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
{yehuda,harel}@wisdom.weizmann.ac.il
fax: +972-8-934-6023

Abstract. Finding a linear ordering of the vertices of a graph is a common problem arising in diverse applications. In this paper we present a linear-time algorithm for this problem, based on the multi-scale paradigm. Experimental results are similar to those of the best known approaches, while the running time is significantly better, enabling it to deal with much larger graphs. The paper contains a general multi-scale construction, which may be used for a broader range of ordering problems.

Keywords: linear ordering, minimum linear arrangement, combinatorial optimization, multi-scale / multi-level algorithms, dynamic programming, simulated annealing

1 Introduction

Many computational tasks involve the combinatorial problem of ordering the vertices of a graph so as to optimize certain objective functions. Applications range from speeding up computations on sparse matrices, via error correcting codes and visualization, to computational biology and even archaeological dating. The reader is referred to [9] for a detailed survey.

We concentrate here on a common variant of the problem, known as *the minimum linear arrangement problem (MinLA)*, which consists of placing the n vertices of the graph at positions $1 \ldots n$ on a line, so as to minimize the sum of the edge lengths. Put differently, given n pins, some of which are connected by wires, we want to fix the pins in a linear sequence of holes so as to minimize the total wire length. This variant arises in VLSI design, graph drawing, modeling nervous activity in the cortex and job scheduling; see [9]. As is the case with many ordering problems, MinLA is NP-hard and the corresponding decision problem is NP-complete [10].

In this paper, we devise a linear-time heuristic algorithm for the MinLA problem. Our experiments with it show its attractiveness in terms of output quality and actual running time. The algorithm embodies a novel multi-scale scheme for the MinLA problem, which we have constructed to fit many linear ordering tasks. Hence, it could have applications beyond the MinLA problem.

We arrived at the ideas in this paper as a result of our interest in graph drawing (see, e.g., [12]), where one seeks an aesthetically pleasing one-dimensional drawing of a graph. Common aesthetic criteria in graph drawing include keeping edges short, preventing vertices from being too close to each other and spreading the vertices uniformly in the

drawing area. The MinLA problem can be viewed as a way to quantify these criteria. It strives to make edge lengths short, while constraining the n vertices to be placed in the integral locations $1, \ldots, n$ (which takes care of the other two criteria). Our MinLA algorithm has been successfully incorporated into a di-graph drawing algorithm [8]. For example see Fig. 1, showing a full binary tree. The x-coordinates of the nodes are determined by their position in the MinLA, while y-coordinates reflect hierarchical precedence using different algorithm (see [8]).

Another visualization application is the ordering of rows and columns of matrices to better understand their structure. For example, we present in Fig. 2(a) an odors similarity matrix generated from 300 measurements taken by an electronic nose. The matrix does not show any structure. In Fig. 2(b) we present the same matrix, after permuting its rows and columns, obtaining a linear arrangement where similar rows (and columns) are put close together. The resulting figure reveals a structure that lies inside the data, where clusters of highly similar odors are expressed as bright rectangles around the main diagonal.

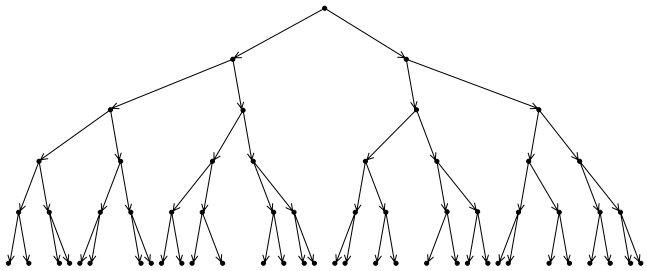

Fig. 1. A depth 5 full binary tree. The x-coordinates are node locations in the minimum linear arrangement.

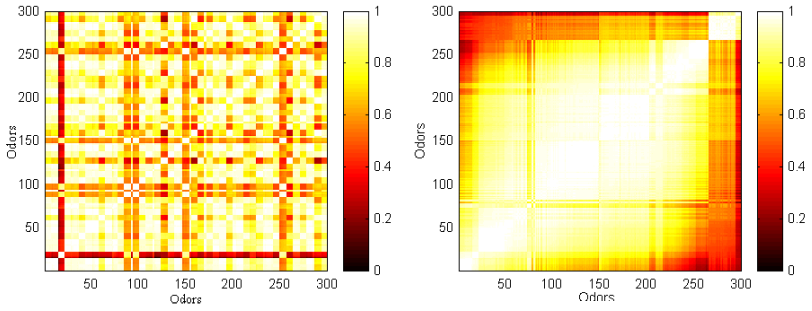

Fig. 2. (a) A similarity matrix of 300 odor patterns as measured by an electronic nose; more similar patterns get higher (=brighter) similarity values. (b) The structure of the data is much better visualized after re-ordering rows and columns of the matrix.

