A Multi-Scale Algorithm for the Linear Arrangement Problem

Yehuda Koren and David Harel

Dept. of Computer Science and Applied Mathematics
The Weizmann Institute of Science, Rehovot, Israel
{harel,yehuda}@wisdom.weizmann.ac.il

Abstract. Linear ordering vertices of a graph is a common problem arising in diverse applications. In this paper we present a novel algorithm for this problem. The algorithm is based on the multi-scale paradigm, and runs in a linear time. Results are similar to those of the best known approaches, while running time is significantly faster, enabling dealing with much larger graphs. The paper contains a special multi-scale construction, which may be used for a broad range of ordering problems.

1 Introduction

Many computational tasks are involved with the combinatorial problem of ordering the vertices of a graph as to optimize a certain objective functions. Applications range from speeding up computations related to sparse matrices to error correcting codes, visualization, computational biology and even archaeological dating. The reader is referred to [7] for a detailed survey.

We concentrate on a common variant of the problem, known as *the minimum linear arrangement problem (MinLA)*, which consists of placing the n vertices in positions $1 \dots n$ on a line, as to minimize the sum of the edge lengths. Putting differently, we are given n pins, some of which are connected by wires, and we want to lay the pins in a sequence of holes in a way that minimize the total wire length. This variant arises in VLSI design, graph drawing, modeling nervous activity in the cortex and job scheduling, see [7]. As most ordering problems, MinLA was proved to be NP-hard and the corresponding decision problem is NP-complete [8].

In this paper we suggest a linear-time heuristic algorithm for the MinLA problem. An experimental study shows its attractiveness in terms of output quality and running time. The proposed algorithm embodies a novel multi-scale scheme for the MinLA problem. This multi-scale construction was crafted as to suit many linear ordering tasks and hence may have applications beyond the MinLA problem.

2 The Minimum Linear Arrangement Problem (MinLA)

Let $G(V, E)$ be a graph, with $V = \{1 \dots n\}$ a set of n vertices, and E a set of weighted edges, w_{ij} being the weight of the edge $\langle i, j \rangle$ connecting vertices i and j.

Sometimes, when the context is clear, we will write simply $\langle i, j\rangle$ instead of $\langle i, j\rangle \in E$. We denote the neighborhood of i by: $N(i) = \{j \mid \langle i, j\rangle \in E\}$. Throughout this paper we assume, without loss of generality, that G is connected. Otherwise, the problem can be solved independently for each connected component.

Definition 21 (Linear arrangement) *A linear arrangement of G is a bijection $\pi : V \longrightarrow \{1, \ldots, n\}$. Equivalently, we can define π as a permutation of V. We call $\pi(i)$ the location of vertex i.*

Definition 22 (Arrangement cost) *The cost of the linear arrangement π is denoted by:*

$$LA_\pi(G) \stackrel{def}{=} \sum_{\langle i,j\rangle \in E} w_{ij} \cdot |\pi(i) - \pi(j)|$$

The objective of the MinLA problem is to find a permutation π that minimizes $LA_\pi(G)$.

2.1 Algorithms

For certain classes of rather regular graphs, like trees, grids and hyper-cubes, there exist polynomial time algorithms for solving the MinLA problem. However, due to the NP-hardness of the problem, there is no polynomial time algorithm for general graphs. Thus most research is carried on approximation and heuristic algorithms. Before providing such algorithms, we want to describe an exact algorithm, which is better than the naive $\Theta(n!)$ algorithm that scan all possible permutations. This exact algorithm will serve us in the sequel.

Dynamic Programming Exact Algorithm Using dynamic programming we can find the minimum linear arrangement in time $O(2^n \cdot |E|)$ and space $O(2^n)$. The algorithm is based on the fact that we dub as *the locality of the minimum linear arrangement problem*. This fact states that the best ordering of a set of vertices S, which is placed to the right of the vertices of L and to the left of the vertices of R, is independent on the internal ordering of the vertices in L and in R. We formalize this fact in the following Theorem:

Theorem 1. *Let $G(V, E)$ be a graph and $L, S \subset V$ where $|L| = l$, $|S| = s$.*
Let π, φ be two permutations of V, such that $\pi(L) = \varphi(L) = \{1, \ldots, l\}$, $\pi(S) = \varphi(S) = \{l + 1, \ldots, l + s\}$.
Denote by π^*_S the permutation that minimizes the linear arrangement cost, over all permutations that differ from π only with respect to the places of vertices in S. (I.e., for all $i \notin S$, $\pi(i) = \pi^*_S(i)$.)
Define a permutation φ^*_S as follows:

$$\varphi^*_S(i) = \begin{cases} \varphi(i) & i \notin S \\ \pi^*_S(i) & i \in S \end{cases}$$

Then, φ^*_S minimizes the linear arrangement cost, over all permutations that differ from φ only with respect to the places of vertices in S.

For proving this theorem we use the following definition and lemma:

Definition 23 (Local arrangement cost) *Let $G(V, E)$ be a graph, $L \cup S \cup R = V$, where L, S, R are mutually disjoint sets and let π be a permutation of V, such that: $\pi(L) = \{1, \ldots, l\}$, $\pi(S) = \{l+1, \ldots, l+s\}$, $\pi(R) = \{l+s+1, \ldots, n\}$. The local arrangement cost of the set S w.r.t. π, denoted by $LA^S_\pi(G)$, is defined as:*

$$LA^S_\pi(G) = \sum_{\langle i,j\rangle,\, i,j \in S} w_{ij} \cdot |\pi(i) - \pi(j)| + \sum_{\langle i,j\rangle,\, i \in S, j \in L} w_{ij} \cdot (\pi(i) - l) + \\ + \sum_{\langle i,j\rangle,\, i \in S, j \in R} w_{ij} \cdot (l + s - \pi(i)) + \sum_{\langle i,j\rangle,\, i \in L, j \in R} w_{ij} \cdot s$$

Lemma 1. *Let $G(V, E)$ be a graph, $L \cup S \cup R = V$, where L, S, R are mutually disjoint sets and let π be a permutation of V, such that: $\pi(L) = \{1, \ldots, l\}$, $\pi(S) = \{l + 1, \ldots, l + s\}$, $\pi(R) = \{l + s + 1, \ldots, n\}$. Then,*

$$LA_\pi(G) = LA^L_\pi(G) + LA^S_\pi(G) + LA^R_\pi(G).$$

Proof. We decompose the cost associated with the permutation π, as follows:

$$LA_\pi(G) = \sum_{\langle i,j\rangle} w_{ij} \cdot |\pi(i) - \pi(j)| = \\ \sum_{\langle i,j\rangle,\, i,j \in L} w_{ij} \cdot |\pi(i) - \pi(j)| + \sum_{\langle i,j\rangle,\, i,j \in R} w_{ij} \cdot |\pi(i) - \pi(j)| + \\ + \sum_{\langle i,j\rangle,\, i,j \in S} w_{ij} \cdot |\pi(i) - \pi(j)| + \sum_{\langle i,j\rangle,\, i \in S, j \in L} w_{ij} \cdot |\pi(i) - \pi(j)| + \\ + \sum_{\langle i,j\rangle,\, i \in S, j \in R} w_{ij} \cdot |\pi(i) - \pi(j)| + \sum_{\langle i,j\rangle,\, i \in L, j \in R} w_{ij} \cdot |\pi(i) - \pi(j)|$$

Observe that:

- If $i \in S$, $j \in L$: $|\pi(i) - \pi(j)| = (\pi(i) - l) + (l - \pi(j))$
- If $i \in S$, $j \in R$: $|\pi(i) - \pi(j)| = (l + s - \pi(i)) + (\pi(j) - l - s)$
- if $i \in L$, $j \in R$: $|\pi(i) - \pi(j)| = (l - \pi(i)) + (\pi(j) - l - s) + s$.

Using these equations and rearranging rows, we can write:

$$LA_\pi(G) = LA^L_\pi(G) + LA^S_\pi(G) + LA^R_\pi(G).$$

□

Now we are ready to prove Theorem 1:

Proof. Define the set $R = V - (L \cup S)$, the set of vertices placed to the right of S. Consider a permutation φ_S for which for every $i \notin S$, $\varphi_S(i) = \varphi(i)$. Using lemma 1, the cost associated with the permutation φ_S is:

$$LA_{\varphi_S}(G) = LA^L_{\varphi_S}(G) + LA^S_{\varphi_S}(G) + LA^R_{\varphi_S}(G).$$

Since for all $i \notin S$, $\varphi_S(i) = \varphi(i)$, we can write:

$$LA_{\varphi_S}(G) = LA^L_{\varphi}(G) + LA^S_{\varphi_S}(G) + LA^R_{\varphi}(G).$$

We conclude that minimizing $LA_{\varphi_S}(G)$ over all permutations that differ from φ only in places of vertices of S, is equivalent to finding such a permutation φ_S that minimize $LA^S_{\varphi_S}(G)$.

Similarly, we can write:

$$LA_{\pi^*_S}(G) = LA^L_{\pi}(G) + LA^S_{\pi^*_S}(G) + LA^R_{\pi}(G).$$

Recall that π^*_S is the minimizer of arrangement cost over all permutations that differ from π only in places of vertices of S. $LA^S_{\pi^*_S}(G)$ depends only on the ordering of vertices in S, and on the content of the set L (or, equivalently, R). Hence, setting $\rho = \pi^*_S$ minimizes $LA^S_{\rho}(G)$ over all permutations satisfying that the set of vertices to the left of S is L.

The permutation φ^*_S is defined such that $LA^S_{\varphi^*_S}(G) = LA^S_{\pi^*_S}(G)$. Thus, setting $\varphi_S = \varphi^*_S$ minimizes $LA_{\varphi_S}(G)$, over all permutations that differ from φ only with respect to the places of vertices in S. □

The locality property implies that when one constructs a linear arrangement by incrementally trying all partial arrangements growing from left to right, the best ordering of the remaining vertices does not depend on the exact ordering of the vertices that have already been placed. Hence, among all different orderings of some $L \subset V$ on places $\{1, \ldots, |L|\}$, the ordering that minimizes the local arrangement cost of L is also superior in a global sense, and there is no need to remember the rest orderings.

This is the heart of the dynamic programming algorithm that stores for each of the 2^n subsets of V the best ordering which was found. The algorithm is depicted in Figure 1. Its time complexity is $O(2^n \cdot |E|)$, since when calculating the best cost of a new subset, we scan each edge at most twice. Space complexity is $O(2^n)$.

To our best knowledge, this dynamic programming algorithm is not mentioned in papers dealing with the MinLA problem. This is probably due to its impractical time and space demands. However, this algorithm will play a role in our multi-scale algorithm.

Spectral sequencing Spectral information has been successfully applied to vertex ordering problems, see e.g., [1],[3],[11]. We review here the basic idea of these methods:

Given a graph $G(V = \{1, \ldots, n\}, E)$, the corresponding Laplacian is a symmetric $n \times n$ matrix L defined as follows:

$$L_{ij} = \begin{cases} \sum_{\langle i,k \rangle \in E} w_{ik} & i = j \\ -w_{ij} & i \neq j,\ \langle i,j \rangle \in E \\ 0 & i \neq j,\ \langle i,j \rangle \notin E \end{cases} \qquad i,j = 1, \ldots, n.$$

```
Function MinLA_DP (G(V = {1,...,n}, E), Ordering)
% Input: A graph G(V, E)
% Output: vector Ordering containing the minimum linear arrangement of V
% Data structure: A table T whose entries are indexed by subsets of V.
% For a S ⊆ V, T[S].cost holds the minimal local arrangement cost of S,
% T[S].cut is the weighted sum of edges connecting S with V − S,
% T[S].right_vtx is the rightmost vertex in the best local arrangement of S.
% The function Cut^G(v, Nodes) returns the weighted sum of edges
% connecting vertex v and Nodes ⊂ V.

  % Initialize table:
  for every S ⊆ V
    table[S].cost ← ∞
  end for
  table[φ].cost ← 0
  table[φ].cut ← 0

  % Fill table:
  for i = 1 to n
    for every S ⊂ V, |S| = i − 1
      cut^S ← table[S].cut
      new_cost ← table[S].cost + cut^S
      for every j ∉ S
        if table[S ∪ {j}].cost > new_cost then
          table[S ∪ {j}].cost ← new_cost
          table[S ∪ {j}].right_vtx ← j
          table[S ∪ {j}].cut ← cut^S − cut^G(j, S) + cut^G(j, V − S)
        end if
      end for
    end for
  end for

  % Retrieve optimal ordering:
  S ← V
  for i = n to 1
    v ← table[S].right_vtx
    ordering[i] ← v
    S ← S − {v}
  end for
end
```

Fig. 1. An algorithm for finding optimal MinLA using dynamic programming

Let v be the normalized eigenvector of L corresponding to the second smallest eigenvalue. Hall [9] has shown that v is the best non-trivial minimum of the following quadratic energy:

$$E = \sum_{\langle i,j \rangle \in E} w_{ij} \cdot (v_i - v_j)^2. \quad (1)$$
$$\text{subject to: } \|v\|_2 = 1$$

As a matter of fact, v is a vector of fundamental importance to many fields as it reflects connectivity properties of the graph, see, e.g., [16]. So fundamental to be privileged in having a unique name — the Fiedler vector.

Sequencing the vertices is done by sorting them according to their components in v. Fortunately, computation of v can be done very rapidly using the multi-scale (or "multi-level") methods of [4],[14]. In fact, in [14] we have computed the two eigenvectors corresponding to the second and the third smallest eigenvalues of the Laplacian of million vertex graphs in less than a minute (on a Pentium 3 PC). Computing only v takes half of this time. Hence, spectral sequencing is a very rapid heuristic for computing linear arrangements.

Simulated annealing Simulated annealing [13] is a powerful, general (if slow) optimization technique that is appropriate for the MinLA problem. This method repeatedly changes the ordering as follows. Given the current candidate ordering π, a new candidate $\hat{\pi}$ is generated that is close to π. If $LA_{\hat{\pi}} \leqslant LA_{\pi}$ the new ordering – $\hat{\pi}$ is accepted. Otherwise, $\hat{\pi}$ may be accepted with a probability that decreases as the process proceeds. Accepting an ordering that worsen the current situation is called *an up-hill move*. Up-hill moves enable escaping bad local minima, and enhance the search space.

Petit [17] has conducted extensive tests of many algorithms for the MinLA problem, on several classes of graphs. We quote the conclusion of Petit:

> ...the best heuristic to approximate the MinLA problem on sparse graphs is Simulated Annealing when measuring the quality of the solutions. However, this heuristic is extremely slow whereas Spectral Sequencing gives results not too far from those obtained by Simulated Annealing, but in much less time. In the case that a good approximation suffices, Spectral Sequencing would clearly be the method of choice.

In the following sections, we will describe an algorithm, whose quality is similar to that of simulated annealing, while running time is much faster.

3 The Median Iteration

In this section we describe *the median iteration*, a rapid randomized iterative algorithm for decreasing the cost of a linear arrangement. The heart of this method is a continuous relaxation of the MinLA problem, where we allow vertices to share the same place, or to be placed on non-integral points.

Given a graph $G(V,E)$, a *linear placement* is a function $p : V \longrightarrow \mathbb{R}$. The cost of the linear placement p is denoted by:

$$LP_p(G) = \sum_{\langle i,j\rangle \in E} w_{ij} \cdot |p(i) - p(j)|$$

Notice that linear arrangement, which was defined as a bijection $V \longrightarrow \{1,\ldots,n\}$, is a special case of linear placement.

It is obvious that the minimum of $LP_p(G)$ is when all the vertices are placed at the same location. Hence, the minimal linear placement is not informative. What will be useful is a *process* of minimizing $LP_p(G)$. Such a process can be used for improving linear arrangements, as we shall describe.

We begin by defining the median of the places of i's neighbors.

Definition 31 (Median) *Let $G(V,E)$ be a graph and p a linear placement. Given some $i \in V$, the median of i's neighborhood is denoted by $med_p^G(i)$.*
$med_p^G(i)$ is a usual median, hence it should satisfy:

$$\sum_{j\in N(i), p(j)\leqslant med_p^G(i)} w_{ij} \geqslant \sum_{j\in N(i), p(j)> med_p^G(i)} w_{ij}$$
$$\sum_{j\in N(i), p(j)\geqslant med_p^G(i)} w_{ij} \geqslant \sum_{j\in N(i), p(j)< med_p^G(i)} w_{ij}$$

When equality holds in these two inequalities, the median can be any point inside an open interval. In this case, we arbitrarily choose the middle point of this interval as the median.

