Weizmann Institute of Science

The Minimum Linear Arrangement problem

Safro Ilya
M.Sc. Thesis under supervision of
Prof. A. Brandt and Prof U. Feige

March 20, 2003

Contents

1 Introduction

The Minimum Linear Arrangement (MinLA) problem belongs to the family of graph layout problems. Besides MinLA, this family contains 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. There is a number of surveys describing the progress and current status of these problems in general, their approximation results and efficiently solvable problems on some graph classes : [15, 34, 28].

1.1 Problem definition

Given a graph $G = (V,\ E)$, $|V| = n$ and $|E| = m$. For all $e \in E$ define weights $w(e) \in \mathbb{R}^+$ and let π be some linear order of V, i.e., π is a bijection which assigns for every $v_i \in V$ some $\pi(v_i) \in [1..n]$. For any two vertices $u, v \in G$ we define a distance between them with given linear order π as

$$dist_\pi(uv) \ = \ \begin{cases} |\pi(u) - \pi(v)|, & uv \in E \\ 0, & otherwise \end{cases}$$

Problem : Given a graph G and weight function on edges w find a linear order π such that the function $C_{G,\pi}$

$$C_{G,\pi} = \sum_{uv \in E} dist_\pi(uv) \cdot w(uv)$$

is minimal. The optimal linear order will be denoted by π^* and its cost C^*_G.

Simplified version of the problem : For any $e \in E$, $w(e) = 1$.

My thesis will concentrate on solving the problem above, which is connected to several areas in Mathematics and Computer Science.

Originally the Minimum Linear Arrangement problem was formulated in 1964 by Harper in [25]. His aim was to design error-correcting codes with minimal average absolute errors on certain classes of graphs. The problem has also received some alternative names, as the Optimal Linear Ordering and Edge Sum problem.

This problem may be motivated as a model to design VLSI layouts. Given a set of modules, the VLSI layout problem consists of placing them on a board in a nonoverlapping manner and wiring together the terminals on the different modules according to a given wiring specification in such a way that the wires won't interfere among themselves. There are two stages in VLSI layout: placement and routing. The placement problem consists of placing the modules on a board; the routing problem consists of wiring together the terminals on different modules to be connected. Several approaches to solve the placement phase use the Minimum Linear Arrangement problem in order to minimize the total wire length [13].

The Minimum Linear Arrangement problem is also connected to graph drawing: A bipartite drawing is a graph representation where the nodes of a bipartite graph are placed in two parallel lines and the edges are drawn with straight lines between them. The bipartite crossing number of a bipartite graph is the minimal number of edge crossings over all bipartite drawings. The task of finding this number can be reduced to the minimum linear arrangement problem [42].

Additionally, in the area of numerical analysis the Minimum Linear Arrangement problem is connected to many applications that deal with reordering of large sparse symmetric matrices in such a way that their non-zero entries lie as close as possible to the diagonal.

1.2 Some properties of the MinLA problem

The naive solution of this problem is based on checking all $n!$ possible arrangements of the vertices. However, the size of the solution space to be searched can be reduced from $n!$ to $O(2^n)$ using dynamic programming. Given an order π of a graph G. Divide this order at some point into left and right segments, L and R respectively :

$$\pi \ : \ \overbrace{v_1,\ v_2,\ .....,\ v_l}^{\text{L}},\ \overbrace{v_{l+1},\ .....,\ v_n}^{\text{R}}$$

Denote by $C_{L,\pi,G}$ the left partial cost of arrangement π (the right partial cost of the arrangement is defined in a similar manner):

$$C_{L,G,\pi} \ = \ \sum_{u,v \in L} w(uv)|\pi(u) - \pi(v)| \ + \ \sum_{u \in L, v \in R} w(uv)(l - \pi(u))$$

and then

$$C_{G,\pi} \ = \ C_{L,G,\pi} + C_{R,G,\pi}.$$

So, given a division D of π, a minimum cost of all linear orders, which preserves D can be obtained by minimization of left and right partial costs independently on one another. Thus we can define a relation which will be the heart of dynamic programming for the MinLA problem. Denote by $C_{L \setminus v,G,\pi} \leftsquigarrow v$ the cost of partial left order L when the last vertex in the order must be v, and denote

$$min(C_{L,G}) \ = \ min_{\pi}(C_{L,G,\pi}), \text{ where } \pi \text{ must preserve the contents of } L$$

Then

$$min(C_{L,G}) \ = \ \begin{cases} 0, & if \ |L| \leq 1 \\ min_{v \in L}\{min_{\pi}(C_{L \setminus v,G,\pi} \leftsquigarrow v)\}, & otherwise \end{cases}$$

The last definition is taken over all permutations π that preserves the contents of $L \setminus v$. Thus $min(C_{V,G})$ produces the solution for the entire graph. The algorithm may be implemented using $n \times 2^n$ table, where the rows will represent the last vertex in the partial left order and each column will point to the respective subset of the vertices. The solution will be obtained in the last column, in the row with the minimum entry.

Let us look at the expected cost of a randomly uniformly generated linear arrangement.

Lemma 1.1 *The expected cost of a randomly uniformly generated order π on G is*

$$E[C_{G,\pi}] \approx \frac{nm}{3}.$$

Proof:

$$E[C_{G,\pi}] = E[\sum_{uv \in E} dist(uv)] = \sum_{uv \in E} E[dist(uv)] = m \cdot E[dist(uv)]$$

Now we will calculate the expected cost of some edge in graph. If some edge has in some order the cost i then there are $n - i$ possible positions for such edge in the set of orders. Then the total number of possible costs will be $1 + 2 + ... + (n-1) = \frac{n(n-1)}{2}$. Then the expectation of the cost of one edge will be :

$$E[dist(uv)] = \sum_{i=1}^{n-1} i \cdot \frac{n-i}{\frac{n(n-1)}{2}} = \frac{2}{n(n-1)} \cdot \sum_{i=1}^{n-1} i(n-i) = \frac{2}{n(n-1)} \cdot \left(n \sum_{i=1}^{n-1} i - \sum_{i=1}^{n-1} i^2\right) \approx \frac{n}{3}$$

Then

$$E[C_{G,\pi}] \approx \frac{nm}{3}.$$

1.3 Previous work

The Minimum Linear Arrangement problem on general graphs is NP-hard and its decision version is NP-complete [22]. The problem remains NP-complete even for bipartite graphs [18].

However, the general NP-completeness does not exclude the existence of efficient algorithms for some special classes of graphs. In spite of the NP-completeness for general bipartite graphs, the problem was solved efficiently on complete bipartite graphs by Juvan and Mohar in [30]. The problem was efficiently solved as well for hypercubes [25] by Harper. There are some algorithms for solving the problem on trees. The algorithm with $O(n^3)$ complexity was developed by Goldberg and Klipker in [23], then Shiloach in [43] with $O(n^{2.2})$ complexity, and by Chung in [14] with $O(n^{\frac{log3}{log2}})$ complexity for binary trees. The Minimum Linear Arrangement problem on rectangular grids was solved by Muradyan and Piliposjan in [36] and then by Fishburn, Tetali and Winkler in [21]. There are some other classes of graphs which have efficient algorithms for this problem, see [3] and [4].

Several approaches were developed for calculating lower and upper bounds. Juvan and Mohar in [30] developed a lower bound using spectral techniques. Suppose M is the Laplacian matrix of the graph defined as following :

$$M(u,v) = \begin{cases} -1 & uv \in E \\ 0 & uv \notin E \\ d(u) & u = v \end{cases}$$

and $\mu_1 \leq \mu_2 ... \leq \mu_k$ its eigenvalues sorted in increasing order. Then

$$\mu_2 \frac{n^2-1}{6} \leq C_G^* \leq \mu_k \frac{n^2-1}{6}.$$

Another lower bound presented by Horton in [29] is based on cut sets. Denote by $c(A, \bar{A})$ (where $\bar{A} = V \setminus A$) the number of edges in G from vertices in A to $\bar{A}$ then we define

$$\sigma_i = min_{A \subset V}\{c(A, \bar{A}) \mid |A| = i\}$$

i.e. the size of a smallest cut in G induced by i vertices. Then

$$C_G^* \geq \sum_{1 \leq i \leq |V|-1} \sigma_i .$$

Using this technique Horton presents the lower bound for discrete torus, recursive graphs and some other particular classes of graphs. Horton also formulates another problem that is similar to Minimum Linear Arrangement problem but has additional constraint : some vertices must be placed at predefined coordinates. He proves that even for a set of disjoint paths the problem is NP-complete.

Rao and Richa described in [40] an approximation algorithm for Minimum Linear Arrangement problem . An α-approximation algorithm is an algorithm that finds a solution to the problem whose cost is at most α times the cost of an optimal solution to the problem. They presented a polynomial time $O(log\ n)$-approximation algorithm on a graph with n nodes. The algorithm is based on spreading metrics usage.

2 The Multilevel algorithm for MinLA

In this chapter we will introduce an algorithm which finds approximate solution to the Minimum Linear Arrangement problem . Besides of sorting, the algorithm works in time that is linear in the number of edges. However, sorting procedures were used in the algorithm only for convenience of implementation. These procedures can be replaced with linear equivalents without quality loss. Later we will present a set of results of the runnings of this algorithm on a test suite from [15], graphs with known optimal solution and several large graphs. The algorithm is based on Algebraic Multigrid strategy which will be overviewed in Section 2.1. Section 2.3 refers to global description of the algorithm. Then in Section 2.4 we will describe the notations and terms we use during detailed description of the algorithm and after that we introduce in more depth all the algorithm's parts.

2.1 Overview of the Algebraic Multigrid (AMG) strategy

2.1.1 Multiscale method

The **multiscale method** is a class of algorithmic techniques for solving 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 envolving 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 ideas of multiscale methods.

The key idea of the multiscale techniques can be presented in the following algorithmic structure. In 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 find the solution Π_K for P_K first and then, level by level, construct Π_{i-1} from Π_i for each i. There are two key steps of the multiscale method :

1. **coarsening :** which defines the hierarchical structure $(P_1, ..., P_K)$ for a given problem $P_0 = P$;

2. **interpolation :** which produces an initial solution of P_{i-1} from the solution Π_i of P_i and then by some optimization process (will be called here *relaxation*) constructs final solution Π_{i-1}.

2.1.2 Multigrid solvers

Multigrid solvers were first developed in the form of multigrid algorithms for solving linear boundary-value partial differential equations (PDE) discretized on regular grids. They were introduced in a restricted forms by Southwell and Fedorenko in [44] and [19] respectively, and in their general modern form by Brandt in [6]. Detailed explanation of multigrid techniques was presented by Brandt in [12]. Suppose that we want to solve the PDE

$$Lu = f$$

on a physical domain Ω. The PDE is approximated (discretized) by a linear system when the physical domain Ω is approximated (or discretized) by a mesh. Thus, every mesh over Ω defines a linear system. Multigrid solves a boundary value problem over the domain Ω by first constructing a sequence of meshes $M_0, ..., M_K$, where $M_0 = M$ is the finest mesh that discretizes Ω. For each i in the range $1 \leq i \leq K$, the mesh M_i is a geometric coarsening of M_{i-1}. At each step i of the coarsening process we obtain the system of equations

$$L_i u_i = f_i \tag{1}$$

that we need to solve, where L_i is some linear operator over $\mathbb{R}^{n_i} \times \mathbb{R}^{n_i}$ and f_i is a vector in $\mathbb{R}^{n_i}$ associated with a grid M_i. The main question of multigrid is the transformation of a partial solution from mesh M_i to the mesh M_{i-1} and vice versa. Informally, these hierarchical methods solve a PDE on Ω by first obtaining an initial vector solution either M_0 or for M_K, and then improving the quality of the vector transforming it hierarchically up and down the hierarchy while applying some efficient methods at each level.