2 The Minimum Linear Arrangement Problem

Let $G(V, E)$ be a graph, where $V = \{1 \dots n\}$ is the set of n vertices, and E is the set of weighted edges, with w_{ij} being the weight of edge $\langle i, j\rangle \in E$. When the context is clear, we will write simply $\langle i, j\rangle$ instead of $\langle i, j\rangle \in E$. Denote the neighborhood of i by $N(i) = \{j \mid \langle i, j\rangle \in E\}$. Throughout the paper we assume, without loss of generality, that G is connected, otherwise the problem we deal with can be solved independently for each connected component.

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

Definition 22 (Arrangement cost) *The* cost *of the arrangement π is defined to be:*

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

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

2.1 Algorithms

For certain classes of "well-structured" graphs, such as trees, grids and hyper-cubes, there exist polynomial time algorithms for solving the MinLA problem. However, the general problem is NP-hard. Thus most research involves approximation and heuristic algorithms. Before proposing one such algorithm, we want to describe an exact algorithm, which is better than the naive $\Theta(n!)$ algorithm that scans all possible permutations, and which will serve us later on.

Dynamic programming 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 the minimum linear arrangement problem possesses a "locality property", to the effect that the best ordering of a set of vertices S that placed to the right of the vertices in the set L and to the left of those in R, is independent of the internal ordering of L and R.

We formalize this fact in the following:

Proposition 1. *Let $G(V, E)$ be a graph and $L, S \subset V$, $L \cap S = \phi$ where $|L| = l$, $|S| = s$. Let π, φ be two permutations of V, such that $\pi(L) = \varphi(L) = \{1, \dots, l\}$, $\pi(S) = \varphi(S) = \{l+1, \dots, l+s\}$. 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 $i \notin S$, $\pi(i) = \pi^*_S(i)$). Now, 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 the vertices in S.

For the proof we will be using the following definition and lemma:

Definition 23 (Local arrangement cost) *Let $G(V, E)$ be a graph, and let L, S and R be mutually disjoint sets with $L \cup S \cup R = V$. 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 $LA^S_\pi(G)$, 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, L, S, R and π be defined as in Def. 23. 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, the desired equality follows. □

Now we are ready to prove Proposition 1:

Proof. Define $R = V - (L \cup S)$, to be the set of vertices placed to the right of S. Consider a permutation φ_S, which for every $i \notin S$ satisfies $\varphi_S(i) = \varphi(i)$. By Lemma 1, the cost associated with 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 the places of vertices in S, is equivalent to finding a permutation φ_S that minimizes $LA^S_{\varphi_S}(G)$.

Similarly, we can write:

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

Now, recall that π^*_S minimizes the arrangement cost over all permutations that differ from π only in the places of vertices in 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 for which 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 the 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, amongst 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 minimal in a global sense, and there is no need to remember the rest of the orderings.

This is the heart of the dynamic programming algorithm, depicted in Fig. 3. For each of the 2^n subsets of V it stores the best ordering found. 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. Its space complexity is $O(2^n)$.

We have not seen this algorithm mentioned in papers dealing with the MinLA problem, probably due to its impractical time and space complexity. However, it 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, 13]. We review here the basic idea of these methods:

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

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

Let v be the normalized eigenvector of L corresponding to the second smallest eigenvalue. Hall [11] has shown that v is the best non-trivial minimum of the following quadratic energy:

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

In fact, the vector v — called the Fiedler vector — is of fundamental importance in many fields, as it reflects connectivity properties of the graph; see, e.g., [18].

Sequencing the vertices is done by sorting them according to their components in v. Fortunately, computation of v can be carried out very rapidly using the multi-scale (or "multi-level") methods of [4, 16].

```
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 do
    table[S].cost ← ∞
  end for
  table[φ].cost ← 0
  table[φ].cut ← 0

  % Fill table:
  for i = 1 to n do
    for every S ⊂ V, |S| = i − 1 do
      cut^S ← table[S].cut
      new_cost ← table[S].cost + cut^S
      for every j ∉ S do
        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 do
    v ← table[S].right_vtx
    ordering[i] ← v
    S ← S − {v}
  end for
end
```

Fig. 3. A dynamic programming algorithm for MinLA

Simulated annealing Simulated annealing [15] is a powerful, general (if slow) optimization technique that is appropriate for the MinLA problem. It repeatedly modifies 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 adopted. Otherwise, $\hat{\pi}$ may still be adopted, but with a probability that decreases as the process proceeds. Adopting a new ordering that worsens the current situation is called *an uphill move*. