \sum_{i \in \mathcal{P}_\mathbf{U}} g_i\mathbf{U}_i + \beta \sum_{i \in \mathcal{P}_\mathbf{U}} \mathbf{U}_i^2 + \sum_{s \in \mathcal{S}(\mathcal{G})} \lambda_s(\mathfrak{eqd}(s) - \mathcal{A}(s)) + \eta \sum_{s \in \mathcal{S}(\mathcal{G})} \lambda_s\ . \tag{5.8}$$

Note that we have removed the constant C since it does not influence the minimization process, as well as the boundary constraint summand $\sum_{k \in \mathcal{B}(\mathcal{G})} \zeta_k \mathbf{U}_k$ since the respective $\mathbf{U}_k$ can be directly replaced by 0 in $\mathfrak{E}$ and $\mathfrak{eqd}$.

Assumption of the equality in the equidensity constraint formulation contradicts the real world situation in problems like placement, graph drawing, etc., i.e., the total (chip/drawing) area is always *bigger* than the total area of all the vertices (or modules). Thus, since the scope of activity of this assumption is strictly non local, we have removed it after receiving some positive results for the minimization under equality constraints. Let us now define the new minimization problem under inequality constraints

$$\begin{array}{ll} \textbf{minimize} & \mathfrak{E}(U, V) \\ \textbf{subject to} & \forall s \in \mathcal{S}(\mathcal{G})\ \mathfrak{eqd}(s) \leq \mathcal{A}(s) + \eta; \\ & \text{if } p \in \mathcal{B}_U(\mathcal{G}) \text{ then } U_p = 0; \\ & \text{if } p \in \mathcal{B}_V(\mathcal{G}) \text{ then } V_p = 0. \end{array} \tag{5.9}$$

5.5 Coarsening scheme

When the geometry of the problem is known we can choose a coarser grid by eliminating points in a geometrically-regular pattern as it presented in Figure 5.7. The correction computed at the coarse grid points will be interpolated to the fine grid in order to refine its current approximation. Let us introduce the notation distinguishing between fine and coarse level variables and functions. In this section we suggest the coarsening scheme for the layout correction problem.

As stated previously, at each point $p \in \mathcal{G}$ (solid circles in Figure 5.7) two variables which correspond to the horizontal and vertical corrections of the layout in

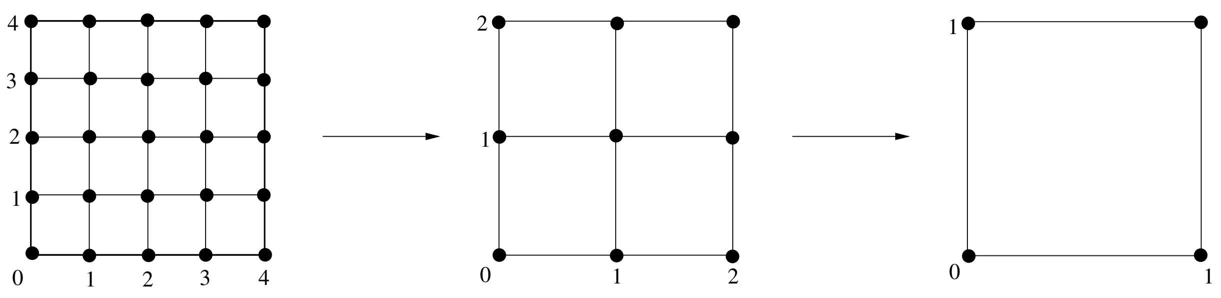


Figure 5.7: Standard geometric coarsening scheme. Each solid point contains two variables U and V of the corresponding level. These variables determine the horizontal and vertical corrections of the nodes positioned inside their neighbor squares, respectively.

its neighboring squares are defined. By lower and upper case letters we will refer to the variables, indexes and coefficients of the fine (u_i, i, q_j, etc.) and the coarse (U_I, I, Q_J, etc.) levels, respectively. The index letters f and c will be used to describe the energy (E) and pseudo-Lagrangian (L) functions at the fine and coarse levels, respectively.

Denote by E_f the energy function, i.e., the minimization part of the pseudo-Lagrangian (5.8) at the fine level, i.e.,

$$E_f = \sum_{ij} \bar{q}_{ij} u_i u_j + \sum p_i u_i \quad , \tag{5.10}$$

where $\bar{q}_{ij}$ (p_i) are the coefficients[1] of the second (first) order terms. Given a current approximation $\tilde{u}_i$ of the fine level variable u_i and a correction inherited from the coarse level variables U_I, u_i is refined by

$$u_i = \tilde{u}_i + \sum_{I \ni i} \alpha_{iI} U_I \ , \tag{5.11}$$

where the notation $\sum_{I \ni i}$ means that the sum is running over all I which include i and α_{iI} are the interpolation coefficients defined by the standard two-dimensional geometric coarse-to-fine projection operator $\uparrow_c^f$. In particular, if (for a matter of example) we will refer to the grid point in $\mathcal{G}_f$ by two-index coordinate $(2l, 2m)$ and its corresponding coarse grid point in $\mathcal{G}_c$ will be refereed by (L, M) (see Figure 5.7), then $\uparrow_c^f$ can be written as

$$u_{2l,2m} = \tilde{u}_{2l,2m} + U_{L,M}, \tag{5.12}$$

$$u_{2l+1,2m} = \tilde{u}_{2l+1,2m} + \frac{1}{2}(U_{L,M} + U_{L+1,M}), \tag{5.13}$$

$$u_{2l,2m+1} = \tilde{u}_{2l,2m+1} + \frac{1}{2}(U_{L,M} + U_{L,M+1}), \tag{5.14}$$

$$u_{2l+1,2m+1} = \tilde{u}_{2l+1,2m+1} + \frac{1}{4}(U_{L,M} + U_{L,M+1} + U_{L+1,M} + U_{L+1,M+1}). \tag{5.15}$$

[1]In order to remove the additional coefficient $\frac{1}{2}$ we will use "bar" notation $q_{ij} = 2\bar{q}_{ij}$

Let us formulate the transition phase between the fine level energy functional E_f to the coarse variables representation:

$$\begin{aligned}
E_f &= \sum_{ij} \bar{q}_{ij}(\tilde{u}_i + \sum_{I \ni i} \alpha_{iI} U_I)(\tilde{u}_j + \sum_{J \ni j} \alpha_{jJ} U_J) + \sum_i p_i(\tilde{u}_i + \sum_{I \ni i} \alpha_{iI} U_I) = \\
&= \sum_{ij} \bar{q}_{ij}(\tilde{u}_i\tilde{u}_j + \sum_{J \ni j} \tilde{u}_i \alpha_{jJ} U_J + \sum_{I \ni i} \tilde{u}_j \alpha_{iI} U_I + \sum_{\substack{I \ni i \\ J \ni j}} \alpha_{iI}\alpha_{jJ} U_I U_J) + \sum_i p_i(\tilde{u}_i + \sum_{I \ni i} \alpha_{iI} U_I) = \\
&= \sum_{ij} \bar{q}_{ij}\tilde{u}_i\tilde{u}_j + \sum_{ij} \bar{q}_{ij} \sum_{J \ni j} \tilde{u}_i \alpha_{jJ} U_J + \sum_{ij} \bar{q}_{ij} \sum_{I \ni i} \tilde{u}_j \alpha_{iI} U_I + \\
&+ \sum_{ij} \bar{q}_{ij} \sum_{\substack{I \ni i \\ J \ni j}} \alpha_{iI}\alpha_{jJ} U_I U_J + \sum_i p_i \tilde{u}_i + \sum_i \sum_{I \ni i} p_i \alpha_{iI} U_I = \\
&= C + \sum_I \big(2 \sum_{\substack{j \\ i \in I}} \bar{q}_{ij}\tilde{u}_j \alpha_{iI} + \sum_{i \in I} p_i \alpha_{iI}\big) U_I + \sum_{IJ} \big(\sum_{\substack{i \in I \\ j \in J}} \bar{q}_{ij}\alpha_{iI}\alpha_{jJ}\big) U_I U_J = \\
&= C + \sum_I P_I U_I + \sum_{IJ} \bar{Q}_{IJ} U_I U_J \quad ,
\end{aligned}$$

where $P_I = 2\sum_{\substack{j \\ i \in I}} \bar{q}_{ij}\tilde{u}_j\alpha_{iI} + \sum_{i \in I} p_i \alpha_{iI}$, $\bar{Q}_{IJ} = \sum_{\substack{i \in I \\ j \in J}} \bar{q}_{ij}\alpha_{iI}\alpha_{jJ}$ and the coarse level energy function will be

$$E_c = \sum_{IJ} \bar{Q}_{IJ} U_I U_J + \sum_I P_I U_I \ .$$

In order to express a uniform distribution of the vertices over the drawing area, for each square k at the fine level we define an equidensity equality or inequality constraint $\mathfrak{eqd}(k)$ of the form $\sum_i a_{ki}u_i = b_k$ or $\sum_i a_{ki}u_i \leq b_k$, respectively. The coarse equidensity constraints are constructed in a geometric-pattern manner as shown in Figure 5.8. Starting from a square $(0,0)$ the drawing area is filled by 2×2 squares. Each four-fine-squares pattern will represent a coarse square K.

Let k (K) be the index that runs over squares at fine (coarse) level. The expression "$k \in K$" will refer to the four fine squares that form a coarse square K. The K-th equidensity constraint of the coarse level is defined by $\sum_I A_{KI}U_I = B_K$ or $\sum_I A_{KI}U_I \leq B_K$ when equality or inequality conditions are employed, respectively. The coarse square equidensity constraint is formed by summation over the

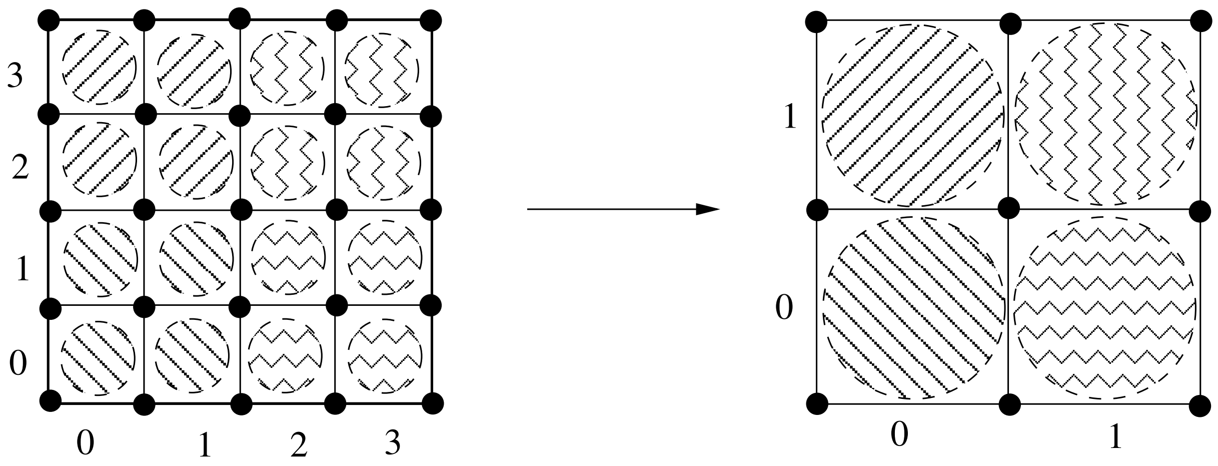


Figure 5.8: Constraints geometric coarsening. The equidensity constraints of every four similarly patterned squares at the fine level form a respectively patterned equidensity constraint of the coarse square.

four corresponding fine level squares

$$\begin{aligned}
\sum_{k\in K}\sum_{i} a_{ki}u_i - \sum_{k\in K} b_k &= \sum_{k\in K}\sum_{i} a_{ki}(\tilde{u}_i + \sum_{I\ni i}\alpha_{iI}U_I) - \sum_{k\in K} b_k \\
&= \sum_{k\in K}\big(\sum_{i} a_{ki}\tilde{u}_i + \sum_{i} a_{ki}\sum_{I\ni i}\alpha_{iI}U_I\big) - \sum_{k\in K} b_k \\
&= \sum_{k\in K}\sum_{i} a_{ki}\sum_{I\ni i}\alpha_{iI}U_I - B_K \\
&= \sum_{I}\sum_{i\in I}\sum_{k\in K} a_{ki}\alpha_{iI}U_I - B_K \\
&= \sum_{I} A_{KI}U_I - B_K \ ,
\end{aligned}$$

where $A_{KI} = \sum_{i\in I}\sum_{k\in K} a_{ki}\alpha_{iI}$, and $B_K = \sum_{k\in K}(b_k - \sum_i a_{ki}\tilde{u}_i)$. Similarly (in case of equality constraints), the additional η-constraint over all squares at the coarse level is inherited from the fine level $\sum_K D_K\Lambda_K = 0$, where $D_K = \sum_{k\in K} d_k$.

Residuals. In order to express a correction for the variables (given a current approximation), let us define a set of coarse residuals for the minimization equations and the equidensity constraints. If L_f is a pseudo-Lagrangian of the fine level system defined by

$$L_f = E_f + \sum_k \lambda_k(\sum_i a_{ki}u_i - b_k) + \eta\sum_k d_k\lambda_k \quad .$$

then the u_i-th residual of ∇L_f, where $i \in \{0 \ldots 2|\mathcal{P}(\mathcal{G})| - 1\}$, is

$$r_i^{\mathfrak{E}} = -p_i - \sum_j q_{ij}\tilde{u}_j - \sum_k \tilde{\lambda}_k a_{ki}.$$

Thus, the residual corresponding to the variable U_I of ∇L_c (where ∇L_c is the coarse level system of equations) will be

$$R_I^{\mathfrak{e}} = \uparrow_f^c r_i^{\mathfrak{e}} = \sum_{i \in I} \alpha_{iI} r_i^{\mathfrak{e}} \ ,$$

where α_{iI} are the anterpolation coefficients defined by $\uparrow_f^c = (\uparrow_c^f)^T$, see (26-29). The residual of the k-th equidensity constraint is

$$r_k^{\mathfrak{eqd}} = b_k - \sum_i a_{ki}\tilde{u}_i - \tilde{\eta} d_k \ ,$$

where k runs over all fine squares. Therefore, the coarse equidensity residual of square K will be $R_K^{\mathfrak{eqd}} = \uparrow_f^c r_k^{\mathfrak{eqd}} = \sum_{k \in K} r_k^{\mathfrak{eqd}}$. The residual of the η-constraint will be

$$r_\eta = -\sum_k d_k \tilde{\lambda}_k = R_H.$$

Full Approximation Scheme (FAS). FAS is a general multigrid strategy usually applied to nonlinear problems. For our goal, we consider FAS-like coarsening rules only partially. In fact, there is no need to construct FAS for the equality equidensity constraints since it is a linear problem that can be solved by the regular correction scheme. The FAS-like coarsening rules are needed and applied only on the set of equations derived from the equidensity inequalities (which represent the non-linear part of our problem). Thus, our scheme is a combination of the correction sceme for the energy equations derived from 5.10-5.11 and FAS-like rules for the remained equations. Denote by LP the linear part of the U_I-th equation in the system ∇L_c

$$LP = \sum_J Q_{IJ} U_J + \sum_K \Lambda_K A_{KI}.$$

From the FAS rule for the I-th coarse equation $LP = R_I^{\mathfrak{e}} + \textsf{Current approx. of the } I\textsf{-th } LP$ we can derive the I-th ∇L_c equation

$$\sum_J Q_{IJ} U_J + \sum_K \Lambda_K A_{KI} - R_I^{\mathfrak{e}} - \sum_J Q_{IJ} U_J^0 - \sum_K \Lambda_K^0 A_{KI} = 0,$$

where $U_J^0 = 0$ and $\Lambda_K^0 = \frac{1}{4}\sum_{k \in K} \tilde{\lambda}_k$. Similarly, the K-th square coarse equation for the equality constraint will be

$$\sum_I A_{KI} U_I + H D_K - R_K^{\mathfrak{eqd}} - \sum_I A_{KI} U_I^0 - H^0 D_K = 0 \qquad (5.16)$$

with only one change of relationship $=$ to $\leq$ for inequality case. The last equation for the H-constraint will be

$$\sum_K D_K \Lambda_K - R_H - \sum_K D_K \Lambda_K^0 = 0.$$

The correction received from a coarse level for the Lagrange multipliers λ_k will be $\lambda_k = \tilde{\lambda}_k + \Lambda_{K \ni k} - \Lambda_{K \ni k}^0$.

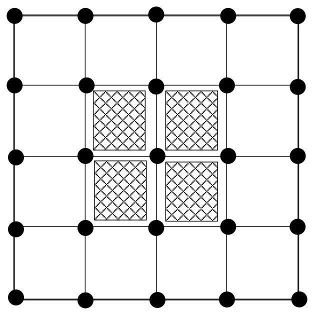


Figure 5.9: Example of a window $\mathcal{W}$: the filled squares are chosen to form a window.

5.6 Relaxation

In our problem the relaxation process is employed at two phases: (1) as the error's smoother of a given first approximation before the construction of the coarse level system and (2) in order to improve the first solution obtained immediately after interpolation from the coarse level. For this purpose we have developed the so called *Window relaxation* procedure which extracts from the entire system small subproblems related with $m \times m$ squares and solves them by direct application of some minimization method.

In fact, these subproblems are exactly the same quadratic minimization problems under appropriate constraints designed for small windows of squares. Let us define a window $\mathcal{W} = \{s \in \mathcal{S}(\mathcal{G}) |$ all squares within a rectangle$\}$ as presented in Figure 5.9 and $\overline{\mathcal{B}}(\mathcal{W})$ - the set of both horizontal and vertical correction variables indexes on the boundary of $\mathcal{W}$. In order to formulate the quadratic minimization problems we

- fix all $\{u_i | i \notin \bar{c}(s), s \in \mathcal{W}\}$ at their current approximation,
- choose the set of equidensity constraints for the appropriate squares $s \in \mathcal{W}$,
- define the boundary conditions, such that there will be no correction movement at the boundary of $\mathcal{W}$, i.e., u_i is fixed for $i \in \overline{\mathcal{B}}(\mathcal{W})$ and
- solve the problem for $\mathcal{W}$.

The running time of the entire relaxation process strongly depends on the algorithm for solving one window. There exists many different versions of well known algorithms for the quadratic minimization under linear inequality constraints (for a survey see [3]). However, in order to keep the coefficient of the linear running time low, we have implemented a simple algorithm for solving a single window, as presented in **SingleWindowSolver**. There is no real demand to achieve the global minimum in every window. Due to the multilevel nature, at each level we have to achieve a small correction which reduces the energy and smooths the error. The entire correction will be accumulated from all levels of the hierarchy.

Our heuristics represents a simplified version of the *active set method.* The algorithm is iterative. At each iteration t, for a given $\tilde{u}$ we first extract the set (denoted by S_t) of squares for which the respective inequality equidensity constraints are violated or almost violated:

$$S_t = \{s \in \mathcal{W} \mid \mathfrak{eqd}(s) > \mathcal{A}(s) - \epsilon\} ,$$

where ϵ is positive and sufficiently small. Then we will turn the inequality constraints of S_t to equalities ignoring the other inequality constraints and formulate the pseudo-Lagrangian function $\mathfrak{L}_{\mathcal{W}}$ for $\mathcal{W}$ with fixed u_i for $i \in \overline{\mathcal{B}}(\mathcal{W})$. Similarly to (5.8), let $\mathbf{U}_{\mathcal{W}}$ be the set of variables inside $\mathcal{W}$ with the respective set of indexes $\mathcal{P}_{\mathbf{U}_{\mathcal{W}}}$. For every $u_i \in \mathbf{U}_{\mathcal{W}}$ we define its correction variable δ_i and reformulate the pseudo-Lagrangian as a function for the correction δ_i variables as follows

$$\begin{aligned}
&\mathfrak{L}_{\mathcal{W}}(u, \delta, \lambda, \eta) = \\
&\quad \sum_{i,j \in \mathcal{P}_{\mathbf{U}_{\mathcal{W}}}} q_{ij}(\tilde{u}_i + \delta_i)(\tilde{u}_j + \delta_j) + \sum_{\substack{i \in \mathcal{P}_{\mathbf{U}_{\mathcal{W}}} \\ j \notin \mathcal{P}_{\mathbf{U}_{\mathcal{W}}}}} q_{ij}(\tilde{u}_i + \delta_i)\tilde{u}_j + \sum_{i \in \mathcal{P}_{\mathbf{U}_{\mathcal{W}}}} g_i(\tilde{u}_i + \delta_i) + \\
&\qquad \beta \sum_{i \in \mathcal{P}_{\mathbf{U}_{\mathcal{W}}}} (\tilde{u}_i + \delta_i)^2 + \sum_{s \in S_t} \lambda_s(\mathfrak{eqd}(s) - \mathcal{A}(s)) + \eta \sum_{s \in S_t} \lambda_s , \quad (5.17)
\end{aligned}$$

where $\tilde{u}_i$ is the current value of u_i. Solving $\nabla \mathfrak{L}_{\mathcal{W}} = 0$ we obtain the correction for $\mathbf{U}_{\mathcal{W}}$ which confines the respective active set variables to the boundary of the equality constraints manifold. However, while accepting this correction we can violate other inequality constraints that were already satisfied at the iteration $t-1$. Let us call this set of new unsatisfied constraints $\overline{S}_t$. One way to overcome this problem is to accept only a partial correction $\varepsilon\delta_i$, $i \in \mathcal{P}_{\mathbf{U}_{\mathcal{W}}}$, where ε is the smallest number which brings some constraint from $\overline{S}_t$ to equality. It is easy to see that accepting the correction $\varepsilon\delta_i$ does not violate other constraints (those which produce a larger ε) from $\overline{S}_t$. At this point we accept this partial correction and continue to the next iteration $t+1$ excluding from the redefined S_t the set of satisfied constraints from S_t with positive Lagrange multipliers λ_s.

```
**SingleWindowSolver**($\mathcal{W}$, $\tilde{u}$)
**begin**
  $t = 0$
  **Repeat** until "optimal enough" (see below)
    $S_t$ = {the violated equidensity constraints} \ {the satisfied constraints with $\lambda_s > 0$}
    **Solve** $\nabla \mathfrak{L}_{\mathcal{W}} = 0$ and extract the smallest $\varepsilon$
    **Accept** the correction $\tilde{u} = \tilde{u} + \varepsilon\delta$
    $t = t + 1$
**end**
```

In order to achieve the correction for all variables, we will sequentially cover by these windows the entire drawing area. For computational reasons we have chosen to apply this relaxation for very small windows (of size 4×4) with an overlap. For our application we have chosen the red-black ordering of windows. First, all windows at odd (red) positions are considered, then the windows at even (black) positions, and finally the previous relaxations are repeated for windows shifted by half of their size in both horizontal and vertical directions. In other words, the entire area is covered three times during one relaxation sweep. Nevertheless, similar numerical results have been obtained if the relaxation order was changed to the lexicographic scan with the same half-window sized shifts, i.e., the horizontal and the vertical overlapping.

The entire algorithm for the two-dimensional layout correction is summarized below in Algorithm **2D-layout-correction**. The superscript index in the following procedures refers to the number of level.

```
2D-layout-correction(graph $G$, current layout $\bar{x}$)
begin
  Apply sufficiently many times
    Construct and initialize $\mathcal{G}^0$, $U^0$
    Initialize the system of equations $\nabla\mathfrak{L}^0$
    Define $\mathcal{C}^0$ be the set of equidensity constraints
    ml-correction($\mathcal{G}^0$, $\nabla\mathfrak{L}^0$, $\mathcal{C}^0$, $U^0$)
    Update $\bar{x}$
Return $\bar{x}$
```

```
ml-correction($\mathcal{G}^i$, $\nabla\mathfrak{L}^i$, $\mathcal{C}^i$, $U^i$)
begin
  If $\mathcal{G}^i$ is small enough
    Solve the problem exactly
  Else
    Set $\mathbf{U}^i = 0$
    Apply sufficiently many times Window relaxation
    Construct and initialize $\mathcal{G}^{i+1}$, $U^{i+1}$
    Initialize the system of equations $\nabla\mathfrak{L}^{i+1}$
    Define $\mathcal{C}^{i+1}$ be the set of equidensity constraints
    ml-correction($\mathcal{G}^{i+1}$, $\nabla\mathfrak{L}^{i+1}$, $\mathcal{C}^{i+1}$)
    Interpolate from level $i+1$ to level $i$, i.e., $U^i = \uparrow_c^f U^{i+1}$
    Apply sufficiently many times Windows relaxation
Return $U^i$
```

In spite of the promising results presented in the following section, the algorithm has not yet been optimized. However, even now it is already clear that several parameters (which growth, certainly, influence the running time) can be kept very small. For example: (a) the number of *Window relaxation* iterations can be fixed between 1 and 3; (b) "optimal enough" in **SingleWindowSolver** means less than 6 iterations and (c) the size of $\mathcal{W}$ in **SingleWindowSolver** is very robust, i.e., the same results can be obtained with the sizes 4x4, 8x8 and 16x16.