The main observation is the following Theorem:

Theorem 2. *Let $G(V,E)$ be a graph and p a linear placement. Fix the places of all vertices except a single $i \in V$. A location of i that minimizes the linear placement cost is $med_p^G(i)$. (This location is not unique, since there may be infinite medians.)*

Proof. Denote by $p[i \to x]$ a linear placement such that:

$$p[i \to x](j) = \begin{cases} x & j = i \\ p(j) & j \neq i \end{cases}$$

Let $\epsilon = \sum_{j\in N(i), p(j)\leqslant med_p^G(i)} w_{ij} - \sum_{j\in N(i), p(j)> med_p^G(i)} w_{ij}$. By the definition of the median, $\epsilon \geq 0$. Let $\delta > 0$. Observe that:

$$LP_{p[i\to med_p^G(i)+\delta]}(G) \geq LP_{p[i\to med_p^G(i)]}(G) + \delta * \epsilon$$

Hence, for every $x > med_p^G(i)$ we get:

$$LP_{p[i\to x]}(G) \geq LP_{p[i\to med_p^G(i)]}(G).$$

The proof for $x < med_p^G(i)$ goes along the same lines. □

Now we can define an iterative process for reducing the cost of a linear placement, based on minimizing the cost associated with each vertex, separately.

Function Median_Iteration $(G(V,E),\ p,\ k)$
% Parameters:
% $G(V,E)$ - a graph, p - a linear placement, k - no. of sweeps
  repeat k times:
    for every $i \in V$
      $p(i) \leftarrow med_p^G(i)$

  % When a valid linear arrangement is needed:
  Sort nodes according to respected entries in p
  for every $i \in V$
    $p(i) \leftarrow \langle$sorted place of $i\rangle$

Suppose that the initial value of p is a valid linear arrangement. When k is overly large, a significant fraction of V will be placed in a single point, leaving us very few information. But, for a not too large value of k, we will get many small clusters of vertices placed together. This gives us important information regarding the global structure of the linear arrangement. Hence, we can construct a new valid linear arrangement by sorting the nodes according to their values in the linear placement p. The decision about the internal order of vertices that were placed at the same point is random.

We call this method the median iteration. Computation of a median can be done in a linear time. Hence, the time complexity is $O(k \cdot |E|)$ for the k sweeps plus $O(n \log n)$ for sorting the nodes by their place. Since k is fixed (a usual value is $k = 50$), the total time complexity is $O(|E| + n \log n)$. When the initial p is a linear arrangement, so all points are placed on integral coordinates, we can use bucket-sort to order the nodes in linear time. This requires a slight modification of the median definition: when the median is inside an open interval between two integral points, we would pick one of these two points as the median. This leads to a running time $O(|E|)$.

The median iteration is a simple and rapid way for reducing the cost of a linear arrangement. This method addresses the global structure of the ordering, while making local decisions randomly. This sort of distinguishing between local and global characteristics, is quite reasonable in the MinLA problem, due to the locality property (Theorem 1), since we can perform later local refinements without increasing the cost of the arrangement.

Relation to spectral sequencing There is a connection between the median iteration and spectral sequencing. The both methods relax the original ordering problem as a continuous placement problem. However, the cost functions are different. The median iteration minimizes $LP_p(G)$, which generalizes the original cost function of linear arrangements – $LA_\pi(G)$. However, spectral sequencing minimizes Hall's energy (Equation 1), which is a quadratic function unlike $LA_\pi(G)$.

Interestingly, by differentiating Hall's energy with respect to v_i, we reveal a version of Theorem 2, for Hall's energy:

> Fix the places of all vertices except a single $i \in V$. The location of i that minimizes Hall's energy is the (weighted) *average* of the places of i's neighbors.

Hence, in the minimizer of Hall's energy, the Fiedler vector, each vertex is placed, roughly, in the average place of its neighbors. While, for minimizing the cost of a linear arrangement, it would be better to place each vertex in the median place of its neighbors. Experiments with the median iteration, as described in Section 5, validate the ability of the median iteration to improve upon spectral sequencing.

4 Multi-Scale Algorithm

The *multi-scale* (or, *multi-level*) paradigm is a powerful general technique that allows fast exploration of properties related to the 'global structure' of complex objects, that depend on many elements within. Multi-scale algorithms have proved to be successful in a variety of areas in physics, chemistry and engineering. See, e.g., [18],[5],[6].

These techniques progressively express an originally high-dimensional problem in lower and lower dimensions in a process called *coarsening* of the problem. In the coarsest scale the problem is solved exactly, and then starts a *refinement* process in which the solution is progressively projected back into higher and higher dimensions, updated appropriately at each scale, until the original problem is reproduced. A multi-scale algorithm should be specifically designed to a problem, such that the coarsening process keeps the essence of the problem unchanged.

In terms of its use for dealing with graph-theoretic optimization problems, the multi-scale approach is widely used for graph-partitioning, see, e.g., [12]. More recently, Walshaw [19] has used this approach for the TSP and vertex-coloring problems. We have used the multi-scale approach for the related problem of drawing graphs aesthetically [10],[14].

4.1 Segment Graphs

One of the most prominent properties of multi-scale algorithms is that they keep the inherent structure of the problem unchanged during coarsening. In our case, the form of the MinLA problem as was introduced in Section 2 is not preserved during coarsening. But, we can define a more general problem that is preserved during coarsening, and which also contains the original problem as a special case. Hence, we dedicate this first part to describe this general problem. In fact, this general problem emerges naturally from trying to arrange a more general entity, that we dub a *segments-graph (s-graph)*.

Definition 41 (segments graph (s-graph)) *An s-graph is a usual graph $G(V,E)$, whose vertices are line segments. The length of vertex i is denoted by l_i ($l_i \geqslant 0$). Each edge $\langle i,j\rangle$ is associated with two coordinates: (1) The place of its endpoint inside vertex i, denoted by ${}_P\langle i,j\rangle$. (2) The place of its endpoint inside vertex j, denoted by $\langle i,j\rangle_P$. Certainly, $0 \leqslant_P \langle i,j\rangle \leqslant l_i,\ 0 \leqslant \langle i,j\rangle_P \leqslant l_j$.*

In a linear arrangement of an s-graph, we place the vertices on a line, in such a way that no two vertices intersect. Since we seek for an arrangement with short edges, we can safely rule out unused gaps between consecutive vertices. Hence, we define a linear arrangement of an s-graph as follows:

Definition 42 *Let G be an s-graph. A linear arrangement of G is a bijection $\pi : V \longrightarrow \{1,\ldots,n\}$. The location of vertex i is denoted by $p^\pi_G(i)$, and satisfy: $p^\pi_G(i) = \sum_{j,\pi(j)<\pi(i)} l_j$. Often, when the identity of the related G is clear, we abbreviate $p^\pi_G(i)$ by $p^\pi(i)$.*

Definition 43 *Given an s-graph G, and a linear arrangement π, the length of edge $\langle i,j\rangle$, w.r.t. π, is defined as follows:*

$$len^\pi_G(\langle i,j\rangle) \stackrel{def}{=} |p^\pi(j) + \langle i,j\rangle_P - p^\pi(i) -_P \langle i,j\rangle|$$

Often, when the identity of the related G is clear, we abbreviate $len^\pi_G(\langle i,j\rangle)$ by $len^\pi(\langle i,j\rangle)$.

Now we generalize the notions of cost and of local-cost of a linear arrangement (which were given in Definitions 22 and 23), as to handle s-graphs:

Definition 44 *Given an s-graph G, the cost of the linear arrangement π is denoted by:*

$$LA_\pi(G) \stackrel{def}{=} \sum_{\langle i,j\rangle \in E} w_{ij} \cdot len^\pi(\langle i,j\rangle)$$

Definition 45 *Let $G(V,E)$ be an s-graph, $L \cup S \cup R = V$, where L,S,R are mutually disjoint sets and let π be a permutation of V, such that: $\pi(L) = \{1,\ldots,l\}$, $\pi(S) = \{l+1,\ldots,l+s\}$, $\pi(R) = \{l+s+1,\ldots,n\}$.*
Denote by a, b the boundaries of $p^\pi(S)$, i.e., $a = p^\pi(\pi^{-1}(l+1))$, $b = p^\pi(\pi^{-1}(l+s)) + l_{\pi^{-1}(l+s)}$.
The local arrangement cost of the set S w.r.t. π, denoted by $LA^S_\pi(G)$, is defined as:

$$LA^S_\pi(G) = \sum_{\langle i,j\rangle,\, i,j\in S} w_{ij} \cdot len^\pi(\langle i,j\rangle) + \sum_{\langle i,j\rangle,\, i\in S, j\in L} w_{ij} \cdot (P^\pi(i) +_P \langle i,j\rangle - a) +$$
$$+ \sum_{\langle i,j\rangle,\, i\in S, j\in R} w_{ij} \cdot (b - P^\pi(i) -_P \langle i,j\rangle) + \sum_{\langle i,j\rangle,\, i\in L, j\in R} w_{ij} \cdot (b-a)$$

Given a regular graph G (not an s-graph), if we set the length of each vertex to be 1, and for every $\langle i,j\rangle$, ${}_P\langle i,j\rangle = \langle i,j\rangle_P = 1$, the definitions of the generalized

cost and local-cost of a linear arrangement coincide with the regular definitions (Definitions 22 and 23). Hence, the newer definitions generalize the older ones, and we can keep using the same notations.

It is easy to verify that Theorem 1 and Lemma 1 are still valid for s-graphs, without any modification.

The dynamic programming algorithm can be applied for s-graphs, with slight modifications. In this algorithm we are computing for each subset $S \subset V$ the best local arrangement cost of S in places $\{1, \ldots, |S|\}$. What is needed is to change the algorithm, as to compute the local arrangement according to Definition 45, instead of Definition 23. Actually, this is a simple modification.

Given an s-graph $G(V, E)$ and a linear placement p, in order to activate the median-iteration on G, we need to take into account the exact location of edges, as follows. For an edge $\langle i, j \rangle$, define the directed distance from i to j as $d_{ij} = p(j) + \langle i, j \rangle_P - p(i) -_P \langle i, j \rangle$. Adding to the place of i the weighted median of the directed distances from i to its neighbors (distance d_{ij} is weighted by w_{ij}), is the best move when only movements of i are allowed. The median iteration should be modified accordingly.

4.2 The Coarsening Process

During the coarsening, we iteratively represent an initial s-graph as smaller and smaller s-graphs. Let G be such an s-graph containing n nodes. A single coarsening step would be to replace G with another s-graph G^R containing only $m < n$ nodes (typically, $m \approx \frac{1}{2}n$). Of course, the structure of G^R should be strongly linked to that of G, such that both will describe approximately the same entity, but in different scales. Moreover, a good linear arrangement of G^R, should correspond to a good linear arrangement of G.

Our attitude to coarsening, is by posing *restrictions* on the possible arrangements. These restrictions reduce the search space, so the restricted problem is of a "lower dimension", and can be formulated using a smaller graph. More specifically, the restrictions that we use are imposing certain pairs of vertices to be placed sequentially together. We hope that those pairs of vertices are arranged fairly close in the minimal arrangement of G. In this optimistic situation, there exist a solution in the restricted subspace, which is quite close to the optimal solution, up to some local refinement that does not change the global structure of the arrangement, but only affects local portions of the arrangement.

Now we elaborate the details of this approach.

Restricting the search space We want to restrict the search space in a way that satisfies two conflicting goals:

1. The degrees of freedom should be significantly reduced, enabling a representation by a much smaller graph.
2. The minimal arrangement is affected only locally; while its global structure is preserved. Putting it more formally, let π^* be the minimal arrangement. There exist an arrangement π satisfying the restrictions, such that for every vertex i, $|\pi(i) - \pi^*|$ is bounded by some constant.

Is the formal expression useful??

We have found a way that attempts to achieve these two competing goals. Consider some initial arrangement π, which was constructed by some fast algorithm like spectral sequencing or median iteration (or both together). For simplicity assume that the number of vertices, n, is even. For every pair of vertices v_1, v_2 such that $\pi(v_1) = 2i-1$, $\pi(v_2) = 2i$ for some $i \geqslant 1$, introduce a restriction that any feasible linear arrangement φ would have to satisfy: $\varphi(v_2) = \varphi(v_1) + 1$. The restricted problem is of arranging $n/2$ *restricted vertex pairs*. Hence, the size of the *restricted search space* is $(n/2)!$ while the size of the original search space was $n!$.

In a case that n is odd, we choose randomly some odd k, $1 \leqslant k \leqslant n$. Then we restrict consecutive pairs in the series $\pi^{-1}(1), \ldots, \pi^{-1}(k-1), \pi^{-1}(k+1), \ldots, \pi^{-1}(n)$. Vertex $\pi^{-1}(k)$ is not restricted.

In a case that the initial arrangement π was not too far from the minimal arrangement, the restrictions we have introduced affect the solution only locally, as desired.

discuss advantages of coarsening based on a given solution, rather than based on the structure of the graph

Now we are showing how to formulate the restricted problem again as a linear arrangement problem, but of a much smaller graph.

Formulating the restricted problem as a coarser MinLA problem Given an s-graph $G(V, E)$, and a set $R \subset V \times V$ of restricted vertex pairs, we build a coarser graph $G^R(V^R, E^R)$, such that there is a 1-1 correspondence between linear arrangements of G^R and linear arrangements in the restricted search space of G.

Before describing G^R, we want to remark that throughout this section we assume that if $\langle i, j\rangle \notin E$ then $w_{ij} = 0$ and ${}_P\langle i, j\rangle = \langle i, j\rangle_P = 0$.

G^R is produced by contracting pairs of restricted vertices. Vertex lengths and edge weights are preserved, by accumulating them. Places of the new edges are calculated by averaging old places. We give here formally the construction of G^R. For simplicity, we assume that n is even, so every $v \in V$ appears in a single restricted pair.

- The coarse vertex set V^R is simply the set of restricted vertex pairs, R.
- The length of a coarse vertex (v_1, v_2) is $l_{(v_1,v_2)} = l_{v_1} + l_{v_2}$
- The coarse edge set is defined as follows:

$$E^R = \left\{ \langle (v_1, v_2), (v_3, v_4)\rangle \;\middle|\; \begin{array}{l} (v_1, v_2), (v_3, v_4) \in V^R, \\ \langle v_1, v_3\rangle \in E \text{ or } \langle v_1, v_4\rangle \in E \\ \text{ or } \langle v_2, v_3\rangle \in E \text{ or } \langle v_1, v_4\rangle \in E \end{array} \right\}$$

- The weight of a coarse edge $\langle (v_1, v_2), (v_3, v_4)\rangle$, denoted as usual by $w_{(v_1,v_2)(v_3,v_4)}$ is $w_{v_1v_3} + w_{v_1v_4} + w_{v_2v_3} + w_{v_2v_4}$.
- The place of endpoints of coarse edge $\langle (v_1, v_2), (v_3, v_4)\rangle$ is the weighted average of endpoints of the corresponding fine edges:

$${}_P\langle (v_1, v_2), (v_3, v_4)\rangle = \frac{\left(\sum_{i\in\{v_1,v_2\}, j\in\{v_3,v_4\}} w_{ij} \cdot {}_P\langle i, j\rangle\right) + l_{v_1} \cdot (w_{v_2v_3} + w_{v_2v_4})}{w_{v_1v_3} + w_{v_1v_4} + w_{v_2v_3} + w_{v_2v_4}}$$

$$\langle (v_1, v_2), (v_3, v_4)\rangle_P = \frac{\left(\sum_{i\in\{v_1,v_2\}, j\in\{v_3,v_4\}} w_{ij} \cdot \langle i, j\rangle_P\right) + l_{v_3} \cdot (w_{v_1v_4} + w_{v_2v_4})}{w_{v_1v_3} + w_{v_1v_4} + w_{v_2v_3} + w_{v_2v_4}}$$

When n is odd, there exist a single unrestricted vertex – v. In this case we add a vertex (v, v) to V^R, where $l_{(v,v)} = l_v$. Definitions regarding edges adjacent to (v, v) should be slightly changed.

There is a simple 1-1 correspondence between the restricted linear arrangements of the fine graph G and the linear arrangements of the coarse graph G^R:

Definition 46 (Correspondence) *Let G be a graph, R a set of restricted vertex pairs, π a restricted linear arrangement of G and let φ be a linear arrangement of the respected coarse graph G^R. The two arrangements are* corresponding *if:*

$$\text{For every } (v_1, v_2) \in R: \ \pi(v_1) = 1 + \sum_{\varphi((i,j)) < \varphi((v_1,v_2))} |(i,j)|$$

where,

$$|(i,j)| \stackrel{def}{=} \begin{cases} 2, & i \neq j \\ 1, & i = j \end{cases}$$

Or equivalently, in the inverse direction:

$$\text{For every } (v_1, v_2) \in R: \ \varphi((v_1, v_2)) = \sum_{(i,j) \in R,\, \pi(i) \leqslant \pi(v_1)} 1$$

Notice that when n is even: $(i, j) \in R \Rightarrow i \neq j$, so we can simply write the condition for correspondence as:

$$\text{For every } (v_1, v_2) \in R: \ \pi(v_1) = 2 * \varphi((v_1, v_2)) - 1$$

Using the close relationship between length of vertices in G and G^R, observe that for two corresponding linear arrangements, π and φ, the actual locations of the vertices are related as follows:

$$\text{For every } (v_1, v_2) \in R : P^{\pi}_{G}(v_1) = P^{\varphi}_{G^R}((v_1, v_2)) \tag{2}$$

During coarsening several edges become self-loops, and are canceled. We calculate the cost related to these lost edges:

Definition 47 *Given a graph $G(V, E)$ and a set of restricted vertex pairs R, we define* the internal cost of R *as:*

$$C(R) = \sum_{\langle i,j \rangle \in E, (i,j) \in R} w_{ij} \cdot (\langle i, j \rangle_P + l_i +_P \langle i, j \rangle)$$

The most important fact is that the costs of two corresponding linear arrangements are identical, up to adding $C(R)$ which is independent on the specific arrangements.

Lemma 2. *Let G be a graph, R a set of restricted vertex pairs, and let π and φ be two corresponding linear arrangements of G and G^R, respectively. Then, $LA_\pi(G) = LA_\varphi(G^R) + C(R)$.*

The proof is rather technical and is given in Appendix B.

We conclude that given a set of restricted vertex pairs, R, we can construct a coarser graph G^R with $\lceil \frac{n}{2} \rceil$ vertices. Linear arrangements of G^R correspond to linear arrangements in the restricted search space of G, and have the same costs (up to adding a uniform constant). For reasonable restrictions, the restricted search space contains arrangements of similar structure to that of the minimum cost arrangements. Hence, we hope that a good arrangement of G^R corresponds to a quite good arrangement of G. This quite good arrangement of G can become a real good one, by *the local refinement process*.

4.3 Local Refinement Process

Given a linear arrangement π of a graph G, we are seeking for a method that improves the quality of π. Such a candidate method can be the median iteration described in Section 3. However, often we are dealing with a case where π has a good global structure, as a result of minimizing the problem on restricted search space. For such a case, the inability of the median iteration to make clever local decisions may be a serious disadvantage. For this reason we have constructed a local refinement process, which specializes in optimizing locally an arrangement.

Our idea is to take each k consecutive vertices in π and to rearrange them such that their *local arrangement cost* (see Definition 45) is minimized. Such an optimal arrangement is achieved using a close variant of the dynamic programming algorithm described in Section 2. By the locality property this local strategy is feasible. We can be sure that optimizing small vertex sets, separately, can only decrease the cost of the arrangement. For improving the results, the process can be iterated few times.

The local refinement process is depicted in Figure 2. Taking k as a constant (in our experiments we usually set $k = 6$), the time complexity is $O(|E|)$ and space demands are also linear in the input size.

4.4 Putting it all Together

At this stage we have all the elements needed for constructing the multi scale algorithm.

Preprocessing stage Prior to running the algorithm we want to obtain, very quickly, a reasonable linear arrangement. For achieving this, we first order the vertices using spectral sequencing (see Section 2) and then improve the result by applying the median iteration for about 50 sweeps. We have no evidence that performing spectral sequencing prior to the median iteration is superior to a simple greedy initialization. Anyway, since we have an extremely fast multi-scale implementation of spectral sequencing [14], it was very comfortable for us to use it. However, the reader that finds the implementation of spectral sequencing too complicated, may replace it with a greedy vertex-by-vertex successive addition algorithm. Such algorithms are described in [17],[15].

Function Local_Refine $(G(V = \{1, \ldots, n\}, E), Ordering)$
% Parameters:
% $G(V, E)$ – An s-graph , $Ordering$ – a linear arrangement of V
% Constants:
% $interval[= 6]$ – number of consecutive vertices to optimize together
% $iterations[= 5]$ – number of iterations the algorithm should run
% Auxiliary function:
% MinLA_local($G(V, E)$, $Ordering$, j, $j + k - 1$) – finds best internal
% ordering of the k vertices placed in: $Ordering[j], \ldots, Ordering[j + k - 1]$.
% Implemented using dynamic programming

for $i = 1$ to $iterations$
  for $j = 1$ to $n - interval + 1$
    MinLA_local($G(V, E)$, $Ordering$, j, $j + interval - 1$)
  end for
end for

Fig. 2. The local refinement process

The V-cycle Our basic multi-scale tool is the *V-cycle*. The V-cycle starts by refining locally the arrangement. The intention of the refinement is not only to minimize the arrangement cost, but also to improve the quality of the following coarsening step. The next step is to coarsen the graph based on restricting consecutive vertex pairs of the current arrangement. Then we solve the problem in the restricted solution space, by running recursively on the coarse graph. Once we have found a good solution in the restricted solution space, we refine this solution locally (in the full solution space).

In the most optimistic case, in each scale the optimal solution is reachable via local optimization from the best restricted solution, what ensures finding it. Our hope is that in a realistic case, a pretty good solution is reachable via local optimization from the restricted solution we have found. Of course, this depends on the quality of the refinement process and on the restrictions we impose.

a needless repetition of the same idea???

In Figure 3 we are showing the V-cycle algorithm. The V-cycle uses several functions. The function 'coarsen(G, π, G^R, π^R)' gets an s-graph - G and a linear arrangement of its vertices - π. Based on this, 'coarsen' produces a coarse graph - G^R and a linear arrangement its vertices - π^R. π and π^R are corresponding linear arrangements in terms of Definition 46. The way to construct G^R and π^R is described in Subsection 4.2. The function 'interpolate(G^R, π^R, π)', gets a coarse graph - G^R and a linear arrangement- π^R, and produces the corresponding linear arrangement of the fine graph - π.

The stopping condition can be when the graph is small enough to facilitate finding the optimum linear arrangement. In all the graphs that we have tested, the local refinement was useless after more than five levels of recursion, due to

the good global structure of the arrangement. Anyway, the coarsest graphs are so small that the exact stopping point has only a negligible effect on the running time.

All the functions that the V-cycle use run in a linear time and space, and the recursive call is to a problem of approximately the half size. Thus, the time and space complexity of the V-cycle algorithm are $O(|E|)$.

Function V-cycle ($G(V,E)$, $Ordering$)
% Parameters:
% $G(V,E)$ – An s-graph , $Ordering$ – a linear arrangement of V

Local_Refine($G, Ordering$)
if G is 'large enough' **then**
coarsen($G, Ordering, G^R, Ordering^R$)
V-cycle($G^R, Ordering^R$)
interpolate($G^R, Ordering^R, Ordering$)
Local_Refine($G, Ordering$)
end if

Fig. 3. The V-cycle (multi-scale) algorithm

Iterating the V-cycle The V-cycle can benefit from initialization with a better arrangement. Thus, a repetitive activation of the V-cycle can improve the results. In fact, this kind of iteration is common in multi-scale algorithms, see [18].

Iterating the V-cycle can only decrease the cost of the arrangement. We have noticed that frequently, after few iterations the process converges and then improvement is very slow, if at all. We can obtain better results by initiating the median iteration, for few (e.g., 10) sweeps, before entering each new V-cycle. The median iteration perturbs the arrangement and provides the next V-cycle with a different starting point, overcoming premature convergence. An interesting point is that since the arrangement may be quite good, the inability of the median iteration to optimize local regions of the arrangement become significant. Thus, the median iteration, may perturb the arrangement into a "worse" arrangement of a higher cost. This reminds "uphill moves" of simulated annealing. But, the median iteration is much better than arbitrary uphill perturbations, as it inherently tends to the global optimum. Hence, we have found that few sweeps of the median iteration are quite effective, even when they temporarily increase the arrangement cost. Furthermore, the resemblance with simulated annealing may be extended. Accepting uphill perturbations can be subjected to a probability that decreases as the process goes on. Similarly, we may gradually reduce the number of sweeps of the median iteration, lessening the effect of uphill perturbations.

In Figure 4 we depict the full multi-scale linear arrangement algorithm.

```
Function MS_MinLA (G(V,E), Ordering, Iterations)
% Parameters:
%   G(V,E) – An s-graph
%   Ordering – a linear arrangement of V
%   Iterations – no. of V-cycle iterations
% Variables:
%   k1[= 40] – no. of sweeps in first median iteration
%   k2[= 10] – no. of sweeps in rest median iterations
  Spectral_Sequencing(G, Ordering)
  Median_Iteration(G, Ordering, k1)
  for i = 1 to Iterations
    Median_Iteration(G, Ordering, k2)
    V-cycle(G, Ordering)
    Decrease(k2)
  end for
```

Fig. 4. The full multi-scale algorithm

4.5 A Case Study

We want to demonstrate the benefit of embedding the local refinement process inside the multi-scale scheme. For this demonstration we have chosen the 10-dimensional hyper-cube, which consist of 1024 vertices and 5120 edges. The results for this graph reflect what have been observed with many other graphs. The optimal linear arrangement of this graph, denoted by π^*, can be computed efficiently, and its cost is: 523776, see [17].