2.1.3 Algebraic multigrid

Algebraic multigrid (AMG) algorithms are solvers of linear systems of equations which are based on multigrid principles. In contrast to the (first developed) geometrically based multigrid, AMG does not require a given problem to be defined on a grid but rather operates directly on the algebraic equations (1). Introduced in [7], the advantage of AMG is that the construction of a problem-dependent hierarchy, including the coarsening process itself, the transfer operators as well as the coarse-grid operators – in the AMG algorithm, is based solely on algebraic information contained in the given system of equations. In classical AMG a set of coarse variables C is chosen so that each fine variable is "strongly coupled" *to one or more* variables in C.

For example, in our problem we reformulate the objective function over real values (instead of integer values) and choose the coarse set of variables C as a subset of the fine set of variables. Each of the non-chosen to C variables is associated with some of the C variables. Then, when the step of interpolation comes we keep the coarse variables at their places and in a number of sweeps define the locations of the fine variables.

Let us concentrate now on the relation between graph minimization problems to AMG solvers. For a more detailed study of this issue see [11]. The AMG coarsening will be interpreted as a process of *weighted aggregation* of the graph nodes to define the nodes of the

next coarser graph (called aggregates). An alternative strategy is called *strict aggregation*, in which the nodes of the fine graph are blocked in small disjoint subsets. Each such subset defines a node in the coarse graph. Two nodes i and j usually will be joined together only if their coupling (the weight of edge between them) is strong. In weighted aggregation, each node can be divided into fractions and different fractions belong to different aggregates. Weighted aggregation is important when we want to express the likelihood of nodes to belong together, i.e., to give to every fine vertex possibility to relate to more than one aggregate. Strict aggregation may run into a conflict between a local blocking decision and a larger-scale picture. Before we started developing the weighted aggregation algorithm, we have implemented the more natural strict aggregation. The best results were obtained in all our tests using weighted aggregation.

The results of running our algorithm on a set of graphs from [15] are at least comparable to the results obtained by other methods.

2.2 Generalized Problem Definition

In order to recursively apply the AMG strategy we first generalize the problem to the form it will acquire at coarse levels. The given graph $G = (V, E)$ is assumed to have two real functions :

$$\begin{aligned} vol &: V \longrightarrow \mathbb{R}^+ \\ w &: E \longrightarrow \mathbb{R}^+ \end{aligned}$$

The first function attaches volume value to every node while the second function defines a weight (or strength) to every edge. At the beginning, all vertices may have equal volume (for example 1, if we solve the simplified version). Our goal is to arrange the vertices on a real axis in a way that the following objective function will be minimized :

$$\mathcal{E}_\pi(x) \;=\; \sum_{e=(v_i,v_j)\in E} w(e)\cdot |x_{v_i} - x_{v_j}|, \tag{2}$$

where $x = (x_{v_1}, ..., x_{v_N})$ is the current state vector of coordinates of the vertices on the real axis, defined for a given order π. Each vertex placed on the axis will capture the segment of its volume. These segments can not overlap each other and we define coordinates of vertices on the real axis in the following way :

- if $\pi(u) = 1$ then $x_u = \frac{vol(u)}{2}$
- otherwise if $\pi(u) = k$ and $\pi(v) = k - 1$ then $x_u = x_v + \frac{vol(v)}{2} + \frac{vol(u)}{2}$.

The absolute value in (2) represents the distance between centers of segments, each region representing a node and has its corresponding volume (length). Therefore, in the case of equal-volume vertices (in the original graph), we will obtain $\mathcal{E}_\pi$ equivalent to $C_{G,\pi}$ defined in the introduction.

Sometimes we will use the notation $x_{\pi,v}$ which means x_v in the arrangement π.

2.3 Overview of the multilevel algorithm

Our algorithm is based on a representation of the same minimization problem on a sequence of graphs generated from G, where every next graph is of smaller order (and possibly greater ratio $\frac{|V|}{|E|}$).

In order to construct a smaller graph H from G, we extract from G at each level, a set of representative vertices, i.e., vertices that according to some heuristic function (described bellow) reflect the graph's structure related to our objective function. Each chosen vertex will represent a region of vertices of the current level graph. Then we connect some of these vertices by edges, where each constructed edge will represent the strength of connections between the regions of the two respective vertices. At this stage, having the smaller graph H constructed, we similarly recursively build a pyramid of coarse graphs until we obtain a small enough graph which can be solved exactly.

When we go back in the recursion from the coarse levels to the fine, at each level (except the coarsest) we have an approximately solved coarse problem from which a first related finer arrangement is produced. This initial ordering is then being improved by relaxations, either deterministic or stochastic, with the aim of locating local minima of that particular scale. In the implemented version of this algorithm for local relaxations we used simulated annealing [33] , several minimizations by flips and LCC [8, 41] algorithms. The global scheme of the algorithm is represented at the algorithm scheme 2.1. One full recursive pass from the beginning to the end will be called a *V-Cycle* (because of its V-like structure of recursive calls and exits).

Algorithm 2.1 (MinLA(Input : G))

if $|V_G| = 9$ **then**
  Solve the problem exactly
else
  Find appropriate subset of nodes of G to serve as the vertex set of the coarse graph H
  Construct edges for graph H
  MinLA(H)
  Interpolate from H to G
  Make relaxation on the vertices of G
Return the linear order of G
end.

2.4 Preliminaries and Notations

Denote by G the graph at the current level of the multiscale process and by H the graph at the following coarse level, i.e., we obtain $V(H)$ from $V(G)$ by choosing a subset of $V(G)$ and $E(H)$ by complete redefinition of the set of edges, based on $E(G)$.

For every coarse vertex v in graph G the algorithm has assigned a number called *volume* and denoted by $vol(v)$.

For every coarse edge uv in graph G the algorithm has assigned a number called *weight* and denoted by $w(uv)$. If $uv \notin E(G)$ we define $w(uv) = 0$. This number will express the strength of the connection between the two nodes.

Denote by C the set of already chosen coarse vertices. At different moments this set will vary and its content will correspond to the current moment of coarsening. In a similar way we denote by F the current set of fine vertices that were not yet chosen to the coarse vertex set, i.e., $C \cup F = V(G)$ and $C \cap F = \emptyset$.

Denote by $\overline{\pi}$ the reverse of the order π.

The *Future volume* of the fine vertex v, denoted by $fvol(v)$, is defined as

$$fvol(v) \;=\; vol(v) \;+\; \sum_{u \in F} vol(u) \cdot w_F^{'}(uv),$$

where $w_S^{'}(uv)$ is defined over set of the vertices S (v must be in S) as

$$w_S^{'}(uv) \;=\; \frac{w(uv)}{\sum\limits_{y \in S} w(uy)}.$$

For example, if $S = C$ then $w'(uv)$ expresses the likelihood of u to be associated with v at the current moment of coarsening, and the future volume adds to the weight of the node itself suitable parts of the weights of its adjacent F-nodes. We will use $fvol(v)$ as a heuristic measure for the importance of v in the coarsening process.

2.5 Coarsening

The idea of the coarsening (or in our case - weighted aggregation) process is to reduce the large problem consisting of many degrees of freedom to a smaller one. Here this process contains two main parts : finding coarse nodes, and then connecting some of them by new edges.

2.5.1 Coarse nodes

At the beginning we say that $F = V(G),\ C = \emptyset$ and then we sort all nodes by their future volume and automatically add to C those whose future volume exceeds (say twice) the average future volume of the graph. At this moment we have two disjoint sets of vertices F and C. The insertion of additional fine nodes to the coarse set C will be done according to the following algorithm :

1. Sort F according to its future volumes
2. Go through all $v \in F$ in the order of decreasing future volumes and check :
   If
$$\frac{\sum\limits_{u \in C} w(vu)}{\sum\limits_{u \in V_G} w(vu)} \leq T \tag{3}$$

(where T is the parameter THRESHOLD_COARSE_NODES, see chapter 2.11) then move this node from F to C. In other words if $v \in F$ is not connected strongly to C then it becomes a coarse node.

3. If $|C| \leq A|V(G)|$ (where A is the parameter AMOUNT_OF_COARSE_NODES, see chapter 2.11), i.e., the parameter of connectivity between C and F was too small and did not provide enough size of C, then

$$T = T + \beta$$

(where β is small enough) and go to step 1, otherwise completion of C is achieved.

Remark 2.1 *In step 1 the complexity of exact sorting will be more than linear. However, it is enough to have a rough sort instead of exact sort. The complexity of such rough sorting procedure should be at most linear. For example, a rough sort procedure may define several intervals and distribute the vertices of F among these intervals. This remark will be valid for all cases where we have used exact sort for convenience of the implementation and explanations.*

Let us call the vertices passed from G to H *seeds*. The vertices moved from F to C have relatively large future volume. The goal of the future volume value of a vertex v (as a possible seed) is to measure the size of the region v is expected to represent. The seeds are selected iteratively and, at any moment of the coarsening process, every fine vertex has a current number of coarse neighbors. Therefore, when we discuss regions of vertices we cannot state exactly which region should include a fine vertex u. Consequently, we examine the relative part of the volume that u may suggest to its neighbor regions, i.e., to its coarse neighbors. Accordingly, when we resort the remaining F by future volume we make an assumption for every fine u that it is a coarse vertex and raise the question : what is the total volume of parts of u's fine neighbors that may be associated with it ? The largest future volume provides more information about the structure of the region. However, we do not add to C all fine vertices with large future volume but check the strength of connectivity of each fine vertex to already existing neighbors from C. The strength of connectivity is represented by the parameter T (THRESHOLD_COARSE_NODES).

The wrong coarsening (in particular the incorrect choice of parameters A and T) can lead to more than one connected components in the coarse graph. If we will not provide the connectivity – local optimum solution, obtained at the coarse graph, may be far away from the real one. This can happen because the order of the C-vertices is a permutation of the entire regions of the fine graph. Hence if the graph is disconnected then the objective of such a permutation may be falsely interpreted on the entire fine graph. Besides, checking the connectivity of each fine vertex to its coarse neighbors, we introduce a parameter A (AMOUNT_OF_COARSE_NODES) that provides a minimum number of vertices in C (in our tests it was $\frac{1}{3}$ of amount of V). When we do not achieve with some T the needed amount of coarse vertices we simply increase T.

2.5.2 Interpolation weights

Once we have defined the set $V(H)$ of coarse nodes, we have to construct the edges connecting them. Every coarse node represents a small subset of nodes in the fine graph and as we solve the problem for the coarse graph H we obtain the order of regions of vertices on a fine graph (from which a relaxation process will determine a detailed order of the fine nodes and then attempt to improve it). Therefore every edge of H should represent most realistically the structure of the connections between the corresponding regions in G. A full description of the algorithm that constructs coarse edges is described later. In order to present it we should first introduce the notion of *interpolation weight* (iw). We assign iw value to every edge that connects fine and coarse vertices (see below that some "non-significant" weights will be 0). The goal of the interpolation weight value is to define the relation of the fine vertex to its coarse neighbors. Each fine vertex has a set of interpolation weights and each interpolation weight value is related to an edge that connects the fine vertex to its coarse neighbor.

More precisely, if $v \in F$ and $u \in C$ then *preliminary weights (pw)* are first defined by

$$pw(v,u) \;=\; \frac{w(vu)}{\sum\limits_{y \in C} w(vy)}$$

These values are stored to be later used in the relaxation part. To reduce complexity, we limit the real interpolation to be defined only in terms of the most significant and heavy edges. This is made as follows :

- for every fine vertex v let l_v be the list of edges to its coarse neighbors
- sort l_v by pw
- if the length of l_v exceeds the maximum number that we permit (parameter MAXIMUM_CONNECTION_BETWEEN_FINE_AND_ITS_COARSES) then remove from l_v its last members
- go through l_v from head (most heaviest) to tail and accumulate pw until their sum exceeds P (parameter POWER_OF_CONNECTION_BETWEEN_FINE_AND_ITS_COARSES).
- remove the rest from l_v.
- remove from l_v all light edges that are less than 0.01 of the heaviest weight in l_v
- define

$$iw(v,u) \;=\; \begin{cases} \frac{pw(v,u)}{\sum\limits_{y \in l_v} pw(v,y)}, & u \in l_v \\ 0, & otherwise \end{cases}$$

2.5.3 Coarse edge construction

Every coarse edge of H should represent a structure of the connection between two regions of vertices in G. Each such region in G contains one vertex v that is the seed passed to C, and its fine neighbors. Some of these fine neighbors are connected to v stronger than others. The strengths of such connections were calculated and stored as pw values while most representative connection strength values stored as iw. Suppose that we have two regions R_1 and R_2, their seeds (coarse vertices) are C_1, C_2 respectively, and their fine neighbors are f_1^i $(1 \leq i \leq k_1)$ and f_2^j $(1 \leq j \leq k_2)$ respectively. Pay attention to the fact that the intersection of some of neighbor sets f_1^i and f_2^j can be non-empty. Let us uncover what are the possible connections between R_1 and R_2 (see Figure 1). There are four possible types of edges between R_1 and R_2 : (a) edge between C_1 and C_2, (b) edge between f_1^i and C_2, (c) edge between f_2^j and C_1 and (d) edge between f_1^i and f_2^j. The weight of the coarse edge between the two regions should include all possible connections between these regions. We accomplish this with the following heuristics for the construction of a coarse edge e :

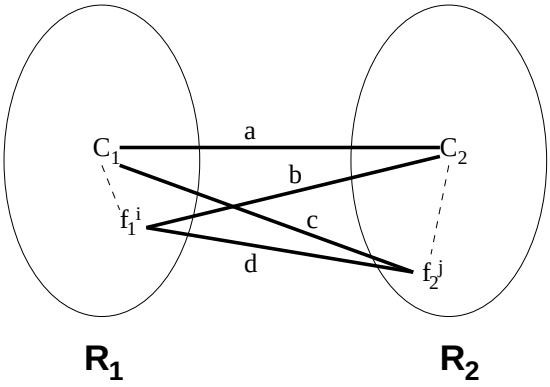

Figure 1: Schematic example of the connections between regions. In calculation of $w(C_1C_2)$, the solid edges will donate their original weights w while the dotted edges – their iw values.

- if there exists an edge between two seeds - we add its weight to the weight of e
- if one of the ends of an edge between the two regions is a fine vertex - we cannot add the entire weight of this edge to e, but check the "strength of connectivity" of the fine end to the coarse node (seed) and add only the corresponding relative part of the weight

The exact Algorithm 2.2 is depicted.

Algorithm 2.2 (input : G)

for every pair $(u, v) \in C^2$
  $cost = 0$
  for every path p from u to v, s.t. $p =< u, y_1, ..., y_k, v >$, $0 < |p| < 4$ and $y_i \in F$
    if $|p| = 1$ **then** $cost = cost + w(uv)$
    if $|p| = 2$ **then**
      $p =< u, y, v >$
      $cost = cost + iw(y, u) \cdot w(yv) + w(uy) \cdot iw(y, v)$
    if $|p| = 3$ **then**
      $p =< u, y_1, y_2, v >$
      $cost = cost + iw(y_1, u) \cdot w(y_1y_2) \cdot iw(y_2, v)$
  if $cost > 0$ add to H new edge (u, v) with weight $cost$
end.

Of course, there exist more longer or complex types of connections between R_1 and R_2, but with every additional type, the complexity grows up while the amount of information passed with the type decreases. There are two reasons why we use iw values and not pw. The main reason is that at the first levels of coarsening process the average degree of the vertices is expected to rise very fast and when we remove the weak edges we significantly improve the running time. Second, we do not want weak connections causing disturbances. For example having less strong but more weak connections can lead to negligibility of strong connections.

Algorithm 2.2 has quadratic time due to its cycle that runs over all pairs $(u, v) \in C^2$. However, it was implemented in another way with linear complexity in a number of edges and here it has the quadratic complexity only for the convenience of understanding the process.

2.5.4 Volumes of vertices in H

Let $vol_G(v)$ be the volume of the vertex coming from the fine level. Every C-vertex v passing to H will get a new volume :

$$vol_H(v) = vol_G(v) + \sum_{u \in F} iw(u, v) vol_G(u) \ ,$$

i.e., equals to the previous volume of the seed with the relative parts of volumes of its neighbors. We assume that all weights of vertices in the initial graph are 1. Note that during the process of coarsening we do not lose the "information" about the vertices and

$$\sum_{v \in H} vol_H(v) \ = \ \sum_{u \in G} vol_G(u) \ .$$

Now the coarse graph H is ready and we can enter the recursion until we obtain a graph for which we can solve the minimum linear order problem exactly. If at the coarsest level we have less than 9 vertices then in inequality (3) T is raised artificially in order to obtain exactly

9 vertices. The solution is obtained in a naive algorithm which scans all $n!$ orders, although, there exists another strategy presented in Chapter 1.2 that works in $O(|E|2^n)$ time. The reason that we used the naive algorithm is the following: during the entire process we try to investigate the results that were gathered not only from the best solution at the coarsest level, but also from other solutions that are not very far from the best one. For detailed explanation of this see chapter 2.7.

2.6 Uncoarsening

2.6.1 Interpolation and real valued relaxation

Once we have solved the problem on H and have its linear order we should return to G and arrange its fine vertices. Starting with the ordering of the coarse vertices according to their arrangement in H, each vertex will capture the segment of its volume in H and will be placed in the middle of this segment. We continue with ordering the fine vertices on the real axis. Every fine vertex v will first get the coordinate equals to a weighted average of its coarse neighbors (that have already been ordered) :

$$x_v \;=\; \sum_{u \in C} x_u \cdot iw(v,u).$$

This placement minimizes the total edge lengths. When all vertices of G are arranged on the real axis and have some real coordinates, we start the relaxation process. It consists of 4 stages.

Stage 1. Compatible Relaxation Scheme. First step will be based on Compatible Relaxation scheme, introduced by Brandt in [10]. This is a fine-level relaxation scheme that keeps coarse-level variables invariant (i.e., it keeps the fine-level configuration always compatible with the same coarse-level configuration). The application of this scheme in terms of the MinLA problem consists of passing several times through all F-vertices and calculating their new coordinates (while keeping the coordinates of the C-vertices fixed):

$$\forall\, v \in F \quad x_v \;=\; \sum_{u \in V} x_u \cdot w_V^{'}(uv).$$

Stage 2. Gauss-Seidel-like relaxation. This relaxation improves the previous by taking into account the fine neighbors of the vertices. Go several times through **all** vertices and calculate their new coordinates :

$$x_v \;=\; \sum_{u \in V} x_u \cdot w_V^{'}(uv)$$

Stage 3. Minimization Relaxation.

- *Minimization of pairs* is a minimization process that gets a graph and its linear order and sequentially tries to flip each pair of consecutive neighbors. If a flip increases the energy then we flip the pairs backwise and move to the next pair of neighbors. This we do RELAX_SWEEPS times, where RELAX_SWEEPS is a parameter.

- *Minimization of triples* is similar to the above process but takes triples of neighbors. For every triple it tries all 6 possible arrangements, takes the minimum energy and continues to the next triple.

- *Local segment minimization.* Described below.

- *Maximum weighted matching minimization.* Described below.

Stage 4. Simulated annealing. This technique is also based on flips of k-distance neighbors. In contrast to all kinds of minimization relaxations, this strategy approves with some probability the flips which increase the cost of the arrangement.

2.6.2 Simulated annealing

This method of stochastic optimization has been introduced into the context of minimization problems by Kirkpatrick in [31]. Simulated annealing is a global optimization method that distinguishes between different local optima. Starting from an initial point, the algorithm takes a step and the function is evaluated. When minimizing a function, any downhill step (which reduces the objective function) is accepted and the process repeats from this new point. An uphill step (which increases the objective function) may be accepted. Thus, it can escape from local optima. As the optimization process proceeds, the length of the steps declines and the algorithm closes in on the global optimum. The algorithm makes very few assumptions regarding the function to be optimized and it is quite robust. In fact, simulated annealing can be used as a local optimizer for difficult functions.

At the heart of the method of simulated annealing is an analogy with thermodynamics, specifically with the way that liquids freeze and crystallize, or metals cool and anneal. At high temperatures, the molecules of a liquid move freely with respect to one another. If the liquid is cooled slowly, thermal mobility is lost. The atoms are often able to line themselves up and form a pure crystal that is completely ordered over a distance up to billions of times the size of an individual atom in all directions. This crystal is the state of minimum energy for this system. The amazing fact is that, for slowly cooled systems, nature is able to find the minimum energy state. In fact, if a liquid metal is cooled quickly or "quenched", it doesn't reach this state but rather ends up in a polycrystalline or amorphous state having higher energy. So the essence of the process is slow cooling, allowing ample time for redistribution of the atoms as they loose mobility. This is the technical definition of annealing, and it is essential for ensuring that a low energy state will be achieved.

In our context the simulated annealing will try with some probability to flip pairs of vertices that are placed at distance k. First we apply it with distance 1 and then with distance 2. Let HC be the parameter for the number of heat-cold steps. The algorithm of SA relaxation is the following :

Algorithm 2.3 (Algorithm SA : k, HC)

for $i = 1$ **to** HC **do**
  Recalculate $TStart$ and $TEnd$
  for $T = TStart$ **to** $T = TEnd$ with step $T = T \cdot \alpha$
    Go through current order of vertices and for every pair $[v_i, v_{i+k}]$ do
      new_linarr = cost of arrangement after possible flip of v_i and v_{i+k}
      δ = new_linarr − $current\_linarr$
      ε = changes in coordinates after the flip (see below)
      Make a flip with probability $min(1,\ e^{\frac{-\delta}{\varepsilon T}})$
  do minimization relaxation
  Update LCC configuration (explained below)
do triples minimization
Restore best seen configuration (or LCC)
end.

The variables $TStart$ and $TEnd$ have values that show us how many flips that increase the energy of the current configuration might be permitted. For SA with distance $k = 1$ $TStart$ will be equal to the value that permits 80% of such flips and $TEnd$ - 10% (i.e., until minimization). For SA with distance $k = 2$ $TStart$ will be equal to the value that permits 20% of such flips and $TEnd$ - 1%.

The number of HC steps at the finest level will be a parameter and at coarser levels it will be decreased as the number of vertices in the graph.

One of the goals of simulated annealing is to push the process to exit from a current configuration. In our content, it can be interpreted as a wish for several arrangement changes. The value ε is inserted for this interpretation. Suppose that before a flip the process was in configuration (order) ϕ and after a flip in ψ then in any moment

$$\varepsilon = \sum_{v \in V} |x_{\phi,v} - x_{\psi,v}|,$$

then the flip which is most influential to the arrangement will be permitted with greater probability. These ε-values lead the process to the behavior which is *individual* for every pair of nodes in the SA process.

2.6.3 Lowest common configuration (LCC)

This method was introduced by Ron in [41] and deals with the improvement of simulated annealing results. Every run of SA will lift us up from a local minimum and bring to another. In order to improve the process of SA relaxation we will use the LCC strategy. The global idea of this strategy consists of storing the best order (call it LCC) and learning how to improve it. Each time that we will obtain a local minimum order we will learn which of its segments may help to achieve the best global order. At the beginning, after the first HC step, LCC will be equal to the obtained order. At every following HC step we update

this LCC. Suppose that after some step of HC we have got an order π. Let us look for subsequences $s =< v_k, ..., v_{k+i} >$ of vertices in π with the following properties :

- there exists in LCC a subsequence s' with the same content but the vertices inside it may be permuted (see [41] for exact algorithm)
- the border vertices in s' and s are the same

Then we will try to substitute s' with s and if it will improve the energy we accept the newly generated subsequence s' in LCC. Finally, if the cost of π is better than the cost of the LCC – we replace LCC by π. The LCC strategy is useful usually when the number of vertices is large enough so we use it only at fine levels whose $|V| \geq LCC\_TRESHOLD$ and when the minimization process is more evolved. The running time of the new LCC calculation after each HC step is linear.

2.6.4 Local segment minimization

The MinLA Problem in redefined context of the real values has the property which was defined in the description of the dynamic programming solution 1.2. Given an arrangement $\pi =< L, M, R >$, where L, M and R are consecutive sub-arrangements of π that grab the segments on the real axis $[1..l]$, $(l...m]$ and $(m..R]$ respectively. Denote by $C_{M,G,\pi}$ the local arrangement cost of subarrangement M :

$$C_{M,G,\pi} = \sum_{u,v \in M} w(uv)|x_u - x_v| + \sum_{u \in L, v \in R} w(uv)(m - l) + \\ + \sum_{u \in L, v \in M} w(uv)(x_{v_i} - l) + \sum_{u \in R, v \in M} w(uv)(m - x_{v_i}). \quad (4)$$

Thus, defining in a similar manner $C_{L,G,\pi}$ and $C_{R,G,\pi}$ we obtain that

$$C_{G,\pi} = C_{L,G,\pi} + C_{M,G,\pi} + C_{R,G,\pi}$$

and minimizing $C_{M,G,\pi}$ by flipping the internal M-vertices may only improve the total cost of π.

The local segment minimization is based on this property of the MinLA. Algorithm 2.4 represents this minimization scheme. It will take the entire segment and make minimization by flipping vertices inside of the segment. The main advantage of this minimization is that it may work with segments larger than triples. Increasing the size of the segments improves the performance of the LCC. Finally it influences the running time by decreasing the number of needed HC steps, if we use it as the minimization after applying the simulated annealing. Unfortunately, our current implementation is not suitable for using this minimization. However, we have tried to use it on several graphs without paying attention to the running time and most of the results were improved.

Algorithm 2.4 (Given order π, S - number of vertices in segment)

for $i = 1$ to $N - S$
  $M = \{m_j\}_{j=1}^{S} \leftarrow$ segment with S consecutive vertices, starting from i
  do sufficient number of repetitions
    $(m_j, m_k) \leftarrow$ pair of vertices in M whose flip makes best improvement of $C_{M,G,\pi}$
    Flip m_j and m_k
    Update the cost of $C_{M,G,\pi}$
end.

2.6.5 Maximum weighted matching minimization

The minimization techniques that were described previously were based on the following property of any linear arrangement π : inside of some π's segment M (as defined by Eq. 4) one can make a flip of two vertices which improves the local cost of M, then the global cost of π will also be improved. Such strategies (due to required short running time, simplicity of implementation, etc.) are not dealing with the question : can a future flip disturb some other flip or make it worse? Trivial example of one problematic situation is depicted on Figure 2. There are two consecutive flips A and B. Flip A improves the objective less than flip B. Suppose that we pass through π from left to right and hence the first met possible flip is A. Then A prevent us from making flip B.

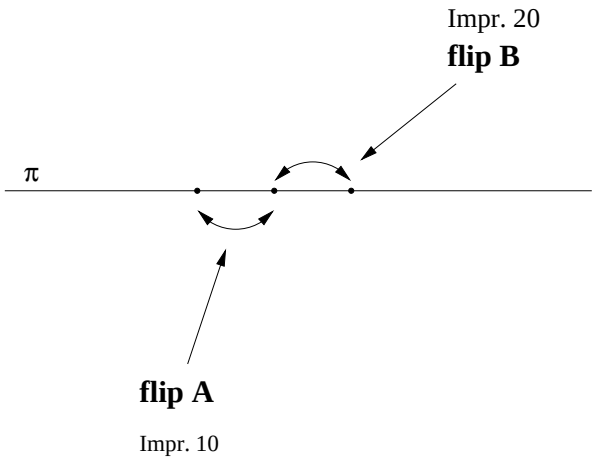

Figure 2: Example of problematic flip

In order to prevent such situations we introduce the strategy which tries to choose the set of flips that do not threaten each other and cause the maximum improvement to the global objective. Suppose that given are the graph $G = (V, E)$ and the linear arrangement π. Let us construct the graph $G' = (V, E')$, where

$$E' = \{uv : u, v \in V \; s.t. \; impr(u \leftrightarrows v) > 0 \; \wedge \; |\pi(u) - \pi(v)| < const\}$$

and $impr(u \leftrightarrows v) > 0$ means that flipping u and v in π improves the global objective. We restrict E' to u and v which are placed in π at some reasonable distance. On graph G' we can formulate the weighted version of the maximum matching problem (WMM). Matching of a graph is a subset of edges such that no two edges share a common vertex. The weighted maximum matching problem asks to maximize the total weight over all chosen edges. The WMM problem has a polynomial time solution and the fastest method known to us has running time $O(|V|(|E| + |V|))$. However the global running time of our algorithm for the MinLA problem is linear in the number of edges and inserting procedures with worse time can damage the global running time. So we need some approximation algorithm with running time at most linear that can give a good feasible solution. There exists an approximation algorithm which finds such a solution with factor 2, i.e. a 2-approximation algorithm as defined at the end of Section 1.3. The algorithm is greedy and at every step takes the heaviest edge it can add to the matching. However the complexity of this algorithm is also too heavy for us. The alternative algorithm that we want to suggest here is also an 2-approximation algorithm but its running time is linear in the number of edges.

The algorithm's strategy uses a concept of alternating paths. Denote for a set of edges S, the value $cost(S)$ – the sum of weights of the edges in S. The algorithm is presented in the following scheme :

Algorithm 2.5 (WMM)

$E_1 = E_2 = \emptyset$
$index = 1$
while G is not empty **do**
  v = randomly chosen vertex
  while $deg(v) > 0$ **do**
    uv = choose an edge uv adjacent to v with maximum weight
    $E_{index} = E_{index} \cup \{uv\}$
    update $cost(E_{index})$
    $G = G - v$
    if $index = 1$ **then** $index = 2$ **else** $index = 1$
    $v = u$
  end
end
if $max(cost(E_1),\ cost(E_2)) = cost(E_1)$ **then**
  return E_1
else
  return E_2
end.

The algorithm has a linear time in the number of edges, since visiting every vertex v it goes through v's adjacent edges in $O(d_v)$ steps. Since every vertex in G must be visited at most once the global running time of the algorithm is $O(|E|)$.

Suppose that we have some maximum weight matching $\mathfrak{M}$ of G. During the execution of the WMM algorithm at every step we scan one vertex and choose the heaviest adjacent edge. After that, all other adjacent edges are removed from the graph. It is easy to see, that according to the pseudocode, for every edge $uv \in \mathfrak{M}$ the following is true : WMM will visit at least at one of its end points u or/and v and will make one of these decisions :

1. add uv to E_{index} if it is the heaviest adjacent edge at that moment,
2. remove uv as it is not the heaviest adjacent edge.

Since every edge must be either chosen or removed, edges from $\mathfrak{M}$ also must be either chosen or removed by WMM. There are no adjacent edges in $\mathfrak{M}$ (by the definition of matching), so we can claim that :

Claim 2.2 *For every edge $uv \in \mathfrak{M}$ the following is true : WMM will choose either uv itself or another edge w.l.o.g. ux (i.e., incident to uv) such that $w(ux) \geq w(uv)$.*

Proof: If uv was chosen by WMM then the claim is true. Suppose that uv was not chosen then it was removed when WMM visited w.l.o.g. vertex u. Then the edge ux was chosen because $w(ux) \geq w(uv)$ and uv is not in $\mathfrak{M}$ due to its adjacency to ux.

So as we see the total cost of $E_1 \cup E_2$ is at least as the total cost of $\mathfrak{M}$. But the cost of one E_i must be greater or equal to half of the total cost of $E_1 \cup E_2$. Thus

$$cost(E_1 \cup E_2) \ \geq \ cost(\mathfrak{M})$$

and

$$max(cost(E_1), cost(E_2)) \ \geq \ \frac{cost(E_1 \cup E_2)}{2} \ \geq \ \frac{cost(\mathfrak{M})}{2}.$$

So the algorithm finds 2-approximation to the weighted maximum matching problem.

In practice this method is much more useful than k-tuples minimization (where k is very small). We have applied this algorithm at every stage when we have used previously only triples minimization. The runnings of the improved algorithm on the test suite immediately show at least three advantages :

- several global results were improved,
- when the execution of the entire algorithm contained more than one solution from the coarsest level, the solutions at the finest level are concentrated very close to the minima,
- any minimization of neighbors (or 3-tuples) is needless after WMM,
- in the minimization used after each HC step of the Simulated Annealing, the benefit of LCC was raised due to an increased numbers of flips that are successful in reducing the cost energy.

2.7 Possible variations of the algorithm

2.7.1 The number of V-Cycles

Before we turn to the explanation of how and why we increase the number of V-cycles, let us discuss the connection of the Minimum Linear Arrangement problem to the Quadratic Arrangement problem. In this context of our generalized problem we define the Laplacian of the graph as a matrix L of size $n \times n$

$$L(u,v) \;=\; \begin{cases} -w_{uv} & uv \in E \\ 0 & uv \notin E \\ \sum_x w_{ux} & u = v \end{cases}$$

The Quadratic Arrangement problem is similar to the MinLA but the objective that must be minimized is

$$f(x) \;=\; \frac{1}{2}\sum_{i,j} w_{ij}(x_i - x_j)^2 \;=\; \frac{1}{2}x^T L x$$

where x_i is a central location of node i. The solution to the Quadratic problem can be used as an approximation to the MinLA problem by increasing the number of V-cycles as follows.

The idea of this technique consists of the fact that when we return from the main recursion to the finest level, an approximate solution to the instance problem is available. Formulated in other words, we detect for every edge a hint whether this edge should be long or short in the optimal solution. So if we decide to depend on this assumption we can run the algorithm a number of times, each time artificially correcting the lengths of edges according to the previous solution. This we can make only after the first full V-cycle when a solution from that V-cycle has already been gathered. The minimization function (2) can be reformulated as

$$\mathcal{E}(x) \;=\; \sum_{(v_i,v_j)\in E} w(v_iv_j)\cdot\frac{(x_{v_i}-x_{v_j})^2}{|x_{v_i}-x_{v_j}|} \;=\; \sum_{(v_iv_j)\in E} \tilde{w}(v_iv_j)\cdot(x_{v_i}-x_{v_j})^2$$

where $\tilde{w}(v_iv_j) = \frac{w(v_iv_j)}{|x_{v_i}-x_{v_j}|}$ are fixed at their values in the available approximate solution. These new values $\tilde{w}(v_iv_j)$ defined for every edge e will be smaller for longer edges. So if we use them as new weights - we artificially permit previous long edges to be light, hence in the new V-cycle they will be easily stretched. In order to prevent very fast permission to the algorithm to stretch the edges, we do this iteratively. Let us define $\tilde{w}(v_iv_j)$ as

$$\tilde{w}(v_iv_j) = \frac{w(v_iv_j)}{|x_{v_i}-x_{v_j}|^\alpha}$$

where α is increased iteratively from 0 to 1 each next V-cycle. These new values are used only in defining the interpolation. We replace the linear objective function by the quadratic one, for which the interpolation weights we use are the proper AMG-type weights (Note however that the AMG solver ignores taking into account the volumes of the nodes).

2.7.2 W-Cycles

The scheduling of the multiscale recursive calls and returns can be defined in a more advanced form called *W-cycle*. The parameter constant γ which is called *cycle index* (in our parameters W_CYCLE_CONST) declares how many times should the process visit the next coarse level before returning to the next finer level. We present the scheme of W-cycle with cycle index 2 in Figure 3. 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.

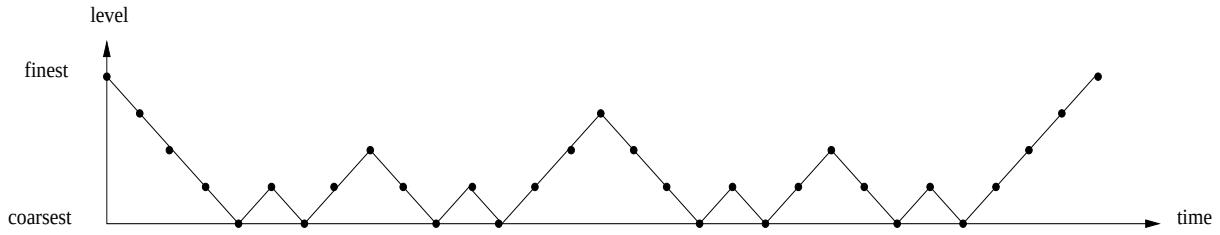

Figure 3: Schematic example of W-Cycle scheduling with 5 coarse levels and cycle index 2.

We have implemented this method. However, there was not any significant improvement of the results on the test suite and since the running time of W-cycle is greater than of V-cycle we can conclude that the combination of W-cycle with our algorithm is not usable.

2.7.3 Increasing the number of Basis solutions

Basis solution is an order obtained at the coarsest level. Due to the structure of the strategy and some elements of randomness used in the algorithm it cannot guarantee that exact optimality will be achieved after the entire interpolation process. In order to increase the set where the solution may be found we extend the space of exact solutions obtained at the coarsest level. The idea is the following : a basis solution which is close to the optimal basis solution can lead after the interpolation process to a local minimum which can be better than the local minimum obtained by the optimal basis solution. Here we introduce three ways to implement this :

Pyramidal structure. We pick a number of solutions at the coarsest level L and move them to the level $L-1$, make the interpolation and relaxation for all of them, and move to the level $L-2$ only part of the obtained orders. This part we can define in different ways : for example, to declare a decreasing constant factor or a factor proportional to the increasing number of vertices on consecutive levels. The advantage of the last rule is in the possibility to take a sufficient large number of solutions at the basis as the time for solving the problem on coarsest levels is negligible. In our tests, we decided that we have interest in seeing, say, 10 possible solutions at the finest level. Therefore, it is possible to calculate automatically the number of solutions we need at every level relatively to the number of vertices. In our implementation the number of solutions taken at the current level was proportional to the number of vertices.

Linear structure. The idea is similar to the pyramidal structure of solutions but in this case every order taken as solution at the basis will be continued until the finest level.

On the other hand, we cannot take a lot of basis solutions due to the running time problem. At the beginning, we used this method because of its simplicity of implementation, but the tests showed that the final results can be improved by taking other (but similar) than best solution at the coarsest level. So the question raised immediately for future work is : how to choose the set of solutions at the coarsest level in order to scan wider space of final results that are close to the optimal.

Wide linear structure. The first attempt to solve the question from the previous method was to choose the basis solutions according to the following criteria :

- the cost of the feasible solution will be as low as possible,
- the positions of the vertices in the arrangement will be not similar to the respective positions in the solution with smallest cost.

Let us introduce one possible strategy to grade the set of feasible solutions according to this criteria. Let π^* be the feasible solution with smallest cost and define the distance between two feasible solutions ϕ and ψ as

$$\Delta(\phi, \psi) \;=\; \sum_{v \in V} |\phi(v) - \psi(v)|.$$

Then for each feasible solution ψ we define the value

$$\frac{C_{G,\psi}}{min(\Delta(\pi^*, \psi), \Delta(\overline{\pi^*}, \psi))}.$$

Now we can sort the solutions according to these values and choose several (according to parameter BASIS_SOLUTIONS_NUMBER) smallest of them.

2.8 Demonstrating the Algorithm

In this chapter the algorithm is illustrated by a specific example. We decided to work with a graph of mesh 9×9, because there exists an efficient algorithm that finds an optimal order for meshes (see [21]) and the comparison between the optimal order and ours will be more comfortable. There exist more than two optimal orders for meshes (an optimal order and its reversal). For example, rotations of the mesh around diagonals, some local flips, etc., applied on the optimal order, conserve the optimal cost.

Let us explain how to understand figures 4,5,6 that demonstrate the algorithm. Each entry in the table is identified with the respective vertex in the graph. Two vertices are connected by an edge iff the respective entries in the table are vertical or horizontal neighbors in the table. There are two types of tables : tables that show the obtained order itself and tables that show the offset of the respective order from the optimal order. If a table shows some order π of the mesh then the numbers written in the table for a vertex v is equal to $\pi(v)$. For example, in π^* (figure 4) the position of the vertex with entry 3 will be equal to 3, while the position of the same vertex in π (figure 5) will be 2.

One possible optimal order π^* of the 9×9 mesh is presented in the left table of figure 4. We begin with the coarsening process. The parameters for it were :

- THRESHOLD_COARSE_NODES=0.25
- AMOUNT_OF_COARSE_NODES=0.33

i.e., initially, each fine vertex that has degree 4 will get at least one coarse neighbor. The process of adding vertices will be continued at least until $|C| \geq \frac{|V|}{3}$. Because of this particular graph's structure, there exist no vertex with large future volume and no vertex will be moved to C automatically. According to the algorithm, we should pass through all fine vertices and check if they have large enough future volume. The vertices visited first are 23, 59, 78, 4 (ID's are according to the left table in figure 4). None of them has neighbors in C (at the moment of checking) and they become coarse. Next visited vertex has ID 58. It has already one coarse neighbor (59), so according to inequality (3)

$$\frac{\sum\limits_{u \in C} w(58\ u)}{\sum\limits_{u \in V_G} w(58\ u)} = \frac{1}{3} > T = 0.25\ .$$

Thus, vertex 58 remains in F. We continue in the same manner to add vertices to C until all fine nodes have at least one coarse neighbor. In our case, there is no necessity to increase T, because after the first pass, we obtain $|C| > 0.33$. Next stage in coarsening will be the construction of coarse edges. As we see, there are no edges between coarse vertices in the fine graph, so all connections will be based on the existence of connections of types (b), (c) and (d) (as described in Section 2.5.3) between two coarse vertices. For example, the weight of the coarse edge between vertices 78 and 71 will be equal to 0.666667 (according to the parameters, fine vertex 76 stores only 3 coarse neighbors and edges to vertices 71 and 78 accumulate only $\frac{2}{3}$ of 76's total edges weight). A connection of type (d) is represented by coarse vertices 48 and 59 which have three possible paths via $< 56, 57 >$, $< 56, 47 >$ and $< 62, 57 >$. The total weight of the new edge will be :

$$w(48\ 59) \ = \ 0.5 + 0.25 + 0.333 \ = \ 1.08333.$$

The real result of the uncoarsening process is not easily comparable to the optimal order (see the right order in Figure 4). In order to demonstrate the correctness of the uncoarsening, we decided to take the finest level coarse vertices and assign to them the optimal order. In other words, we have simulated manually the correct order, gathered from the coarse level, while interpolation weights, volumes of the vertices and weights of the edges were not changed. The left table in Figure 5 shows the order π obtained after interpolation, and the right table of the same figure shows the values of deviation between respective positions of vertices in π and π^*.

Next pair of tables (in figure 6) shows the result of applying the relaxation procedures. The structure of the tables is the same as in figure 5.

2.9 Results and Related Work

We have implemented and tested the algorithm using standard C++ and LEDA libraries [35] on Linux machine with 1700MHz processor. The implementation is non-parallel. Actually

60	61	**63**	66	**69**	72	**77**	80	**81**
58	**59**	62	**65**	68	**71**	76	**78**	79
55	56	57	64	**67**	70	**73**	74	75
46	47	**48**	49	50	**51**	52	53	**54**
37	**38**	39	**40**	41	42	43	**44**	45
28	29	**30**	31	32	**33**	34	35	**36**
7	8	9	12	**15**	18	**25**	26	27
3	**4**	6	**11**	14	**17**	20	**23**	24
1	2	**5**	10	**13**	16	**19**	21	**22**

11	10	15	33	16	36	27	24	22
5	12	32	13	17	35	26	23	21
2	9	6	14	18	34	25	20	19
28	29	30	31	7	8	3	1	4
37	38	39	40	41	42	43	44	45
46	47	48	49	50	51	52	53	54
55	56	57	64	67	70	79	74	75
58	59	62	65	68	71	76	81	78
60	61	63	66	69	72	77	73	80

Figure 4: Left table : An example of 9x9 mesh with minimum linear order (cost=668); the coarse vertices chosen at the finest level are marked with large bold font. Right table : the order obtained with our algorithm (cost=674)

60	61	63	66	67	71	77	79	81
59	50	62	65	70	69	74	80	78
55	56	58	52	64	68	72	73	76
45	49	48	57	53	51	54	75	46
38	39	35	41	42	40	47	44	43
34	30	9	29	37	33	36	27	31
5	10	11	14	17	20	25	28	7
2	8	6	13	16	22	24	23	26
1	3	4	12	15	19	21	18	32

0	0	0	0	2	1	0	1	0
1	9	0	0	2	2	2	2	1
0	0	1	12	3	2	1	1	1
1	2	0	8	3	0	2	22	8
1	1	4	1	1	2	4	0	2
6	1	21	2	5	0	2	8	5
2	2	2	2	2	2	0	2	20
1	4	0	2	2	5	4	0	2
0	1	1	2	2	3	2	3	10

Figure 5: Status of the finest graph after interpolation. The cost of arrangement is 685. Left table shows the arrangement itself. Right table shows the respective offsets from the optimal positions.

59	61	63	66	69	72	77	80	81
58	60	62	65	68	71	74	78	79
55	56	57	64	67	70	73	76	77
46	47	48	49	50	51	52	53	54
37	38	39	40	41	42	43	44	45
28	29	30	31	32	33	34	35	36
5	6	9	12	15	18	24	26	27
2	4	8	11	14	17	20	22	23
1	3	7	10	13	16	19	25	21

1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	2	0	0
0	0	0	0	0	0	0	2	2
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
2	2	0	0	0	0	1	0	0
1	0	2	0	0	0	0	1	1
0	1	2	0	0	0	0	4	1

Figure 6: Status of the finest graph after relaxation (without SA and minimizations). The cost of arrangement is 668 (optimal). Left table shows the arrangement itself. Right table shows the respective offsets from the optimal positions of Figure 4.

the running time of the tests that we present is not the real minimum running time of the algorithm. There are two reasons for this. The main reason is that the algorithm was drastically changed a lot of times during the development, so its implementation is convenient for a short time programming but does not minimize the running time. A lot of implementation parts use classes (like arrays and graphs) from LEDA library. However, there exist other strategies which could obtain much better performance for our algorithm. The second reason is less crucial but also has some influence on the performance: the machines that were used for tests are public. In spite of these reasons we present the running time, which is comparable to the results of other algorithms even in this form (see Table 2).

2.9.1 Petit's et al. test suite

Most of the graphs for our test suite were taken from the set of articles by Petit et al. [15, 16, 37, 38]. This suite represents several graphs families. Table 2 describes the graphs in the set. Petit computed linear arrangements and estimated the lower and upper bounds of these graphs with several algorithms. We do not compare our results in this paper to the estimation of lower and upper bounds. In practice, these estimations are very far from the costs of arrangements obtained by heuristic algorithms. The best results in most cases were obtained by combining spectral sequencing method with simulated annealing. However, in Table 3 (column "Petit et al.") we have extracted the best results from all Petit's et al. articles.

In Table 3 (column "[KH]") we placed the results of Koren and Harel [32] on the same test suite. They developed a linear-time algorithm for the MinLA, based on the combination of spectral methods with the multi-scale paradigm. In principle, the linear order, obtained with the spectral sequencing method, is constructed using the Fiedler vector. If the eigenvalues are sorted by increasing value, the eigenvector of the Laplacian corresponding to the second eigenvalue is known as the Fiedler vector. The Fiedler vector has been used in heuristics for a number of graph manipulations including partitioning, linear arrangement, envelope minimization, etc. Sorting the vertices of the graph accordint to the corresponding values in the Fiedler vector defines the Fiedler linear order. However, Juvan and Mohar proved in [30] that the order obtained by the Fiedler vector solves optimally the minimum-2-sum problem[1] under the condition, that all differences between two successive values in the second eigenvector are equal. Unfortunately, this is not a very likely condition and the differences will more usually vary. The method of using the second eigenvector does not solve our problem (also because ours is the minimum-p-sum problem with $p = 1$, not 2) and it is easy to see the growth of the differences between the optimal costs and costs obtained by the Fiedler vector in mesh graphs (see Table 1). We present the results of Koren after 10 full V-cycles. Comparing these two multiscale methods (ours and Koren's) we can conclude that our method (that was applied to the real $p = 1$ problem) produces better arrangements even when using just 3 V-cycles.

Besides the techniques of Petit we would like to compare our algorithm to the algorithms

[1] **Minimum-p-sum problem :** Given a graph G. Define $\sigma_p(G, \phi) = (\sum_{uv \in E(G)} w(uv)|\phi(u) - \phi(v)|^p)^{1/p}$. Find linear order ϕ that minimizes $\sigma_p(G, \phi)$

Graph	Fiedler cost	Optimal cost
Mesh 4x4	60	60
Mesh 5x5	138	116
Mesh 7x7	392	318
Mesh 9x9	828	668

Table 1: Comparative table of Fiedler arrangement costs with optimal.

of Bar-Yehuda et al. [2] and Poranen [39]. Their best results on Petit's test suite are presented in the respective columns in Table 3. Bar-Yehuda et al. present a polynomial time algorithm for computing an optimal ordering induced by a binary balanced decomposition tree as an initial solution and then apply simulated annealing. In spite of the high quality of Bar-Yehuda's results the running time of the algorithm is $O(n^{2.2})$ which creates a problem in running it on very large inputs. Poranen has presented the stochastic algorithm called "genetic hillclimbing" which in general is comparable to the results of Petit but has few poor deviations.

Table 3 shows the results of several modifications of the algorithm. The modifications are described under the table itself. We have tested these modifications on the entire test suite, so each column with our results refers to one modification. Our algorithm contains a stochastic element, which provides some variability in the results. However, the tests show that a combination of multigrid with LCC technique strongly reduces the variability. The worst result, obtained by a changing the random seed, did not exceed the best cost by more than 1%.

2.9.2 Experimenting with random graphs

Two kinds of random graphs were introduced in Petit's test suite :

- Uniform random graphs $G_{n,p}$ (randomA[1..4]): Graphs with n nodes and probability p to have an edge between any pair of nodes.
- Geometric random graph $G_{n,d}$ (randomG4): Graphs with n nodes located randomly in a unit square. Two nodes will be connected by an edge if the Euclidian distance between them is less than d.

Our algorithm succeeds when the graph has some structure which is different from complete randomness. In the geometric random graphs there is a special structure of cliques which increases the diameter of the graph and makes the remote vertices (in the unit square) disconnected. This is why our algorithm was better on geometric random graph than on uniform random graphs. However we can not say that the algorithm completely fails on the uniform random graphs but in this case the completely stochastic heuristics have a chance to lead us to the better solution. Here we can cite Petit and refer the reader to [16, 37] : "... any algorithm computing a feasible layout, no matter how good or bad, will perform rather well on random graphs, pointing out that evaluating heuristics on uniform random graphs may be unworthy for layout problems".

Graph Name	$\|V\|$	$\|E\|$	Degree	Diameter	Time of 1 V-Cycle with 1 Basis solution (in h.)
randomA1	1000	4974	1/9.95/21	6	0.085
randomA2	1000	24738	28/49.7/72	3	0.311333 *
randomA3	1000	49820	72/99.64/129	4	0.425639 *
randomA4	1000	8177	4/16.35/29	4	0.164722
randomG4	1000	8173	5/16.34/31	23	0.0347222
hc10	1024	5120	10/10/10	10	0.0613889
mesh33x33	1089	2112	2/3.88/4	64	0.0227778
bintree10	1023	1022	1/1.99/3	18	0.00861111
3elt	4720	13722	3/5.81/9	65	0.0744444
airfoil	4253	12289	3/5.78/10	65	0.0636111
whitaker3	9800	28989	3/5.91/8	161	0.173056
c1y	828	1749	2/422/304	10	0.0155556
c2y	980	2102	1/4.29/327	11	0.0158333
c3y	1327	2844	1/4.29/364	13	0.0188889
c4y	1366	2915	1/4.26/309	14	0.0197222
c5y	1202	2557	1/4.25/323	13	0.0175
gd95c	62	144	2/4.65/15	11	0.00555556
gd96a	1076	1676	1/3.06/111	20	0.0155556
gd96b	111	193	2/3.47/47	18	0.00611111
gd96c	65	125	2/3.84/6	10	0.00527778
gd96d	180	228	1/2.53/27	8	0.00583333
crack	10240	30380	3/5.93/9	121	0.206389

Table 2: The description of test suite taken from [15]. (*) – Since the average degrees in these graphs are very high while the diameters are low, the parameters for coarsening were more aggressive (see Section 2.11)

Graph	Petit	KH	BEFN	Poranen	V3B10	V3B10*	B10Pow2*	V1B1	V3B1	V1B30	B50
randomA1	867570	909126	884261	878637	876640	876640	875322	886952	882923	884204	897320
randomA2	6528780	6606174	6576912	6553553	6575088	6575088	6575088	6668009	6601252	6657950	6653687
randomA3	14202700	14457452	14289214		14376620	14445900	14445900	14445858	14445858	14363324	14363964
randomA4	1721670	1765217	1747143	1739317	1744518	1744518	1744518	1768118	1747811		1766377
randomG4	150940	149513	146996	142587	142288	142938	145563	148469	147906	143427	146690
hc10(**)	523776	523776	523776	523776	523776	523776	523776	523810	523784	523776	538112
mesh33x33(**)	32605	31729	33531	32178	32016	32050	32016	32393	32393	32393	32363
bintree10(**)	4069	3950	3762	3899	3700	3704	3778	3804	3804	3772	3808
3elt	363204	373464	363204	383286	358138	362655	361399	369115	366485	363208	364478
airfoil	285231	291794	289217	306005	277590	278354	279630	283172	283172	281453	284775
crack	1491126				1505482	1505482	1510135	1518809	1511268	1511960	1523810
whitaker3	1151064	1205919	1200374	1203349	1152509	1152726	1154464	1160939	1157760	1156395	1157285
c1y	62936	64934	62333	62857	62345	62475	62345	64875	62660	62515	62709
c2y	79673	80148	79571	80327	79019	79174	79039	91373	83088	79291	81287
c3y	124708	127315	127065	125654	123861	124324	124689	125347	125347	124825	125440
c4y	116382	118437	115222	119232	115711	115989	115711	117789	117588	116795	118301
c5y	100264	104076	96956	99389	97101	97352	97521	99311	97575	97628	97565
gd95c	506	509	506	506	506	506	506	513	507	506	506
gd96a	96342	106668	99944	101394	97562	96800	97731	102080	98537	98960	99239
gd96b	1416	1434	1417	1416	1416	1416	1416	1416	1416	1416	1416
gd96c	519	519	519	519	519	519	519	521	519	519	519
gd96d	2393	2393	2409	2391	2397	2391	2406	2432	2402	2405	2419

KH	Koren and Harel
BEFN	Bar-Yehuda, Even, Feldman, and Naor
V3B10	3 V-cycles and 10 solutions at the coarsest level with the linear structure
V1B1	1 V-cycle and 1 solution at the coarsest level with the linear structure
V3B1	3 V-cycles and 1 solution at the coarsest level with the linear structure
V1B30	1 V-cycle and 30 solutions at the coarsest level with the linear structure
B50	1 V-cycle and 50 solutions at the coarsest level with the pyramidal structure
B100	1 V-cycle and 100 solutions at the coarsest level with the pyramidal structure
B10Pow2	1 V-cycle, 10 solutions at the finest level with the pyramidal structure
(*)	The best out of 5 random seeds, 50 HC steps (otherwise 1 seed and 30 HC steps, see Chapter 2.11 for other parameters)
(**)	Graphs that have the exact solutions

Table 3: Comparative table of results.

Graph	$\|V\|$	$\|E\|$	[KH] after 10 V-Cycles	Our cost after 1 V-cycle
tooth	78136	452591	254099155	239144220
ocean	143437	409593	140087494	134314337
rotor	99617	662431	245237685	234608385

Table 4: Comparative table of the results on large graphs.

Graph	$\|V\|$	$\|E\|$	Our cost	Optimal cost
mesh33x33	1089	2112	32016	31680
bintree10	1023	1022	3700	3696
hc10	1024	5120	523776	523776
Proper Interval Graph I	200	3213	30766	30766
Proper Interval Graph II	500	14784	250241	250241
Proper Interval Graph III	1000	45393	1.19709e+06	1.19709e+06

Table 5: Comparative table of the results on the graphs with known minima.

2.9.3 Experimenting with larger graphs

The linear running time facilitates its application to graphs whose sizes are larger than those in Petit's test suite. We have found only one paper [32] with results on large graphs. In order to compare our results to others we decided to take several large graphs from [32]. The results appear in Table 4. The algorithm was tested on the same machines, but the number of HC steps, minimization sweeps and V-cycles was decreased. The tests were done with one random seed only. Since the time of simulated annealing was decreased - efficiency was reduced too. The results may be significantly improved by increasing the number of V-cycles and time of local relaxations.

2.9.4 Experimenting with graphs with known optimal solution

In the case of graphs where the optimal MinLA value is known, we can use those optimal values to measure the quality of the results obtained by the heuristics we have described. The results of applying the algorithm to these graphs are reflected in Table 5. In all cases we used the same parameters for the algorithm. Besides the graphs from Petit's suite we added three proper interval graphs which also have a known minima (see section "MinLA on Proper Interval Graphs"). In most cases of the graphs with known minima, the results of our algorithm are equal to the optimal.

2.10 Conclusions and future work

We have presented a new heuristic for the MinLA problem on general graphs. The heuristic is based on the multilevel paradigm and becomes at least comparable to the existing heuristics. The running time of the algorithm, which is linear in the number of edges, allows to apply it on large graphs. Since the algorithm contains several stochastic parts, we have

checked the convergence of the results. The stochastic variability was not more than $1-2\%$. Our algorithm is still in the process of development. Many improvements are anticipated, including the following :

- Possible improvement of the local segment minimization may be done using dynamic programming which solves the problem exactly. Since its running time is $O(|E|2^n)$, this technique can be applied only to small segments.

- In Chapter 2.7 we discussed the set of solutions taken at the coarsest level. The important question is how to choose this set of solutions during the passing from level L to $L-1$. One can choose the set of the best (in their costs) solutions. However there may be a replication of the information in the half of neighbor pairs of solutions (solution and its reverse). For example, taking two best solutions, we obtain the orders which are equal up to rotation.

- The advantage of the minimization process, that was described in Chapter 2.6.5, is in the possibility to implement it easily. However the global strategy of minimization can be formulated in other way : always make the best possible flip. Such strategy demands a special data structure which allows a fast update of the future possible flips in the current configuration. This strategy is introduced in [20].

Our main conclusion is that the average and the best results of our algorithm 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. On the other hand the tests with uniform random graphs show that some results obtained by stochastic heuristics are better than our multigrid algorithm. Our algorithm succeeds when the graph has non-stochastic structure, i.e., in other more intuitive words it has "some geometry". The algorithm finds the parts of the dense substructures of the graph but does not completely separates them away. The algorithm is based on the basic principle that during the multigrid process each vertex may be associated to more than just one cluster according to some "likelihood" measure. Finding this measure is one of the main goals of our algorithm. Combining this principle with several local minimization techniques that improve the arrangements we obtain the full algorithm. The running time of our algorithm is linear at every level. Thus, the process can be applied on very large graphs. We recommend our multigrid algorithm as a general method for solving the Minimum Linear Arrangement Problem.

2.11 Overview of the Parameters used in algorithm

This section is devoted to an overview of the parameters used by the algorithm as an input besides the graph itself. During all runnings the parameters remained the same. After the name of each parameter we give the value used in the runnings reported in Table 3.

- **THRESHOLD_COARSE_NODES**(0.4) The lower bound of the strength of connection between fine node to its coarse neighbors in the coarsening procedure. The parameter takes values between 0 and 1. When this parameter is insufficient to reach the entire amount of coarse nodes, it grows up by 0.025.

- **AMOUNT_OF_COARSE_NODES**(0.33 - between 0 and 1) Minimum amount of coarse nodes.
- **POWER_OF_CONNECTION_BETWEEN_FINE_AND_ITS_COARSES**(50 - in %) Used in calculation of interpolation weights as lower bound of connection's strength between fine and its remained coarse neighbors. This parameter was equal to 20 for randomA[23].
- **MAXIMUM_OF_CONNECTIONS_BETWEEN_FINE_AND_ITS_COARSES**(100) Used in calculation of interpolation weights as an upper bound of amount of coarse neighbors of fine vertex. This parameter was equal to 10 for randomA[23].
- **RELAX_SWEEPS**(50) The number of relaxation sweeps. The parameter used in minimization procedure only. The minimization is stopped when the entire pass over all vertices do not improve the energy.
- **SA_ALPHA**(0.75) The coefficient used to decrease the temperature of simulated annealing.
- **SA_COOR**(0.125) This factor is inserted to decrease the temperature in simulated annealing. When the difference in coordinates of vertices influence on the probability of a flip the temperature should be decreased.
- **LCC_TRESHOLD**(100) The minimum number of vertices in the graph above which the LCC is used.
- **BASIS_SOLUTIONS_NUMBER**(10) The number of arrangements taken from the coarsest level.
- **V_CYCLES_NUM**(3) The number of full V-cycles.
- **SEED** Random seed (for SA)
- **W_CYCLE_CONS** The constant α for the energy calculation in W-cycles.
- **HOT_COLD_STP**(30) The number of HC steps in SA at finest level.
- **CONST_NUMBER_OF_HC**(NO) "YES" – Use the same number of HC steps in SA; "NO" – Decrease the number of HC steps in SA proportionaly to the number of vertices in the coarse graphs.
- **START_SA1PERC**(95) The number of flips (in %) that can be allowed at the beginning of SA with distance 1. Used for a calculation of $TStart$.
- **FINISH_SA1PERC**(10) The number of flips (in %) that can be allowed at the end of SA with distance 1. Used for a calculation of $TEnd$.
- **START_SA2PERC**(20) The number of flips (in %) that can be allowed at the beginning of SA with distance 2. Used for a calculation of $TStart$.

- **FINISH_SA2PERC**(1) The number of flips (in %) that can be allowed at the end of SA with distance 2. Used for a calculation of $TEnd$.
- **USE_LCC_ALGORITHM**(YES) LCC Algorithm switch on/off.

3 The Minimum Linear Arrangement Problem on Proper Interval Graphs

3.1 Preliminaries

Let $\mathcal{F}$ be a family of nonempty sets. The *intersection graph* of $\mathcal{F}$ is obtained by representing each set in $\mathcal{F}$ by a vertex and connecting two vertices by an edge if and only if their corresponding sets intersenct. The intersection graph of a family of intervals on a linearly ordered set (like the real line) is called an *interval graph*. If these intervals are constructed such that no interval properly contains another then such graph is called a *proper interval graph*. The families of interval and proper interval graphs are widely studied and used in different fields. In this chapter we present an algorithm which produces an optimal solution of the MinLA on proper interval graphs.

Let us construct graph $G = (V, E)$ in a following way (algorithm A):

- set n as number of vertices in a graph
- drop n vertices on an axis with integer coordinates from 1 to n
- take a subset of successive vertices and make a clique from them
- return to the previous step t times

As a result of this construction we obtain a graph with the representation like on Figure 7. If we have a situation with nested cliques, we can ignore the clique that is placed inside of some other clique. We solve the problem for a family of graphs obtained by applying the algorithm A and then show that there is an algorithm which produces such representation for proper interval graphs.

In the following claims we will work with a graph $G = (V, E)$ that is a chain of k cliques $C_1 ... C_k$ constructed using algorithm A. In all following orders we index the vertices from 1 to n, where $|V| = n$.

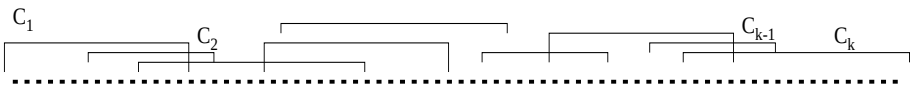

Figure 7: Schematic example of the chain of cliques

The orders of the vertices that preserve the order of cliques $C_1, C_2, ..., C_k$ and full rotation $C_k, C_{k-1}, ..., C_1$ will be called 'natural orders' (or $N - order$) of G.

Denote by d_v the degree of vertex v.

3.2 The Algorithm

Let us formulate and prove two claims that will serve as the basis of induction for Claim 3.3.

Claims 3.1 and 3.2 refer to a graph $G = (V, E)$ that is a union of two cliques C_1 and C_2 whose intersection is not empty and no clique contains the other (all other cases of union of two cliques are trivial).

Claim 3.1 *Given a graph $G = (V, E)$. If G is a union of two cliques C_1 and C_2 then in every optimal linear order of G the last vertex will not be from the intersection of C_1 and C_2.*

Proof: Take any optimal order φ. Assume that the claim is false and the last vertex u in optimal arrangement φ comes from the intersection. Let us take vertex v that is not in the intersection and $\varphi(v) = max_{x \notin (C_1 \cap C_2)} \varphi(x)$. Suppose that $\varphi(v) = n - k$ and w.l.o.g. $v \in C_1 \setminus (C_1 \cap C_2)$. Exchange u and v. Since

$$N(v) \subset N(u),$$

by exchanging u and v, the connections to $N(v)$ do not increase the cost of φ after the flip. However u is connected to the vertices from $C_2 \setminus (C_1 \cap C_2)$ too. By moving u to the position $n-k$ we reduce the cost of these edges, and does not change the cost of edges from u to the vertices at $[n-k+1, ..., n-1]$ positions (because they are from $C_1 \cap C_2$). So $u \rightleftarrows v$ flip must reduce the cost of the optimal arrangement and this is a contradiction to the assumption.

Claim 3.2 *Given a graph $G = (V, E)$. If G is an union of two cliques C_1 and C_2 then every optimal linear order of G has the following structure*

$$< \{v \mid v \in C_1 \setminus (C_1 \cap C_2)\}, \{u \mid u \in (C_1 \cap C_2)\}, \{w \mid w \in C_2 \setminus (C_1 \cap C_2)\} >$$

or

$$< \{v \mid v \in C_2 \setminus (C_1 \cap C_2)\}, \{u \mid u \in (C_1 \cap C_2)\}, \{w \mid w \in C_1 \setminus (C_1 \cap C_2)\} >$$

Call the order of this type $N_{2C} - order$.

Proof: Let us prove the claim by induction on a number of vertices in G.

- The basis of induction is when $|V| = 1$ and the claim in this case is true.
- Suppose that if $|V| = n - 1$ the claim is true.
- Let $|V| = n$. Take some optimal order φ of G. Suppose, that w.l.o.g. the last vertex in φ is $v \in C_1 \setminus (C_1 \cap C_2)$ (as was proved in Claim 3.1). Remove v from G with all adjacent edges. We obtain a new graph $G^{'}$ with $n-1$ vertices. Look at its arrangement ψ such that $\psi(u) = \varphi(u)$. There are two possible cases : ψ is optimal and ψ is not optimal.

  If ψ is optimal then, by induction hypothesis, it has $N_{2C} - order$ type and, after returning v to G, we save an $N_{2C} - order$ type. Since we started with the optimal order then φ has $N_{2C} - order$ type.

  Assume that ψ is not optimal. Then there exists some optimal order ρ for $G^{'}$ and $C_{G^{'},\psi} > C_{G^{'},\rho}$. Thus

$$C_{G,\varphi} \geq C_{G^{'},\psi} + \alpha > C_{G^{'},\rho} + \frac{d_v(d_v+1)}{2}$$

  Recall that φ is optimal for G. But $C_{G^{'},\rho}$ produces a cost of linear order which is $N_{2C} - order$ on $G^{'}$ and, by adding vertex v, we save $N_{2C} - order$ increasing $C_{G^{'},\rho}$ by $\frac{d_v(d_v+1)}{2}$. So we obtain a strict inequality for given optimal order φ and this is a contradiction. Thus ψ cannot be not optimal.

Let us switch to the main claim.

Claim 3.3 *Given a graph $G = (V, E)$ constructed using algorithm A then every minimum linear order of G is an N-order.*

Proof:

Suppose that after removing all nested cliques, the graph will remain with k different cliques. We can assume that G is connected, otherwise the problem can be divided into the similar subproblems per connected component. Let us prove it by induction on k.

- When $k = 0$ or 1 the claim is trivial and when $k = 2$ we prove it in claims 3.1 and 3.2.
- Suppose that the claim is true for $k - 1$ and let us prove it for k cliques chain. Here we use the second induction on the number of vertices in $T = C_k \setminus \cup_{i=1}^{k-1} C_i$ (call it l).
  1. When $l = 0$ there are $k - 1$ cliques in G and the claim is true by induction hypothesis.
  2. If $l = 1$ there exist unique vertex v in T. We will call vertex $u \in G$ p-vertex if it belongs to the intersection of C_k with some other clique. Assume that we have some optimal order ϕ. If ϕ is N-order then the claim is true. Suppose that ϕ is not an N-order. Then there exist two possible cases of ϕ's structure depicted in Figure 8.

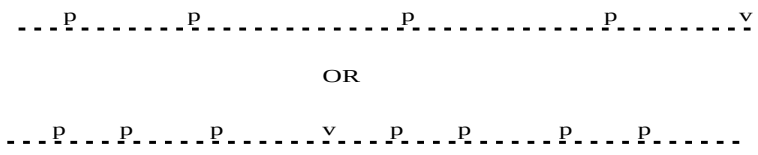

Figure 8: Possible cases of structure of ϕ

First case : If $\phi(v) > \phi(p_i)$ (or less) for any i then

$$C_{G,\phi} \;=\; C_{G\setminus v,\phi} + \alpha$$

where $\alpha \geq \frac{d_v(d_v+1)}{2}$. From the other hand we have

$$C_{G,N-order} \;=\; C_{G\setminus v,N-order} + \frac{d_v(d_v+1)}{2} \tag{5}$$

Then

$$C_{G\setminus v,\phi} + \alpha \;\leq\; C_{G\setminus v,N-order} + \frac{d_v(d_v+1)}{2}$$

$$C_{G\setminus v,\phi} + \alpha - \frac{d_v(d_v+1)}{2} \;\leq\; C_{G\setminus v,N-order}$$

So we obtain that ϕ is at least good at $G \setminus v$ as $N - order$. The situation when it is better is impossible by induction hypothesis and if it is not better then it is exactly $N - order$ and then on G it will be also $N - order$.

Second case : v is placed between p-vertices. Suppose w.l.o.g. that there are L p-vertices p_i such that

$$\forall\ p_i\ \phi(p_i) < \phi(v),\ 1 \leq i \leq L$$

and R p-vertices q_i such that

$$\forall\ q_i\ \phi(q_i) > \phi(v),\ 1 \leq i \leq R$$

then

$$C_{G,\phi}\ \geq\ C_{G\setminus v,\phi} + \frac{R(R+1)}{2} + \frac{L(L+1)}{2} + LR.$$

From the other hand we have equation 5. Combining them we obtain

$$\begin{aligned} C_{G\setminus v,N-order} + \frac{d_v(d_v+1)}{2}\ &= \\ &= C_{G\setminus v,N-order} + \frac{(L+R)(L+R+1)}{2}\ \geq \\ &\geq C_{G\setminus v,\phi} + \frac{R(R+1)}{2} + \frac{L(L+1)}{2} + LR \end{aligned} \quad (6)$$

and then

$$C_{G\setminus v,N-order} \geq C_{G\setminus v,\phi}. \quad (7)$$

If inequality 7 is strict, this is a contradiction to the induction hypothesis and if there is an equality, ϕ should be an $N - order$ by induction. However, placing v in the middle of the $N - order$ cannot allow the optimal order and this is a contradiction too.

3. Suppose the claim is true for $|T| = l - 1$ and now we need to prove it for l. Let v be the l-th vertex in T. Assume that ϕ is some optimal order. If ϕ is an $N - order$ then the claim is true. Suppose that ϕ is not an $N - order$. If v is the last (or first, the same proof in this case) vertex in ϕ then

$$C_{G,\phi}\ =\ C_{G\setminus v,\phi} + \alpha$$

where $\alpha \geq \frac{d_v(d_v+1)}{2}$. From the other hand we have

$$C_{G,N-order}\ =\ C_{G\setminus v,N-order} + \frac{d_v(d_v+1)}{2}$$

Then

$$C_{G\setminus v,\phi} + \alpha \;\leq\; C_{G\setminus v,N-order} + \frac{d_v(d_v+1)}{2}$$
$$C_{G\setminus v,\phi} + \alpha - \frac{d_v(d_v+1)}{2} \;\leq\; C_{G\setminus v,N-order}$$

So we obtain that ϕ is at least good at $G \setminus v$ as $N - order$. If ϕ is better then it is impossible by the induction hypothesis. Otherwise, if it is not then ϕ is exactly $N - order$ and then on G it will be also $N - order$.

Suppose that v is not last (or first) and no other vertex from T is (in this case we can flip them and obtain the same optimal order). Then there are two indexes i and j s.t. $1 < i < j < n$, where i is a first occurrence of some vertex from T in ϕ and j is a last. Let us flip v with j-th vertex and call the new order ϕ'. The cost of ϕ' is remained the same as it was in ϕ because the flipped vertices have the same set of neighbors. Like in a previous case, there are L p-vertices to the left of v and R to the right. If $R = 0$, we remove v like in the case of $l = 1$ and obtain a contradiction to the induction on l. Otherwise, if $R > 0$ then

$$C_{G,\phi'} \;\geq\; C_{G\setminus v,\phi'} + \frac{R(R+1)}{2} + \frac{(l+L-1)(L+l)}{2} + (L+l-1)R.$$

From the other hand we have equation 5. Combining them we obtain

$$\begin{aligned} C_{G,N-order} \;&=\; C_{G\setminus v,N-order} + \frac{d_v(d_v+1)}{2} \;= \\ &=\; C_{G\setminus v,N-order} + \frac{(l-1+L+R)(l+L+R)}{2} \;\geq \\ &\geq\; C_{G,\phi'} \;\geq\; C_{G\setminus v,\phi'} + \frac{R(R+1)}{2} + \frac{(l+L-1)(L+l)}{2} + (L+l-1)R \end{aligned} \quad (8)$$

This case is similar to the case of $l = 1$ but while removing v we use the induction on l and the claim is true.

Now we can define a process of calculation of the MinLA on proper interval graphs in polynomial time.

Claim 3.4 *The minimum linear order of proper interval graph is calculated in polynomial time.*

Proof: Proper interval graphs can be recognized by a linear order: their vertices can be linearly ordered such that the vertices contained in the same clique are consecutive. The recognition works in linear time and produces such order [27]. Graphs with such possible order may be created with algorithm A. Then the minimum linear arrangement of proper interval graphs is calculated in polynomial time.

Thus, the algorithm for calculation of the MinLA is following :

1. Apply any polynomial algorithm for proper interval graph recognition which produces an $N - order$ of vertices,

2. This order is a minimum linear arrangement.

3.3 4-Approximation for interval graphs

The complexity of the MinLA problem on general interval graphs is not known. The order of vertices by their start (or finish) time is not optimal already on the example at Figure 9. In this chapter we introduce 4-approximation algorithm for the problem on interval graphs.

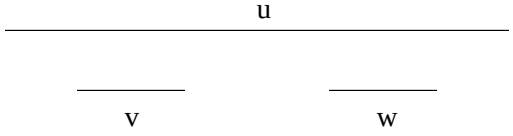

Figure 9: Counterexample of interval graph.

Before we will start with the algorithm let us look at some preliminary facts :

Theorem 3.5 (Gilmore-Hoffman [1964]) *The following are equivalent :*

1. *G is an interval graph.*

2. *It is possible to order the maximal cliques in G so that for every $v \in V$ the cliques that contain v appear consecutively in this order.*

The polynomial algorithms that construct such orders are introduced in many sources (for example in [24]) given an interval graph (possibly with no interval representation). Suppose we have such clique order ϕ

$$C_1, C_2, ..., C_k.$$

Then for every vertex $v \in V$ it is possible to define a segment $[C_i, ..., C_{i+l}]$ in ϕ that contains the maximal cliques of v. Let us call the corresponding i and $i + l$ by s_v and f_v respectively (start and finish). Now we can order the vertices of the graph by their s_v values (call the order by π and ordered vertices by $v_1, ..., v_n$).

Theorem 3.6 *Linear order π is a 4-approximation for the Minimum Linear Arrangement problem on general interval graphs.*

Proof: Let us estimate the cost of arbitrary π-order. We call an incident to u edge $e = (uv)$ in order π – "right oriented edge" if

$$\pi(v) > \pi(u).$$

The similar definition will be for "left oriented edges". Every vertex in π has "right oriented edges" and/or "left oriented edges" as well. Every edge in π is right oriented and left oriented for its different ends. We will pass vertex by vertex in order π and estimate the total cost

of right oriented edges for every vertex and this will give the estimation of the total cost of the order π. Denote by R_j the set of right oriented edges of vertex v_j in order π.

$$C_{G,\pi} = \sum_{v_i \in V} \sum_{e \in R_i} cost_\pi(e).$$

Since the vertices are ordered by their s_v values

$$\sum_{e \in R_i} cost_\pi(e) \leq \frac{d_{v_i}(d_{v_i}+1)}{2}.$$

Thus

$$C_{G,\pi} \leq \sum_{v \in V} \frac{d_v(d_v+1)}{2}.$$

From the other hand for any optimal order ψ the following is true :

$$C_{\psi^*} \geq \frac{\sum\limits_v \left(2\frac{\frac{d_v}{2}(\frac{d_v}{2}+1)}{2} \right)}{2} = \frac{\sum\limits_v \left(\frac{d_v}{2}(\frac{d_v}{2}+1) \right)}{2}. \tag{9}$$

Combining previous results we obtain

$$A = \frac{\sum\limits_v \left(\frac{d_v}{2}(\frac{d_v}{2}+1) \right)}{2} \leq C_{\psi^*} \leq C_\pi \leq \sum_v \frac{d_v(d_v+1)}{2} = B. \tag{10}$$

Clearly that for $\alpha = 4$

$$B \leq \alpha A$$

and this proves that any π-order gives 4-approximation for the Minimum Linear Arrangement problem on general interval graphs.

Easy to see that given some interval representation of interval graph, the π-order is equivalent to the order by start (or finish) time of the intervals.

References

[1] D. Adolphson and T.C. Hu, *Optimal Linear ordering*, SIAM Journal of Applied Mathematics, 25(3):403-423, 1973

[2] 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, vol. 5, no. 4, pp. 1-27, 2001.

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

[4] S.L. Bezrukov, *Edge isoprmetric problems on graphs (a survey)*, Graph theory and combinatorial biology, volume 7, 157-197, 1999.

[5] J. Bhasker and S. Sahni, *Optimal Linear Arrangement of Circuit Components*, Journal of VLSI and Computer Systems, Vol 2, pages 87-109, 1987.

[6] A. Brandt, *Multi-level adaptive technique (MLAT) for fast numerical solutions to boundary value problems.*, In Proc. 3rd Int. Conf. on Numerical Methods in Fluid Mechanics (Cabannes, H. and Temam, R., eds.), Lecture Notes in Physics 18, Springer-Verlag, 1973, pp. 82-89.

[7] 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.

[8] 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.

[9] A. Brandt, *Guide to multigrid development*, in Multigrid methods (W. Hackbush and U. Trottenberg, eds.), Springer-Verlag, 220-312, 1982.

[10] A. Brandt, *General highly accurate algebraic coarsening*, Gauss Minerva Technical Reports, can be retrieved at "http://www.wisdom.weizmann.ac.il/ achi/gmc.html", 2000.

[11] 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.

[12] A.Brandt, *Multiscale Scientific Computation: Review 2000*, can be retrieved at "http://www.wisdom.weizmann.ac.il/ achi/review00.ps", 2000.

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

[14] F.R.K. Chung, *Labelings of graphs*, In L. Beineke and R.Wilson *Selected topics in graph theory*, 3, 151-168. Academic Press, 1988.

[15] J. Diaz, J. Petit, and M. Serna. *A survey on graph layout problems.* ACM Computing Surveys, 2002. To appear.

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

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

[18] S. Even and Y. Shiloach, *NP-completeness of several arrangements problems*, TEchnical report TR-43, The Technion, Haifa, 1978.

[19] R. Fedorenko, *The speed of convergence of an iterative process.*, USSR comp. Math. Phys., 4(3):227-235, 1964.

[20] C. M. Fiduccia and R. M. Mattheyses, *A linear time heuristic for improving network partitions*, Proc. ACM/IEEE Design Automation Conf. (1982) pp. 175-181.

[21] P. Fishburn, P. Tetali and P. Winkler, *Optimal linear arrangement of a rectangular grid*, Discrete Mathematics, 213:123-139, 2000.

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

[23] M.K. Goldberg and I.A. Klipker, *An algorithm for minimal numeration of tree vertices*, Sakharth. SSR Mecn. Akad. Moabe, 81(3):553-556, 1976. In Russian.

[24] M. Golumbic, *Algorithmic graph theory and perfect graphs*, Academic Press, 1980.

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

[26] L.H. Harper, *Chassis layout and isopermetric problems*, Technical report SPS 37-66, vol II, Jet Propulsion Laboratory, 1970.

[27] Celina M. Herrera de Figueredo, Joao Meidanis and Celia Picinin de Mello. *A lexBFS Algorithm for Proper Interval Graph Recognition*, Instituto de Computago, UNICAMP, Technical Report DCC-04/93, 1993.

[28] S. B. Horton, *The optimal linear arrangement problem: algorithms and approximation*, PhD. Thesis, 1997

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

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

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

[32] 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), Springer Verlag, to appear.

[33] P.J.M. van Laarhoven, E.H.L. Aarts, *Simulated Annealing: Theory and Applications*, 1988

[34] 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.

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

[36] D.O. Muradyan and T.E. Piliposyan, *Minimal numberings of vertices of a rectangular lattice*, Akad. Nauk Armjan. SSR, 1(70):21-27, 1980.

[37] J. Petit, *Approximation heuristics and benchmarkings for the MinLA problem*, In R. Battiti and A. Bertossi, editors, Alex '98, Building bridges between theory and applications, pages 112-128. Universit di Trento, 1998.

[38] J. Petit, *Experiments for the MinLA problem*, Report de recerca LSI-01-7-R, Departament de Llenguatges i Sistemes Informtics, Universitat Politcnica de Catalunya, 2001.

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

[40] S. Rao and A. Richa, *New approximation techniques for some ordering problems*, Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 1998.

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

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

[43] Y. Shiloach, *A minimum linear arrangement algorithm for undirected trees*, SIAM Journal of Computing, 8(1):15-32, 1979

[44] R. Southwell, *Stress calculation in frameworks by the method of systematic relaxationof constraints I, II.*, Proc. Roy. Soc. London Ser., A 151:56-95, 1935.