5.7 On the single window boundary conditions

The boundary condition is very important component of the single window solver since it can permit various types of the movement inside the window and over the boundary. The multilevel nature of the entire scheme allows to perform only local movements. Therefore, it is advisable to define the boundary conditions so that the movement over the boundary will be minimized in order to ensure only local operations. We have experimented with two types of boundary conditions:

1. $\overline{\mathcal{B}}(\mathcal{W})$ - the set of both horizontal and vertical correction variables indexes on the boundary of $\mathcal{W}$;

2. $\mathcal{B}(\mathcal{W})$ - the set of both horizontal and vertical correction variables indexes on the boundary of $\mathcal{W}$ which directions are perpendicular to the corresponding boundary.

5.8 Examples of graph drawing layout correction

As it was 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 briefly present several results of the two-dimensional layout correction algorithm. The set of examples is shown in Figures 5.11, 5.18 and 5.19, 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.

We begin with a circle graph (Figure 5.11, row a) which is initially placed as a straight line, such that its first and last vertices are at the opposite sides of the drawing area. The difficulty of this example lies in the first relaxation. It will randomly shift the vertices towards either sides of the area and the algorithm must cope with this initial random disturbance. The second example consists of a mesh graph with three holes (Figure 5.11, row b). It is intended to demonstrate that the empty space stays empty and the energy does not increase. More complicated examples are shown in Figure 5.11, rows c and d. The initial optimal positions of the mesh's vertices were randomly changed by independent shifts in different directions within a distance d, where

$$d \;\leq\; \begin{cases} 2w_s & \text{in example c} \\ 4w_s & \text{in example d.} \end{cases}$$

Let us call these meshes M_1 and M_2, respectively. While the correction of M_1 is looking really nice, the example of M_2 demonstrates a weak point in our algorithm which certainly must be improved. The initial layout c1 is much more complicated than d2, where the desired final layout is similar to c2. Thus, by applying additional V-cycles we can definitely come to a better solution and we can experimentally show it. At this moment, this weak point of our algorithm is in the main driving routine **2D-layout-correction**. This problem is discussed in Section 5.9.

A typical example of the energy behaviour is presented at Figures 5.12-5.14. These figures refer to the mesh example at Figure 5.11-c. The general energy minimization progrss is shown at Figure 5.12. In this example the driving routine consists of the sequence of various grid size V-cycles ("V-cycles" axis). Each odd V-cycle solves the correction problem for the 16x16 grid while even V-cycles improve the previous iterations with the grid 32x32. In all our experiments every level of such V-cycle (**ml-correction** scheme) contains only one iteration of Window relaxation at the coarsening and the interpolation stages. Next two Figures 5.13 and 5.14 show an energy behaviour of Window relaxations (without V-cycles) for 16x16 grid iterations and exchangable 16x16 and 32x32 grid sizes, respectively.

We have continued to analyze the behaviour of the algorithm on the compressed mesh graph. The first result is presented in Figure 5.15. The lines shown in both pictures are deviation vectors of graph nodes from their near-to-optimal[2] placement. The upper picture is almost filled by these vectors since all mesh nodes were concentrated at the bottom left corner. After several iterations of the algorithm, all nodes have been moved to their near-to-optimal positions (see the bottom image) and the deviations became very small.

More complicated example is shown in Figure 5.16 in which the 64x64 mesh graph was not only compressed but also slightly randomly permuted and reinforced by 50 additional randomly chosen diagonals (Figure 5.16-a). The final result of the algorithm is presented in Figure 5.16-b where all vertices are placed almost at their optimums. The graph of energy function convergence is shown in Figure 5.16-c. After less than two FMG cycles the total energy was very close to its real minimum and additional iterations have only slightly corrected the layout. Each point at the horizontal axis in Figure 5.16-c corresponds to a V-cycle of 2-FMG driving routine. Such driving routine works with the following sequence of V-cycles at scales: 2, 2, 4, 4, 8, 8, ..., 128, 128, 2, 2, 4,4, etc. . This 2-FMG cycle was also applied on the 64x64 compressed mesh with three holes. The initial and final layouts are presented in Figures 5.17-a and 5.17-b, respectively.

Two additional examples contain the layout corrections for graphs whose vertices have non-equal volumes (see Figures 5.18 and 5.19). In both cases the initial layout of these graphs was random.

[2]In this case, the optimal placement is not that in which all vertices are at the respective mesh positions but there exist better configurations like those with the rounded corners.

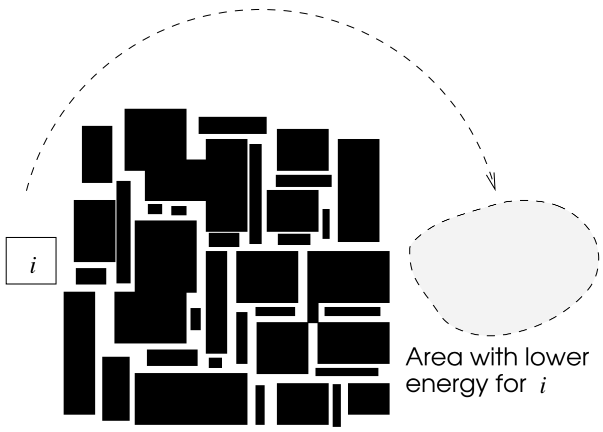


Figure 5.10: Example of a cluster of the vertices with one vertex stackled aside.

5.9 Future work

The research reported here only scratches the surface of what is possible using the multigrid algorithms for the constrained optimization problems. We have shown only one application of this class of problems namely, the graph drawing layout correction.

During the future work on the graph drawing application, it is very important to improve the driving routine of the V-cycle. The iterative correction of the vertex position represents a movement of this vertex. Each iteration prolongates the vertex path by the short local movement towards the position which minimizes the energy subject to the equidensity constraints. Sometimes, when the nodes are arranged in dense clusters (see Figure 5.10), there is a difficulty when a vertex aspires to be relocated from one side of this cluster to another, where the energy will be lower. Similar situation (with only few stacked vertices) is shown in the right side of M_2 in Figure 5.11-d2. The problem can be easily explained: the existance of large amount of material does not allow the vertex to pass through the cluster but only to bend it at best. This problem can be treated by an adaptive V-cycle driving routine that will be able to temporarily scale the vertex sizes and/or the grid. This can give the vertex an opportunity to move between the vertices of the cluster. Let us summarize the key goals of our future work:

- to define more precisely the 'stopping criteria' for the main iterative processes;
- to automatically decide upon the starting grid size;
- to fix the desired size of the sufficient per vertex correction in **ml-correction**;
- to make the driving routine adaptive in the sense of dynamic changes for better grid sizes, window sizes, etc.;
- to check the convergence speed of different Window relaxation schemes;

- to check the infuence and the convergence speed of many non-linear V-cycles;
- to design a local relaxation for individual displacement of nodes;
- to enlarge the benchmark to more complicated examples.

Much more advanced goal consists of the formulation of the multigrid principles on the general geometric constrained optimization problems.

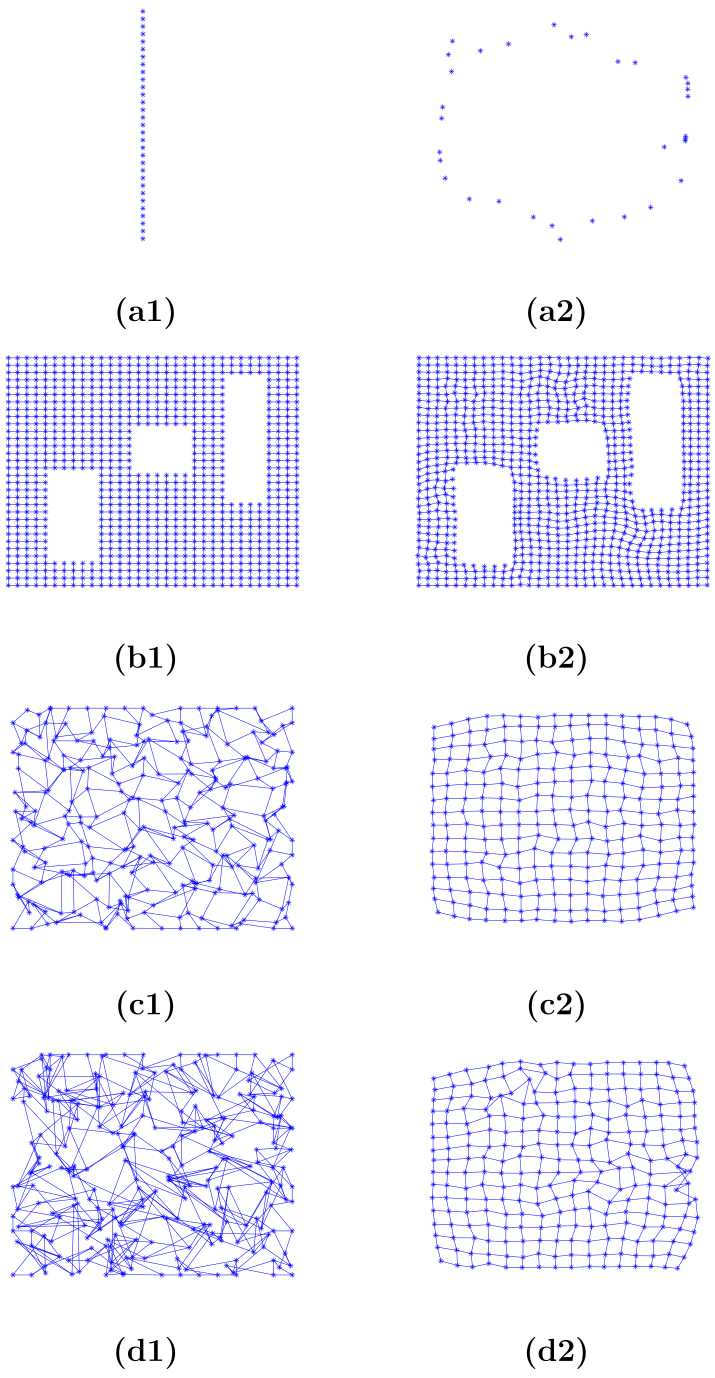


Figure 5.11: Examples for the 2D-layout of graphs with equal vertices.

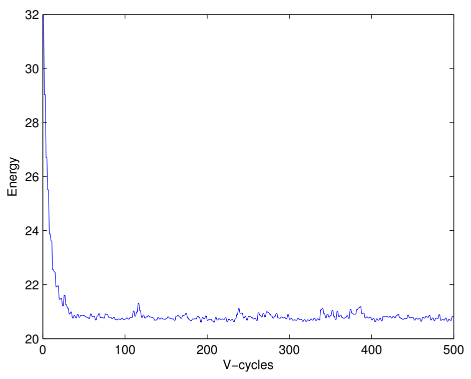


Figure 5.12: Energy behaviour of the mesh at Figuure 5.11-c.

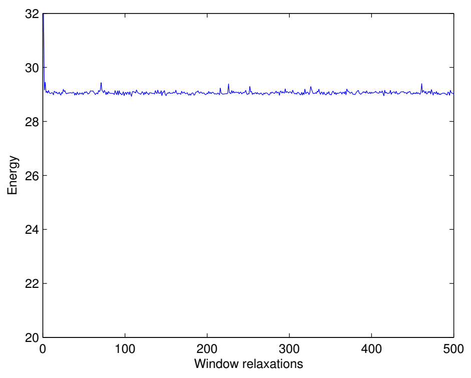


Figure 5.13: Energy behaviour of Window relaxation iterations (16x16 grid) of the mesh at Figuure 5.11-c.

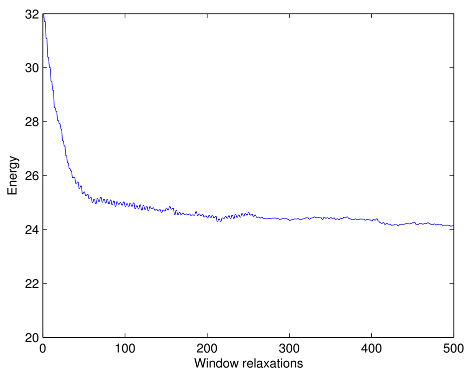
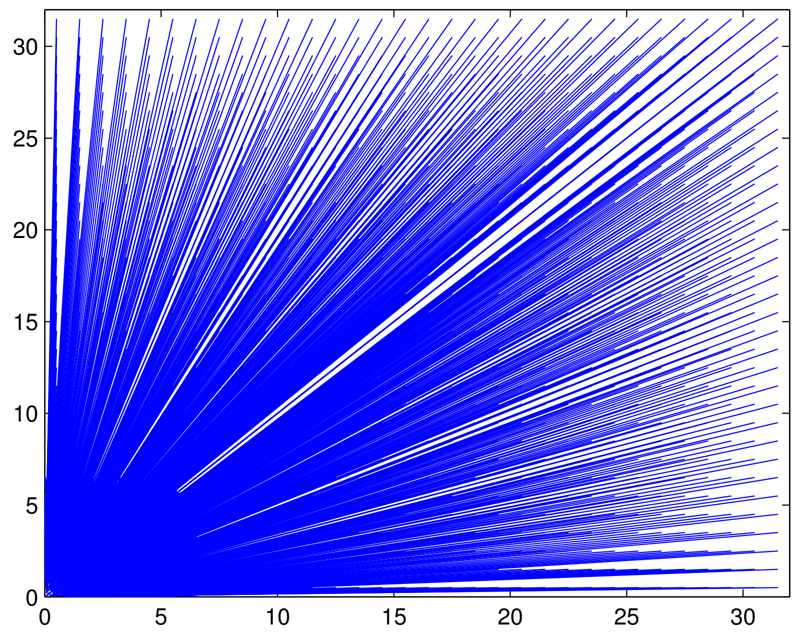
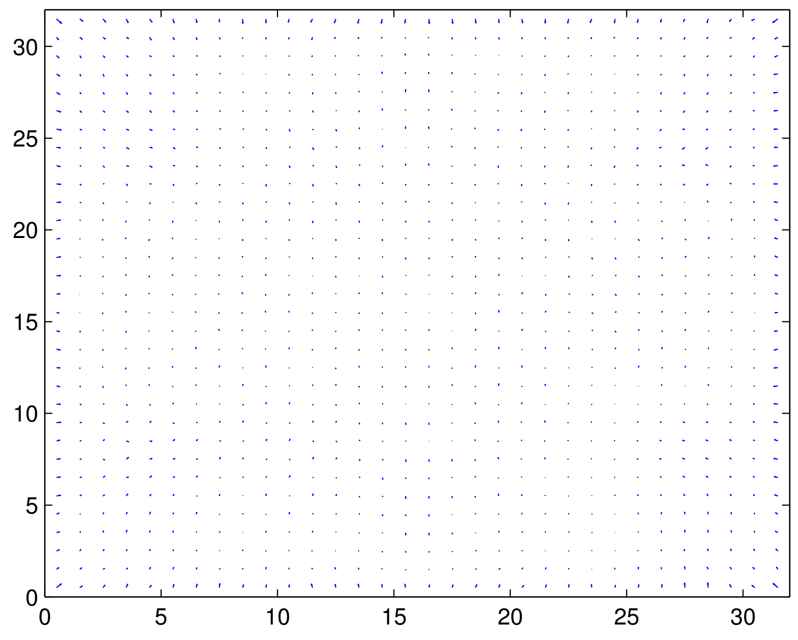


Figure 5.14: Energy behaviour of Window relaxation iterations (16x16 and 32x32 grids) of the mesh at Figuure 5.11-c.

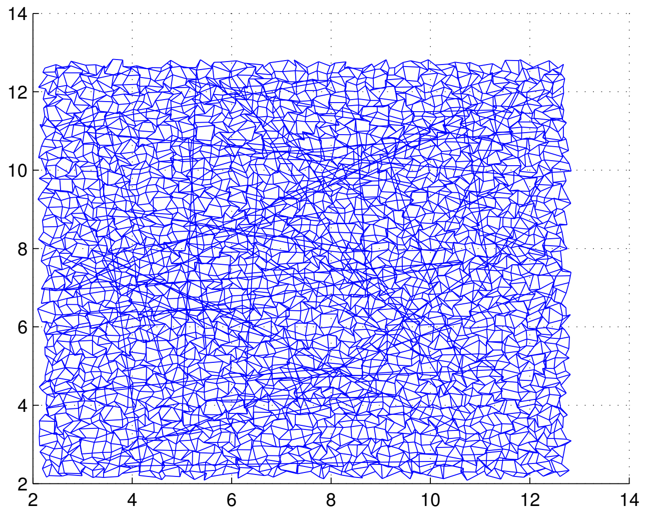


(a)

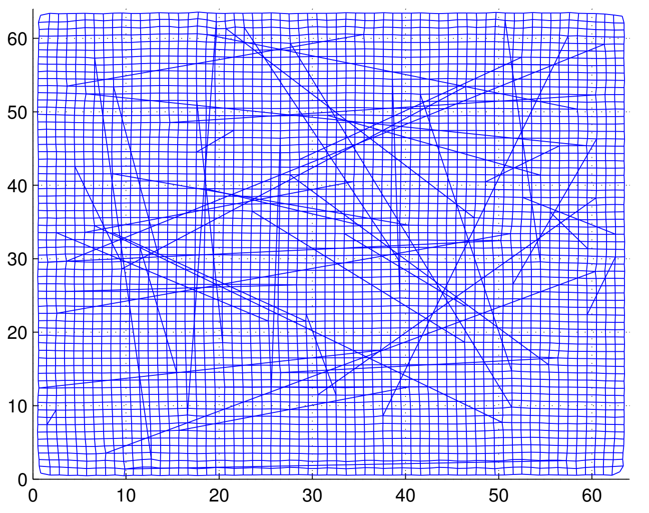


(b)

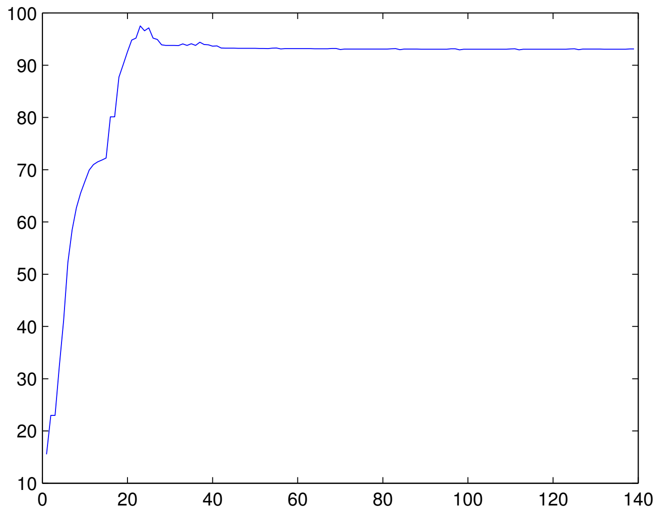
Figure 5.15: An example for the 32x32 mesh layout.



(a)

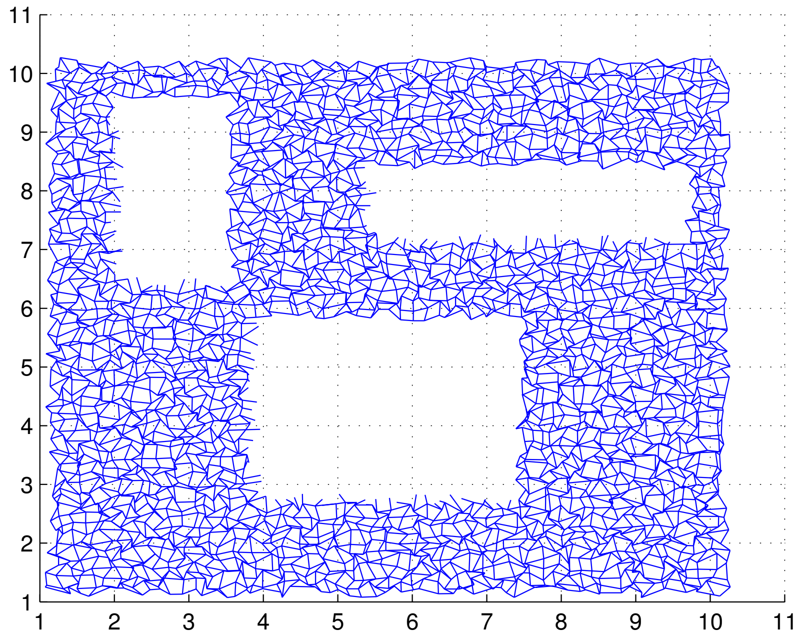


(b)

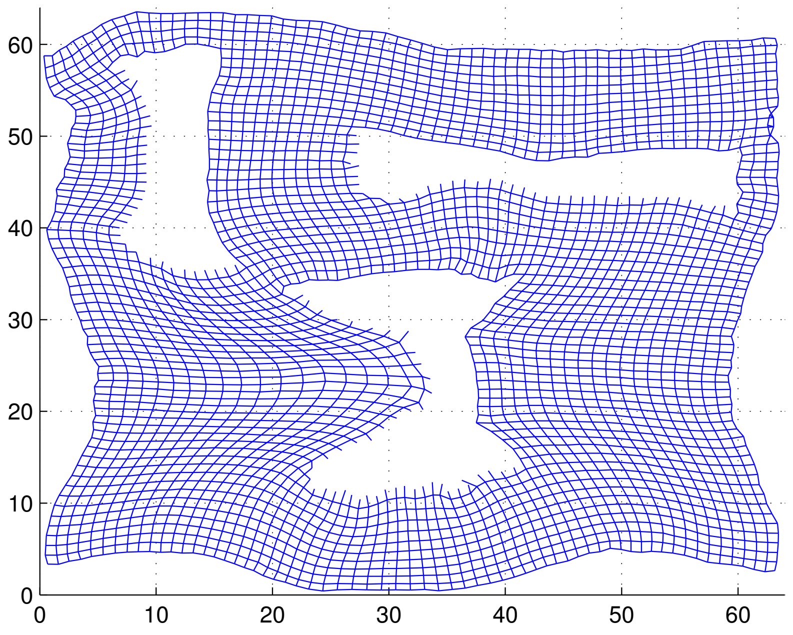


(c)

Figure 5.16: An example for the layout of the 64x64 mesh with additional random edges.



(a)



(b)

Figure 5.17: An example for the 64x64 holed mesh layout.

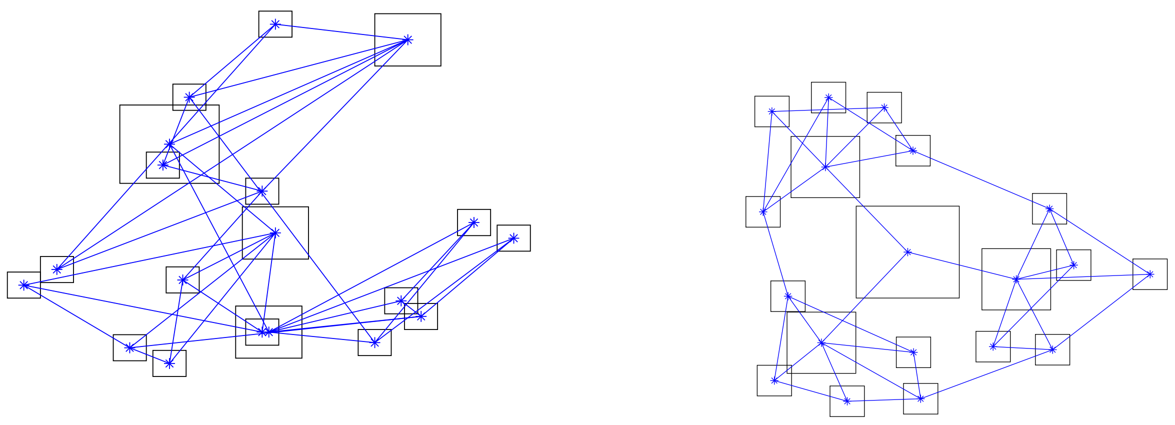


Figure 5.18: An example for the 2D-layout of the graph with non-equal volumes.

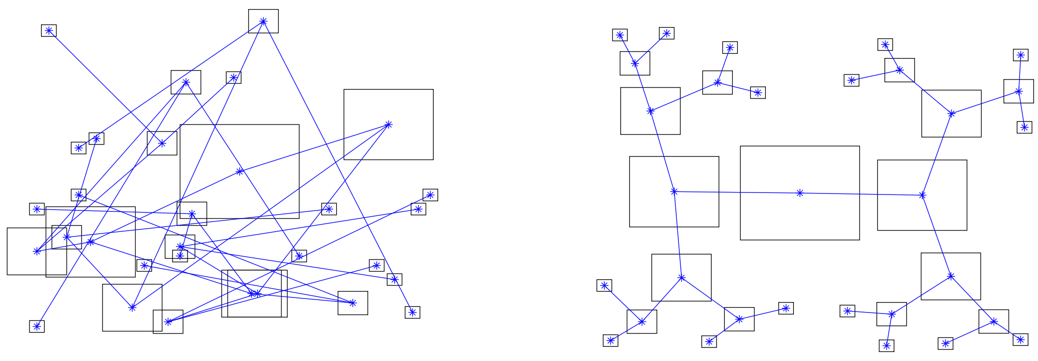


Figure 5.19: An example for the 2D-layout of 5-level binary tree with non-equal vertices.

Bibliography

[1] Amine Abou-Rjeili and George Karypis. Multilevel algorithms for partitioning power-law graphs. In *IPDPS*, 2006.

[2] Charles J. Alpert, Jen-Hsin Huang, and Andrew B. Kahng. Multilevel circuit partitioning. In *Design Automation Conference*, pages 530–533, 1997.

[3] M. Avriel. *Nonlinear Programming: Analysis and Methods*. Dover Publications, 2003.

[4] N.S. Bakhvalov. On the convergence of a relaxation method under natural constraints on an elliptic operator. *Zhurnal vychislitel'noj matemaki i matematicheskoj fiziki*, 6:861–883, 1966.

[5] R. Banos, C. Gil, J. Ortega, and F.G. Montoya. Multilevel heuristic algorithm for graph partitioning. In *Applications of Evolutionary Computing*, pages 143–153, 2003.

[6] R. Bar-Yehuda, G. Even, J. Feldman, and J. Naor. Computing an optimal orientation of a balanced decomposition tree for linear arrangement problems. *Journal of Graph Algorithms and Applications*, 5(4):1–27, 2001.

[7] S.T. Barnard, A. Pothen, and H. Simon. A spectral algorithm for envelope reduction of sparse matrices. *Numerical Linear Algebra with Applications*, 2(4):317–334, 1995.

[8] Stephen T. Barnard and Horst D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6:101–107, 1994.

[9] G. Di Battista, P. Eades, R. Tamassia, and I. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.

[10] A. Brandt. Multi-level adaptive technique (MLAT) for fast numerical solutions to boundary value problems. In H. Cabannes and R. Temam, editors, *Lecture Notes in Physics 18*, pages 82–89, pub-SV:adr, 1973. Proc. 3rd Int. Conf. Numerical Methods in Fluid Mechanics, pub-SV.

[11] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Math. Comp.*, 31(138):333–390, April 1977.

[12] A. Brandt. Algebraic multigrid theory: The symmetric case. *Jornal of Appl. Math. Comp.*, 19:23–56, 1986. Preliminary proceedings of the International Multigrid Conference, April 6–8, 1983, Copper Mountain, CO.

[13] A. Brandt. General highly accurate algebraic coarsening. *Electronic Trans. Num. Anal. 10 (2000) 1-20*, 2000.

[14] A. Brandt. Multiscale scientific computation: Review 2001. In T. Barth, R. Haimes, and T. Chan, editors, *Multiscale and Multiresolution methods (Proceeding of the Yosemite Educational Symposium, October 2000)*. Springer-Verlag, 2001.

[15] A. Brandt, S. McCormick, and J. Ruge. Algebraic multigrid (AMG) for automatic multigrid solution with application to geodetic computations. Technical report, Institute for Computational Studies, Fort Collins, CO, POB 1852, 1982.

[16] A. Brandt, S. McCormick, and J. Ruge. Algebraic multigrid (AMG) for sparse matrix equations. In D. J. Evans, editor, *Sparsity and its Applications*, pages 257–284, 1984.

[17] A. Brandt and D. Ron. Chapter 1 : Multigrid solvers and multilevel optimization strategies. In J. Cong and J. R. Shinnerl, editors, *Multilevel Optimization and VLSICAD*. Kluwer, 2003.

[18] A. Brandt, D. Ron, and D. Amit. Multi-level approaches to discrete-state and stochastic problems. In W. Hackbush and U. Trottenberg, editors, *Multigrid Methods II*, pages 66–99. Springer-Verlag, 1986.

[19] Achi Brandt. Rigorous quantitative analysis of multigrid, I: Constant coefficients two-level cycle with L_2-norm. *SIAM Journal on Numerical Analysis*, 31(6):1695–1730, 1994.

[20] Achi Brandt and Dorit Ron. Renormalization multigrid (rmg): Statistically optimal renormalization group flow and coarse-to-fine monte carlo acceleration. *J. Stat. Physics*, 102(231), 2001.

[21] U. Brenner and A. Rohe. An effective congestion driven placement framework, 2002.

[22] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A multigrid tutorial: second edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[23] Andrew E. Caldwell, Andrew B. Kahng, and Igor L. Markov. Can recursive bisection alone produce routable placements? In *Design Automation Conference*, pages 477–482, 2000.

[24] V. Campos, F. Glover, M. Laguna, and R. Martí. An experimental evaluation of a scatter search for the linear ordering problem. *Journal of Global Optimization*, 21(4):397–414, 2001.

[25] Alberto Caprara and Juan José Salazar González. Laying out sparse graphs with provably minimum bandwidth. *INFORMS Journal on Computing*, 17(3):356–373, 2005.

[26] Tony F. Chan, Jason Cong, Tianming Kong, and Joseph R. Shinnerl. Multilevel optimization for large-scale circuit placement. In *ICCAD '00: Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, pages 171–176, Piscataway, NJ, USA, 2000. IEEE Press.

[27] Tony F. Chan, Jason Cong, Michail Romesis, Joseph R. Shinnerl, Kenton Sze, and Min Xie. mpl6: a robust multilevel mixed-size placement engine. In *ISPD*, pages 227–229, 2005.

[28] Tony F. Chan, Jason Cong, Joseph R. Shinnerl, Kenton Sze, and Min Xie. mpl6: enhanced multilevel mixed-size placement. In *ISPD*, pages 212–214, 2006.

[29] C. Chang, J. Cong, D. Pan, and X. Yuan. Multilevel global placement with congestion control. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 22:395–409, 2003.

[30] C.-K. Cheng. Linear placement algorithm and applications to VLSI design. *Netw.*, 17(4):439–464, 1987.

[31] J. Cong and J. R. Shinnerl, editors. *Multilevel Optimization and VLSICAD*. Kluwer, 2003.

[32] Jason Cong, Jie Fang, Min Xie, and Yan Zhang. Mars-a multilevel full-chip gridless routing system. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 24(3):382–394, 2005.

[33] Jason Cong and M'Lissa Smith. A parallel bottom-up clustering algorithm with applications to circuit partitioning in VLSI design. In *Design Automation Conference*, pages 755–760, 1993.

[34] Gianna M. Del Corso and Francesco Romani. Heuristic spectral techniques for the reduction of bandwidth and work-bound of sparse matrices. Technical Report TR-01-02, Universitá di Piza, Dipartimento di Informatica, 2001.

[35] Gianna M. Del Corso and Francesco Romani. Heuristic spectral techniques for the reduction of bandwidth and work-bound of sparse matrices. *Numerical Algorithms*, 28(1–4):117–136, December 2001.

[36] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, pages 157–172, New York, NY, USA, 1969. ACM Press.

[37] T. Davis. University of florida sparse matrix collection. *NA Digest*, 97(23), 1997.

[38] J. Díaz, M. D. Penrose, J. Petit, and M. Serna. Approximating layout problems on random geometric graphs. *Journal of Algorithms*, 39(1):78–116, 2001.

[39] J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002.

[40] J. Díaz, J. Petit, M. Serna, and L. Trevisan. Approximating layout problems on random graphs. *Discrete Mathematics*, 235(1–3):245–253, 2001.

[41] Ding-Zhu Du and P. Pardalos. *Handbook of Combinatorial Optimization*. Kluwer Academic Publishers, 1999.

[42] G. W. Dueck and J. Jeffs. A heuristic bandwidth reduction algorithm. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 18:97–108, 1995.

[43] P. A. Eades. A heuristic for graph drawing. In *Congressus Numerantium*, volume 42, pages 149–160, 1984.

[44] Hans Eisenmann and Frank M. Johannes. Generic global placement and floorplanning. In *DAC '98: Proceedings of the 35th annual conference on Design automation*, pages 269–274, New York, NY, USA, 1998. ACM Press.

[45] R.P. Fedorenko. A relaxation method for solving elliptic difference equations. *Zhurnal vychislitel'noj matemaki i matematicheskoj fiziki*, 1:922–927, 1961.

[46] R.P. Fedorenko. The speed of convergence of one iteration process. *Zhurnal vychislitel'noj matemaki i matematicheskoj fiziki*, 4:559–563, 1964.

[47] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991.

[48] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63, New York, NY, USA, 1974. ACM Press.

[49] Michael R. Garey and David S. Johnson. *Computers and Interactability. A Guide to the Theory of NP-Completeness.* A Series of Books in the Mathematical Sciences. Freemann And Company, 1979.

[50] A. George and A. Pothen. An analysis of spectral envelope reduction via quadratic assignment problems. *SIAM Journal on Matrix Analysis and Applications*, 18(3):706–732, 1997.

[51] K. Hall. An $r-$dimensional Quadratic Placement Algorithm. *Management Science*, 17:217–229, 1970.

[52] D. Harel and A. Inger. On the aesthetic layout of higraphs. *submitted*, 2006.

[53] David Harel. On visual formalisms. *Commun. ACM*, 31(5):514–530, 1988.

[54] L. H. Harper. Optimal assignments of numbers to vertices. *Journal of the Society for Industrial and Applied Mathematics*, 12(1):131–135, March 1964.

[55] Bruce Hendrickson and Robert Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing*, 16(2):452–469, 1995.

[56] Bruce Hendrickson and Robert W. Leland. A multi-level algorithm for partitioning graphs. In *Supercomputing*, 1995.

[57] Steven Bradish Horton. *The Optimal Linear Arrangement Problem: Algorithms And Approximation.* PhD thesis, Georgia Institute of Technology, May 1997.

[58] Bo Hu and Malgorzata Marek-Sadowska. Fine granularity clustering for large scale placement problems. In *ISPD '03: Proceedings of the 2003 international symposium on Physical design*, pages 67–74, New York, NY, USA, 2003. ACM Press.

[59] Y. F. Hu and J. A. Scott. A multilevel algorithm for wavefront reduction. *SIAM J. Sci. Comput.*, 23(4):1352–1375, 2001.

[60] Sung Woo Hur and John Lillis. Mongrel: hybrid techniques for standard cell placement. In *ICCAD '00: Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, pages 165–170, Piscataway, NJ, USA, 2000. IEEE Press.

[61] Zhe-Wei Jiang, Tung-Chieh Chen, Tien-Chang Hsu, Hsin-Chen Chen, and Yao-Wen Chang. Ntuplace2: a hybrid placer using partitioning and analytical techniques. In *ISPD*, pages 215–217, 2006.

[62] Martin Juvan and Bojan Mohar. Optimal linear labelings and eigenvalues of graphs. *Discrete Appl. Math.*, 36(2):153–168, 1992.

[63] Andrew B. Kahng and Qinke Wang. A faster implementation of aplace. In *ISPD*, pages 218–220, 2006.

[64] G. Karypis and V. Kumar. hmetis 1.5: A hypergraph partitioning package. *Tech. Report, Dept. of Computer Science, Univ. of Minnesota*, 1998.

[65] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: applications in vlsi domain. *IEEE Trans. Very Large Scale Integr. Syst.*, 7(1):69–79, 1999.

[66] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report TR 95-035, 1995.

[67] S. Kirkpatrick. Models of disordered systems. In C. Castellani, editor, *Lecture Notes in Physics 149*. Springer-Verlag, 1981.

[68] Y. Koren and D. Harel. Multi-scale algorithm for the linear arrangement problem. *Proceedings of 28th Inter. Workshop on Graph-Theoretic Concepts*, 2002.

[69] Shinichi Kouda, Chikaaki Kodama, and Kunihiro Fujiyoshi. Improved method of cell placement with symmetry constraints for analog ic layout design. In *ISPD*, pages 192–199, 2006.

[70] Y. Lai and K. Williams. A survey of solved problems and applications on bandwidth, edgesum, and profile of graphs. *J. Graph Theory*, 31:75–94, 1999.

[71] Jianhua Li and Laleh Behjat. Net cluster: a net-reduction based clustering preprocessing algorithm. In *ISPD*, pages 200–205, 2006.

[72] Zhuoyuan Li, Xianlong Hong, Qiang Zhou, Shan Zeng, Jinian Bian, Hannah Honghua Yang, Vijay Pitchumani, and Chung-Kuan Cheng. Integrating dynamic thermal via planning with 3d floorplanning algorithm. In *ISPD*, pages 178–185, 2006.

[73] K. Marriott, P. Stuckey, V. Tam, and W. He. Removing node overlapping in graph layout using constrained optimization, 2003.

[74] Kurt Mehlhorn and Stefan Näher. Leda: a platform for combinatorial and geometric computing. *Commun. ACM*, 38(1):96–102, 1995.

[75] G.-J. Nam and J. Cong. *Modern circuit placement.* Springer Publishers, 2007.

[76] Aaron N. Ng, Igor L. Markov, Rajat Aggarwal, and Venky Ramachandran. Solving hard instances of floorplacement. In *ISPD*, pages 170–177, 2006.

[77] J. Petit. Approximation heuristics and benchmarkings for the minla problem, 1998.

[78] J. Petit. Combining spectral sequencing and parallel simulated annealing for the minla problem. *Parallel Processing Letters*, 13(1):77–91, 2003.

[79] J. Petit. Experiments on the minimum linear arrangement problem. *ACM Journal of Experimental Algorithmics, 8*, 2003.

[80] E. Pinana, I. Plana, V. Campos, and R. Martí. GRASP and path relinking for the matrix bandwidth minimization. *Eur. J. Oper. Res.*, 153(1):200–210, 2004.

[81] T. Poranen. A genetic hillclimbing algorithm for the optimal linear arrangement problem, 2002, unpublished.

[82] A. Pothen, H. Simon, and L. Wang. Spectral nested dissection. Technical Report CS-92-01, 1992.

[83] A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990.

[84] R. Pozo, J. J. Dongarra, and D. W. Walker. Lapack++: a design overview of object-oriented extensions for high performance linear algebra. In *Supercomputing '93: Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, pages 162–171, New York, NY, USA, 1993. ACM Press.

[85] Sherief Reda and Amit Chowdhary. Effective linear programming based placement methods. In *ISPD*, pages 186–191, 2006.

[86] D. Ron. *Ph.D. Thesis. Development of fast numerical solvers for problems in optimization and statistical mechanics.* PhD thesis, The Weizmann Institute of Science, 1990.

[87] D. Ron, S. Wishko-Stern, and A. Brandt. An algebraic multigrid based algorithm for bisectioning general graphs. Technical Report MCS05-01, Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, 2005.

[88] Jarrod A. Roy, David A. Papa, Aaron N. Ng, and Igor L. Markov. Satisfying whitespace requirements in top-down placement. In *ISPD*, pages 206–208, 2006.

[89] J. Ruge and K. Stüben. *Algebraic Multigrid*, pages 73–130. SIAM, 1987.

[90] I. Safro. Homepage of our projects. *http://www.wisdom.weizmann.ac.il/˜safro.*

[91] I. Safro. The minimum linear arrangement problem on proper interval graphs. *arXiv*, cs.DM/0608008, 2002.

[92] I. Safro, D. Ron, and A. Brandt. Graph minimum linear arrangement by multilevel weighted edge contractions. *Journal of Algorithms*, 60(1):24–41, 2006.

[93] I. Safro, D. Ron, and A. Brandt. Multilevel algorithm for the minimum 2-sum problem. *Journal of Graph Algorithms and Applications*, 10(2):237–258, 2006.

[94] I. Safro, D. Ron, and A. Brandt. Multilevel algorithms for linear ordering problems. *submitted to "Journal of experimental algorithmics"*, 2007.

[95] Kirk Schloegel, George Karypis, and Vipin Kumar. Parallel multilevel algorithms for multi-constraint graph partitioning. In *Euro-Par*, pages 296–310, 2000.

[96] Farhad Shahrokhi, Ondrej Sýkora, László A. Székely, and Imrich Vrto. On bipartite drawings and the linear arrangement problem. *SIAM Journal on Computing*, 30(6):1773–1789, 2001.

[97] E. Sharon, A. Brandt, and R. Basri. Fast multiscale image segmentation. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, pages 70–77, 2000.

[98] Daniel A. Spielman and Shang-Hua Teng. Spectral partitioning works: Planar graphs and finite element meshes. In *IEEE Symposium on Foundations of Computer Science*, pages 96–105, 1996.

[99] K. Stüben. *An introduction to algebraic multigrid*, pages 413–532. Academic Press, 2001.

[100] K. Stüben. A review of algebraic multigrid. *J. Comput. Appl. Math.*, 128(1-2):281–309, 2001.

[101] Wern-Jieh Sun and Carl Sechen. Efficient and effective placement for very large circuits. In *ICCAD '93: Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, pages 170–177, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[102] Taraneh Taghavi, Xiaojian Yang, Bo-Kyung Choi, Maogang Wang, and Majid Sarrafzadeh. Dragon2006: blockage-aware congestion-controlling mixed-size placer. In *ISPD*, pages 209–211, 2006.

[103] Ulrich Trottenberg and Anton Schuller. *Multigrid.* Academic Press, Inc., Orlando, FL, USA, 2001.

[104] C. Walshaw. A multilevel approach to the travelling salesman problem. Tech. Rep. 00/IM/63, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, August 2000.

[105] C. Walshaw. A Multilevel Approach to the Graph Colouring Problem. Tech. Rep. 01/IM/69, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, May 2001.

[106] C. Walshaw. Multilevel refinement for combinatorial optimisation problems. *Annals Oper. Res.*, 131:325–372, 2004.

[107] Irad Yavneh. Why multigrid methods are so efficient. *Computing in Science and Engineering*, 8(6):12–22, 2006.