As the first stage we have applied spectral sequencing to the hyper-cube, resulting in an arrangement of cost 659490. Then, we improved the arrangement using 40 iterations of the median iteration. We call the resulting arrangement π^0, its cost is 599958. We should note that this is the random part of the experiment, and other runs provide quite different arrangements (while the median iteration consistently significantly improves the initial arrangement).

We have tried to improve π^0 in two ways. First, we have applied our Local_Refine function for 100 iterations (setting the local constant *iterations* to 100), and we have got an arrangement of cost 593120. More iterations of Local_Refine cannot decrease the cost further. Second, we have applied a single V-cycle to π^0. In this case, the Local_Refine function has performed only 5 iterations in each of its executions during the V-cycle, thus running time is much faster than the first way. The result was the optimal linear arrangement, π^* of cost 523776. Clearly better than the direct application of Local_Refine.

Hence, embedding the local refinement process in the V-cycle multi-scale scheme has strong impact in terms of both running time and quality. In fact, embedding a local refinement process in a multi-scale scheme adds global considerations to the refinement process. This is because each local movement on

a coarse graph, expresses a more global move at the original graph. Thus, in some sense, the "wisdom" of a multi-scale algorithm can be divided into two parts: One part is related to local optimization and is encoded inside the local optimization process. The second part deals with the global properties of the problem, and is encoded in the coarsening construction.

5 Experiments

We have implemented our algorithm using C++ and used ACE [14] for spectral sequencing. The program runs on a 700MHZ Pentium 3, under Windows NT. We test our algorithm using a test suite collected by Petit [17]. The graphs were chosen as to represent several graph families. This test suite was also used by [2]. In Table 5 we describe the graphs.

Petit has computed linear arrangements of these graphs using many algorithms. The best results were obtained by first running spectral sequencing, and then refining using simulated annealing (SA). The SA process was run for $C \cdot n^2$ iterations ($C > 1$ is a constant related to the rate of temperature decrease in SA). This is quite a long run. We cannot directly compare running times with ours, since platforms are different. Anyway to get an impression, for the largest graph – whitaker3, running time of SA was over 11 hours. We provide the costs computed by SA in Table 5.

For each graph, we have run our multi-scale algorithm (of Figure 4) with $Iterations = 1$ (a single V-cycle) and also with $Iterations = 10$. We provide the results of these runs in Table 5. We also provide the results of spectral sequencing and median iteration, which are the first stage of our algorithm. It should be mentioned that this first stage (i.e., spectral sequencing and median iteration) is randomized, thus we have run it 10 times for each graph and picked the best result. It can be seen that the quality of our results (after 10 iterations) is very similar to that of Petit's SA (though our linear running time is significantly faster). In fact, the results may be further improved by executing more iterations and/or by enlarging the value of the constant *interval* in the function Local_Refine. However, the rate of improvement would be slow.

The table also shows the ability of the median iteration to be an efficient improvement over spectral sequencing. Its rapid running time makes it very attractive for huge graphs. Notice that for the graph randomA3, the median iteration increased the arrangement cost. This happens due to inability of the median iteration to make clever local decisions. Thus, in cases where the arrangement is close to optimal and local improvements are needed, the median iteration is inappropriate.

6 Related Work

Bar-Yehuda et al. [2] have applied a divide-and-conquer approach to the MinLA problem. Their idea is to divide the vertices into two sets, then to recursively arrange each set internally on consecutive places and finally to join the two ordered

Graph Name	Size |V|	|E|	Degree min/avg./max	Diameter	Result of SA
randomA1	1000	4974	1/9.95/21	6	900992
randomA2	1000	24738	28/49.47/72	3	6584658
randomA3	1000	49820	72/99.64/129	4	14310861
randomA4	1000	8177	4/16.35/29	4	1753265
randomG4	1000	8173	5/16.34/31	23	150940
hc10	1024	5120	10/10/10	10	548352
mesh33x33	1089	2112	2/3.88/4	64	34515
bintree10	1023	1022	1/1.99/3	18	4069
3elt	4720	13722	3/5.81/9	65	375387
airfoil1	4253	12289	3/5.78/10	65	288977
whitaker3	9800	28989	3/5.91/8	161	1199777
c1y	828	1749	2/4.22/304	10	63858
c2y	980	2102	1/4.29/327	11	79500
c3y	1327	2844	1/4.29/364	13	124708
c4y	1366	2915	1/4.26/309	14	117254
c5y	1202	2557	1/4.25/323	13	102769
gd95c	62	144	2/4.65/15	11	509
gd96a	1076	1676	1/3.06/111	20	104698
gd96b	111	193	2/3.47/47	18	1416
gd96c	65	125	2/3.84/6	10	519
gd96d	180	228	1/2.53/27	8	2393

Table 1. Test suite of Petit [17]. For each graph we provide the cost of the linear arrangement computed by Petit using simulated annealing

Graph Name	Spectral Sequencing	Median Iteration	Multi-Scale 1 Iteration	Multi-Scale 10 Iterations	Time (sec.) of a single V-cycle
	arrangement-cost / time (sec.)				
randomA1	1156890/.08	1020028/.03	938168/2.39	909126/22.94	2.28
randomA2	7377237/.27	7284497/.42	6755035/7.02	6606174/63.94	6.33
randomA3	15279645/.38	16543660/.96	14731040/12.12	14457452/109.17	10.78
randomA4	2167121/.11	1955837/.05	1807038/3.15	1765217/30.04	2.99
randomG4	195054/.06	175879/.06	154990/2.11	149513/18.97	1.99
hc10	580910/.02	542476/.03	523776/1.9	523776/18.51	1.85
mesh33x33	35750/.04	34118/.01	32486/1.04	31729/9.91	.99
bintree10	52992/.09	6114/0	4246/.88	3950/7.96	.79
3elt	429086/.43	394238/.11	385572/6.55	373464/61.24	6.07
airfoil1	352897/.37	312387/.11	305191/6.53	291794/61.02	6.05
whitaker3	1259607/.92	1238557/.28	1226902/13.86	1205919/127.79	12.66
c1y	103224/.02	71359/.01	66836/.92	64934/8.88	.89
c2y	95346/.03	84259/.01	82070/1.12	80148/10.81	1.08
c3y	175700/.04	145332/.02	137131/1.58	127315/15.3	1.52
c4y	133044/.05	124576/.02	121460/1.62	118437/15.57	1.55
c5y	144603/.04	115239/0.02	109280/1.39	104076/13.33	1.33
gd95c	599/.02	595/0	509/.08	509/.61	.06
gd96a	170700/.04	122567/.01	115525/1.24	106668/11.94	1.19
gd96b	1836/0	1825/.01	1435/.11	1434/.98	.1
gd96c	701/0	601/0	522/.06	519/.6	.06
gd96d	3691/0	2807/.01	2438/.16	2420/1.53	.15

Table 2. Results of: spectral sequencing, median iteration, multi-scale with a single V-cycle and multi-scale with 10 iterated V-cycles. The times for multi-scale include spectral sequencing, median iteration and V-cycles. We also provide the time of a single V-cycle

sets deciding which set will be put to the left of the other. The computed ordering is specified by a decomposition tree that describes the recursive partitioning of the subproblems. At each node of the tree there is a degree of freedom, regarding the order in which the two vertex sets are glued together. Thus, the goal of the algorithm is to decide for each node of the decomposition tree the order of its two children. Bar-Yehuda et al. [2] have suggested a dynamic programming algorithm for computing the best possible ordering for a given decomposition tree. They also applied their algorithm iteratively, starting each iteration with the result of the previous iteration. After few tens of iterations, they have got very good results similar to those of Petit's SA [17] and of ours.

It is interesting to compare our algorithm with that of [2], since both algorithms are hierarchic, one being divide-and-conquer (DAC) and the other being multi-scale (MS). A main difference between MS and DAC is expressed nicely in the following observation. In DAC, the nodes are first divided into two groups, which cannot be mixed during the entire process, a kind of a global constraint. In contrary, at MS there is no such a global constraint, except on the coarsest scale, where the vertices are restricted into few groups. Instead, MS imposes many local constraints, restricting small sets of vertices together, throughout all the hierarchy. Interestingly in the heart of the DAC approach of [2] lies a multi-scale algorithm of [12] that is used for building the decomposition tree.

Regarding running time, the method of [2] has a time complexity of $O(n^{2.2})$ for bounded degree graphs, while our method runs in time $O(|E|)$, which is $O(n)$ in the bounded degree case. This complexity gap is quite meaningful as graphs become large. For example consider the largest graphs in the test suite: 3elt, airfoil1 and whitaker3; the size of whitaker3 is slightly more than twice the size of 3elt or of airfoil1. Executing 10 iterations of [2] on these graphs take 534, 434 and 1652 seconds, respectively, comparing with 61,61 and 128 seconds, respectively, of our method. Times for [2] were taken from their paper, and reflect running a non-parallel program on a dual processor Pentium 3 600MHz, which is comparable with our platform. Notice that it is fair to compare 10 iterations of the both methods, since for these 3 graphs as for most graphs in the test suite our method produces better arrangements when using 10 iterations.

7 Discussion

We have presented a multi-scale algorithm for the minimum linear arrangement method. The algorithm delivers arrangements on a par with best known algorithms, using significantly less time, allowing to deal with much larger graphs.

The heart of the multi-scale algorithm is a novel construction of a hierarchy of coarse graphs, representing the problem in various dimensions. The coarse representations of the graph correspond to restricted parts of the original problem. A local refinement scheme was considerably amplified, when embedded inside the multi-scale construction. In fact, the multi-scale construction is independent on the specific refinement heuristic, and we believe that many heuristics may benefit from embedding inside a multi-scale scheme that allows them to traverse

the search space in several scales. Moreover, variants of the proposed multi-scale construction can be used for other vertex ordering problems such as Bandwidth minimization, and minimum Cutwidth [7].

We believe that several principles of our algorithm might be applicable for other combinatorial optimization problems. Especially, the process of constructing coarser representations of a problem by restricting the original problem.

Another contribution of our paper is the median iteration, which is an extremely fast method for decreasing arrangement cost, using a continuous relaxation of the original problem. This method may be applied for huge graphs of millions of vertices.

A Demonstration of the Coarsening Process

We illustrate the coarsening process, which was described in Subsection 4.2, using a specific example.

We begin with a graph G. All vertices of G are of length 1, all its edges are of weight 1 and for every edge $\langle i,j\rangle$: ${}_P\langle i,j\rangle = \langle i,j\rangle_P = 1$. An initial linear arrangement of G, denoted by π is depicted in Figure 5(a). Notice that in a linear arrangement, as it was defined in Definition 42, there is no gap between consecutive vertices. However to make the figure clearer we put small gaps between the vertices. As the reader can verify, the cost of this linear arrangement is $LA_\pi(G) = 43$.

The second step of the coarsening process is to restrict consecutive vertex pairs in π, to be adjacent in all the linear arrangements. Hence, the restricted pairs set is $R = \{(1,2),(3,4),(5,6),(7,8)\}$. Three edges lie within a restricted pair: $\langle 1,2\rangle$, $\langle 3,4\rangle$, $\langle 7,8\rangle$. Thus the internal cost, as defined in Definition 47 is $C(R) = 3$.

Based on the restriction R, we build a coarse graph – G^R, such that there is a 1-1 correspondence between its arrangements and the arrangements of the restricted search space. The vertices of G^R are the restricted pairs of R. The length of each vertex is 2, and weights and coordinates of edges were calculated as to preserve the information of the original graph.

In Figure 5(b) we show φ, a linear arrangement of G^R corresponding to π (see definition 46). The cost of φ is $LA_\varphi(G^R) = 40$. This value is in agreement with Lemma 2, as $LA_\pi(G) = LA_\varphi(G^R) + C(R)$.

As can be seen, the visual complexity of G^R is much lower than that of G. In fact, it is not too hard to compute the minimum linear arrangement of G^R – φ^*, which is depicted in Figure5(c). The cost of this arrangement is $LA_{\varphi^*}(G^R) = 24$.

After solving the problem in the coarse graph, we interpolate the resulting arrangement to the original graph, and get the corresponding linear arrangement of G – π^* shown in Figure 5(d). In accordance with Lemma 2 $LA_{\pi^*}(G) = LA_{\varphi^*}(G^R) + C(R) = 24 + 3$, a significant decrease of the arrangement cost, which can be also appreciated visually.

We should remark that this demonstration reflects only the coarsening process and not the full multi-scale algorithm, for two reasons. First, in the multi-

scale algorithm we perform refinements also on the fine graph, before the coarsening and after the interpolation. Second, in the multi-scale algorithm we do not start with an arbitrary arrangement of the fine graph, but with a hopefully good arrangement of G produced by the median iteration.

B Proof of Lemma 2

We provide here a proof of the following lemma, which is stated in Subsection 4.2

Lemma 2. *Let G be a graph, R a set of restricted vertex pairs, and let π and φ be two corresponding linear arrangements of G and G^R, respectively. Then, $LA_\pi(G) = LA_\varphi(G^R) + C(R)$.*

For simplicity we assume that the number of vertices in the fine graph, n, is even. Thus, $|R| = n/2,\ (i,j) \in R \Rightarrow i \neq j$. Otherwise, the proof should be slightly changed.

Proof. By definition,

$$\begin{aligned} LA_\pi(G) = &\textstyle\sum_{i,j} w_{ij} \cdot len^\pi(\langle i,j\rangle) = \\ &\textstyle\sum_{(v_1,v_2)\in R} w_{v_1v_2} \cdot len^\pi(\langle v_1,v_2\rangle) + \\ &\textstyle+ \sum_{(v_1,v_2),(v_3,v_4)\in R} \sum_{i\in\{v_1,v_2\},j\in\{v_3,v_4\}} w_{ij} \cdot len^\pi(\langle i,j\rangle) \end{aligned}$$

Let $(v_1,v_2),(v_3,v_4) \in R$ be two restricted pairs. Analyze their joint contribution to $LA_\pi(G)$, which is: $T = \sum_{i\in\{v_1,v_2\},j\in\{v_3,v_4\}} w_{ij} \cdot len^\pi(\langle i,j\rangle)$

Assume, without loss of generality, that $\pi(v_1) < \pi(v_3)$.
Recall that for an edge $\langle i,j\rangle$, for which $\pi(i) < \pi(j)$, we defined $len^\pi(\langle i,j\rangle) = P^\pi(j) + \langle i,j\rangle_P - P^\pi(i) -_P \langle i,j\rangle$.
Use also the fact: $P^\pi(v_2) = P^\pi(v_1) + l_{v_1}$, $P^\pi(v_4) = P^\pi(v_3) + l_{v_3}$.
Rewrite T as follows:

$$\begin{aligned} &T = \\ &w_{v_1v_3} \cdot P^\pi(v_3) + w_{v_1v_4} \cdot (P^\pi(v_3) + l_{v_3}) + w_{v_2v_3} \cdot P^\pi(v_3) + w_{v_2v_4} \cdot (P^\pi(v_3) + l_{v_3}) + \\ &+ w_{v_1v_3} \cdot \langle v_1,v_3\rangle_P + w_{v_1v_4} \cdot \langle v_1,v_4\rangle_P + w_{v_2v_3} \cdot \langle v_2,v_3\rangle_P + w_{v_2v_4} \cdot \langle v_2,v_4\rangle_P - \\ &- w_{v_1v_3} \cdot P^\pi(v_1) - w_{v_1v_4} \cdot P^\pi(v_1) - w_{v_2v_3} \cdot (P^\pi(v_1) + l_{v_1}) - w_{v_2v_4} \cdot (P^\pi(v_1) + l_{v_1}) - \\ &- w_{v_1v_3} \cdot_P \langle v_1,v_3\rangle - w_{v_1v_4} \cdot_P \langle v_1,v_4\rangle - w_{v_2v_3} \cdot_P \langle v_2,v_3\rangle - w_{v_2v_4} \cdot_P \langle v_2,v_4\rangle = \end{aligned}$$

$$\begin{aligned} &= (w_{v_1v_3} + w_{v_1v_4} + w_{v_2v_3} + w_{v_2v_4}) \times \\ &\times \Bigg(P^\pi(v_3) - P^\pi(v_1) + \\ &+ \frac{\left(\sum_{i\in\{v_1,v_2\},j\in\{v_3,v_4\}} w_{ij}\cdot\langle i,j\rangle_P\right) + l_{v_3}\cdot(w_{v_1v_4}+w_{v_2v_4})}{w_{v_1v_3}+w_{v_1v_4}+w_{v_2v_3}+w_{v_2v_4}} - \\ &- \frac{\left(\sum_{i\in\{v_1,v_2\},j\in\{v_3,v_4\}} w_{ij}\cdot_P\langle i,j\rangle\right) + l_{v_1}\cdot(w_{v_2v_3}+w_{v_2v_4})}{w_{v_1v_3}+w_{v_1v_4}+w_{v_2v_3}+w_{v_2v_4}} \Bigg) \end{aligned}$$

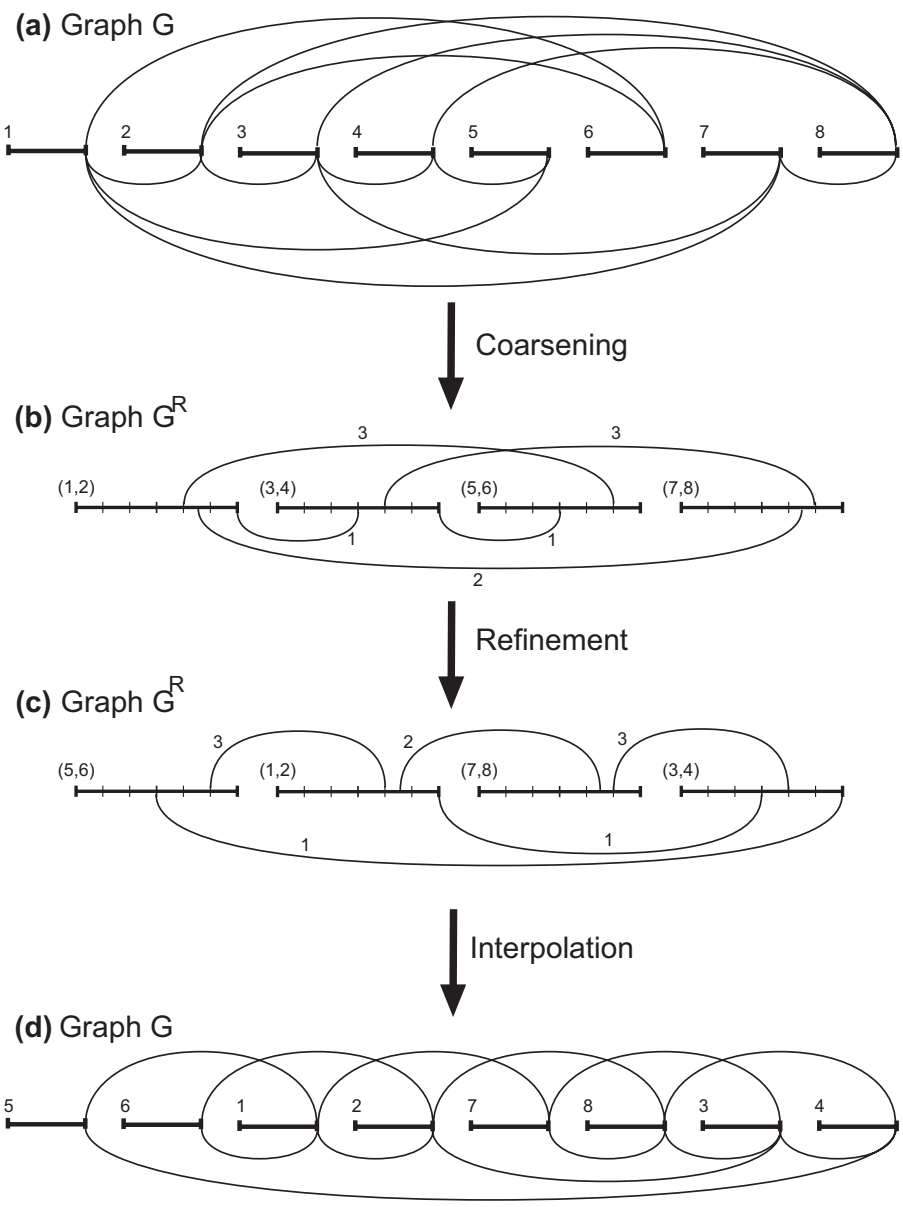

Fig. 5. The coarsening process

As observed in Equation 2, $P^{\pi}(v_1) = P^{\varphi}((v_1, v_2))$, $P^{\pi}(v_3) = P^{\varphi}((v_3, v_4))$. By the construction of G^R, we deduce:

$$\begin{aligned} T = \; & w_{(v_1,v_2)(v_3,v_4)} \times \\ & \times \Big(P^{\varphi}((v_3, v_4)) - P^{\varphi}((v_1, v_2)) + \\ & + \langle (v_1, v_2), (v_3, v_4) \rangle_P - \\ & - _P \langle (v_1, v_2), (v_3, v_4) \rangle \Big) \; = \\ = \; & w_{(v_1,v_2)(v_3,v_4)} \cdot len^{\varphi}_{G^R}(\langle (v_1, v_2), (v_3, v_4) \rangle) \end{aligned}$$

Hence,

$$\begin{aligned} LA_{\pi}(G) &= C(R) + \textstyle\sum_{(v_1,v_2),(v_3,v_4) \in R} w_{(v_1,v_2)(v_3,v_4)} \cdot len^{\varphi}_{G^R}(\langle (v_1, v_2), (v_3, v_4) \rangle) = \\ &= C(R) + LA_{\varphi}(G^R) \end{aligned}$$

□

References

1. J. E. Atkins, E. G. Boman and B. Hendrickson, "A Spectral Algorithm for Seriation and the Consecutive Ones Problem", *SIAM Journal on Computing* **28** (1998), 297–310.
2. R. Bar-Yehuda, G. Even, J. Feldman and Seffi Naor, "Computing an Optimal Orientation of a Balanced Decomposition Tree for Linear Arrangement Problems", *Journal of Graph Algorithms and Applications* **5** (2001), 1–27.
3. S. T. Barnard, A. Pothen, and H. D. Simon, "A Spectral Algorithm for Envelope Reduction of Sparse Matrices", *Numerical Linear Algebra with Applications* **2** (1995),317–334.
4. S. T. Barnard and H. D. Simon, "A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems", *Concurrency: Practice & Experience* **6** (1994), 101–117.
5. A. Brandt, "The Gauss Center Research in Multiscale Scientific Computation: Six Year Summary", Report WI/GC-12, Weizmann Institute of Science, May, 1999.
6. A. Brandt, "Multiscale Scientific Computation: 2000", *Proc. Yosemite Educational Symposium*, Lecture Notes in Computational Science and Engineering, Springer Verlag, 2001, to appear.
7. J. Diaz, J. Petit and M. Serna, "A Survey on Graph Layout Problems", Technical report LSI-00-61-R, Universitat Politècnica de Catalunya, Departament de Llenguatges i Sistemes Informàtics, 2000.
8. M. R. Garey and D. S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
9. K. M. Hall, "An r-dimensional Quadratic Placement Algorithm", *Management Science* **17** (1970), 219-229.
10. D. Harel and Y. Koren, "A Fast Multi-Scale Method for Drawing Graphs Nicely", *Proceedings of Graph Drawing'00* , LNCS 1984, Springer Verlag, pp. 183–196, 2000.
11. M. Juvan, B. Mohar, "Optimal Linear Labelings and Eigenvalues of Graphs", *Discrete Applied Math.* **36** (1992), 153–168.

12. G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs", *SIAM Journal on Scientific Computing* **20** (1999), 359–392.
13. Kirkpatrick, S., Gelatt, Jr., and Vecchi, M.P., "Optimization by Simulated Annealing", *Science* **220** (1983), 671-680.
14. Y. Koren, L. Carmel, and D. Harel "ACE: A Fast Multiscale Eigenvectors Computation for Drawing Huge Graphs", Technical Report MCS01-17, Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, 2001.
15. A. J. McAllister, "A New Heuristic algorithm for the Linear Arrangement Problem", manuscript.
16. B. Mohar, "The Laplacian Spectrum of Graphs", *Graph Theory, Combinatorics, and Applications* **2** (1991), 871–898.
17. J. Petit, "Experiments on the Minimum Linear Arrangement Problem", Technical report LSI-01-7-R, Universitat Politècnica de Catalunya, Departament de Llenguatges i Sistemes Informàtics, 2001. (Preliminary version in *Alex '98 — Building Bridges between Theory and Applications*, pp. 112–128, 1998.)
18. S. H. Teng, "Coarsening, Sampling, and Smoothing: Elements of the Multilevel Method", *Algorithms for Parallel Processing*, vol. 105 of *IMA Volumes in Mathematics and its Applications*, Springer Verlag, pp. 247–276, 1999.
19. C. Walshaw, "Multilevel Refinement for Combinatorial Optimisation Problems", Technical Report 01/IM/73, Comp. Math. Sci., Univ. Greenwich, London, UK, 2001.