Graph Minimum Linear Arrangement by Multilevel Weighted Edge Contractions

Ilya Safro *

Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, POB 26, Rehovot 76100, Israel

Dorit Ron

Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, POB 26, Rehovot 76100, Israel

Achi Brandt

Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, POB 26, Rehovot 76100, Israel

Abstract

The Minimum Linear Arrangement problem is widely used and studied in many practical and theoretical applications. In this paper we present a linear-time algorithm for the problem inspired by the Algebraic Multigrid approach which is based on weighted edge contraction rather than simple contraction. Our results turned out to be better than every known result in almost all cases, while the short running time of the algorithm enabled experiments with very large graphs.

Key words: Minimum Linear Arrangement, Combinatorial Optimization, Multilevel Computations, Graphs, Weighted Edge Contractions, Weighted Aggregation, Coarsening, Interpolation, Relaxation, Simmulated Annealing

* Corresponding author.
Email addresses: ilya.safro@weizmann.ac.il (Ilya Safro), dorit.ron@weizmann.ac.il (Dorit Ron).

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 [14].

Originally the MinLA problem was formulated in 1964 by Harper in [15]. 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 [10]. Additionally, the MinLA appears in several biological applications, graph drawing, reordering of large sparse matrices and other fields (see [11,20,16,28]).

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 [17], Optimally Oriented Decomposition Tree [1], Multilevel based [19], Simulated Annealing [22,23], Genetic Hillclimbing [24] and some of their combinations. All these heuristics were tested on the test suite that was compounded by Petit [22], 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) [2–4,9,26,30,31]. 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.) [8,6]. 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 [29]. 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) [18]. 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 [1,13,22,23,11,24,19], while its speed (despite its unoptimized current state) is much better than the fastest algorithm [19].

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

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 1). The original form of the problem is the special case where all the lengths are equal.

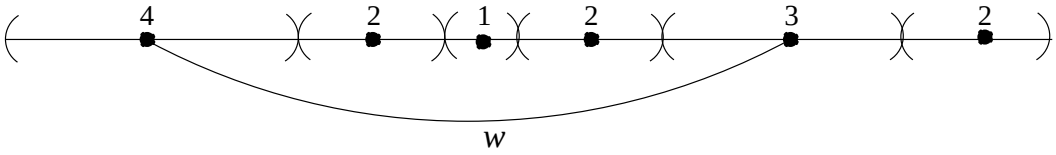

Fig. 1. An example for the generalized form of the problem. Each node captures a segment on the real axis. Its length is written above it. If, for instance, the edge between the first node to the fifth one has weight w, then its contribution to the cost of the linear arrangement is $w \cdot 8.5$.

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 $|V| + |E|$.

3 The algorithm

In the multilevel framework a hierarchy of decreasing size graphs : $G_0, G_1, ..., G_k$ is constructed, see Figure 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*.

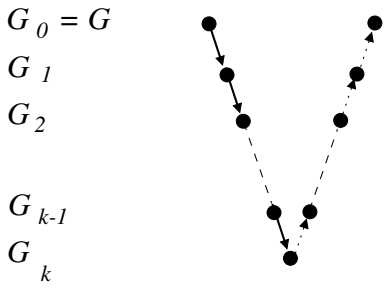

Fig. 2. The Scheme of a V-cycle. Solid arrows stand for coarsening, dotted for uncoarsening.

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.

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 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

$$A_{ij} = \begin{cases} -w_{ij} & \text{for } ij \in E,\ i \neq j \\ 0 & \text{for } ij \notin E,\ i \neq j \\ \sum_{k \neq i} w_{ik} & \text{for } i = j \end{cases} . \tag{1}$$

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

$$w_S(ij) = \frac{w_{ij}}{\sum\limits_{k \in S} w_{ik}} . \tag{2}$$

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

$$\vartheta_i = v_i + \sum_{j \in F} v_j \cdot min(1, \frac{d_j}{\rho_j} \cdot w_V(ji)) \quad , \tag{3}$$

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 ϑ_i for each $i \in F$
$C \leftarrow$ nodes with $\vartheta > \eta \cdot (average\ of\ \vartheta)$

$F \leftarrow V \setminus C$
Recalculate ϑ_i for each $i \in F$
Sort F in descending order of ϑ
Go through all $i \in F$ in decreasing order of ϑ
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

$$P_{ij} \ = \ \begin{cases} w_{N_i}(ij) & \text{for } i \in F,\ j \in N_i \\ 1 & \text{for } i \in C,\ j = i \\ 0 & \text{otherwise} \quad . \end{cases} \tag{4}$$

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 $A^c = P^T A P$. Here, however, we follow the approximated scheme used in [29], 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.

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.

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.

3.3.1 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

$$\{y \ : \quad |\sum_{y_j<y,\ j\in V'} w_{ij} - \sum_{y_j>y,\ j\in V'} w_{ij}| \quad \text{is minimal}\}, \tag{5}$$

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 [19]. 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

$$x_i \ = \ \frac{v_i}{2} \ + \ \sum_{y_k<y_i} v_k \qquad , \tag{6}$$

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

3.3.2 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 (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 [7], only improves the positions of the F-nodes according to the minimization criterion (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 (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%.

3.3.3 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 (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 3.

3.3.4 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) [18], 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) [5] 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 γ
  Apply at most k_3 sweeps strict minimization within distance k_4 on $\mathcal{F}$
  Update $BSF \leftarrow LCC(BSF, \text{current arrangement of } \mathcal{F})$
Return BSF

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 [8]). 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 :

$$\text{minimize} \sum_{ij\in E} \frac{w_{ij}}{|\tilde{x}_i - \tilde{x}_j|^\alpha}(x_i - x_j)^2 \text{ , with } \alpha = 1. \tag{7}$$

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 (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.

4 Results and Related Works

We have implemented and tested the algorithm using standard C++ and LEDA libraries [21] 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 [22]. The test suite graphs are given in Table 1. In Table 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 [11,13,22,23]. 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 [19]. 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. [1] 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" [24].

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) 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 5) only for the much larger graphs given in Table 4 and discussed below.

Table 1
Petit's benchmark [22].

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

Table 2
Comparative table of results for the test suite of Table 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**

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 [13,22], uniform random graphs are actually unworthy for the purpose of evaluating heuristic algorithms (see analytical explanation in [12]).

Table 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

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 1, namely, the hypercube graph ("hc10"), the lattice graph ("mesh33x33") and the binary tree ("bintree10") [11]. In addition, we have added four larger lattices ("mesh100x100", "mesh200x200", "mesh500x500", "mesh1000x1000") and three proper interval graphs which also have known minima [27]. The results are shown in Table 3. We have employed three "extended" V-cycles enhanced by the segment minimization (see Section 3.3.3). 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 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 [19], 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 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 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 (~ 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%.

Table 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

(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.

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

Table 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

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 [25]. 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:

$$(SM)_{i+k}^\varphi - (SM)_i^\varphi \;=\; (SM)_{j+k}^\psi - (SM)_j^\psi \;. \tag{8}$$

The opposite is, however, not always true : (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 (8) holds while the SCSP is not a CSP is extremely low.

Equation (8) can also be written as

$$(SM)_i^{\varphi} - (SM)_j^{\psi} \;=\; (SM)_{i+k}^{\varphi} - (SM)_{j+k}^{\psi} \;. \tag{9}$$

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"

$$A_l = (SM)_{\pi_1(l)}^{\varphi} - (SM)_{\pi_2(l)}^{\psi}, \tag{10}$$

Equation (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 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 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.

References

[1] 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

Table 6
The parameters used for the "quick", "extended" and "super" V-cycles. (* used only to obtain the results of Table 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	$+\log(\sqrt{R})$

Applications, vol. 5, no. 4, pp. 1-27, 2001.

[2] A. Brandt, S. McCormick, and J. Rudge, *Algebraic multigrid (AMG) for automatic multigrid solution with application to geodetic computations.*, Institute for Computational Studies, POB 1852, Fort Collins, Colorado, 1982.

[3] A. Brandt, S. McCormick, and J. Rudge, *Algebraic multigrid (AMG) for sparse matrix equations.*, In Sparsity and its Applications (Evans, D.J., ed.), Cambridge University Press, Cambridge, 1984, pp. 257-284.

[4] A. Brandt, *Algebraic Multigrid Theory : The symmetric case*, Appl. Math. Comput., 19:23-56, 1986.

[5] A. Brandt, D. Ron and D. Amit, *Multi-level approaches to discrete-state and stochastic problems*, Multigrid Methods, II (Hackbush, W. and Trottenberg, U., eds.), Springer-Verlag, 1986, pp. 66-99.

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

[7] A. Brandt, *General highly accurate algebraic coarsening*, Gauss Center Report WI/GC-13, May 1999, Electronic Trans. Num. Anal. 10 (2000) 1-20.

[8] A. Brandt and D. Ron, *Multigrid solvers and multilevel optimization strategies*, in "Multilevel Optimization and VLSICAD" edited by J. Cong and J. R. Shinnerl, Kluwer, 2002.

[9] W.L. Briggs, V.E. Henson and S.F. McCormick, *A Multigrid Tutorial*, 2nd Edition, SIAM, 2000.

[10] C.K. Cheng, *Linear Placement Algorithms and Applications to VLSI Design*, Networks, vol. 17, pp. 439-464, Winter 1987.

[11] J. Díaz, J. Petit, and M. Serna. *A survey on graph layout problems.* ACM Computing Surveys, Volume 34, Issue 3:313-356, 2002.

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

[13] J. Díaz, M. D. Penrose, J. Petit, M. J. Serna, *Approximating Layout Problems on Random Geometric Graphs*, J. Algorithms 39(1): 78-116, 2001.

[14] M.R. Garey, D.S. Johnson, and L. Stockmeyer, *Some Simplified NP-complete graph problems*, Theoretical Computer SCience, 1:237-267, 1976.

[15] L.H. Harper, *Optimal assignments of numbers to vertices*, Journal of SIAM, 12(1):131-135, 1964.

[16] S. Horton, *The optimal linear arrangement problem: algorithms and approximation*, PhD Thesis, Georgia Institute of Technology, 1997.

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

[18] S. Kirkpatrick, *Models of disordered systems*, Lecture Notes in Physics149 (C. Castellani et al., eds.), Springer-Verlag, Berlin.

[19] Y. Koren and D. Harel, *A Multi-Scale Algorithm for the Linear Arrangement Problem*, Proceedings of 28th Inter. Workshop on Graph-Theoretic Concepts in Computer Science (WG'02), Lecture Notes in Computer Science, Vol. 2573, Springer Verlag, pp. 293–306, 2002.

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

[21] K. Mehlhorn and S. Naher, *LEDA - A platform for combinatorial and geometric computing*, Cambridge University Press, 1999.

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

[23] J. Petit, *Combining Spectral Sequencing and Parallel Simulated Annealing for the MinLA Problem*, Parallel Processing Letters, 13(1):77-91, 2003.

[24] T. Poranen, *A genetic hillclimbing algorithm for the optimal linear arrangement problem*, http://www.cs.uta.fi/tp/optgen/, 2002.

[25] D. Ron, *Ph.D. Thesis. Development of fast numerical solvers for problems in optimization and statistical mechanics*, The Weizmann Institute of Science, 1989.

[26] J. Ruge, K. Stüben, *Algebraic Multigrid*, In Multigrid Methods (McCormick, S. F., ed.), SIAM, Philadelfia, 1987, pp. 73-130.

[27] I. Safro, *M.Sc. Thesis. The Minimum Linear Arrangement Problem*, The Weizmann Institute of Science, 2002. http://www.wisdom.weizmann.ac.il/~safro/thesis.ps

[28] Farhad Shahrokhi, Ondrej Sykora, Laszlo A. Szekly, Imrich Vrto, *On Bipartite Drawings and the Linear Arrangement Problem*, Workshop on Algorithms and Data Structures, 1997.

[29] E. Sharon, A. Brandt, R. Basri, *Fast Multiscale Image Segmentation*, Proceedings IEEE Conference on Computer Vision and Pattern Proceedings IEEE Conference on Computer Vision and Pattern Recognition, I:70-77, South Carolina, 2000.

[30] K. Stüben, *An introduction to algebraic multigrid*, Appendix in: *Multigrid* (Trottenberg, U., Oosterlee, C.W. and Schüller, A., eds.), Academic Press, 2001, pp. 413-532.

[31] K. Stüben, *A review of algebraic multigrid*, J. Comput. Appl. Math. 128 (2001) 281-309.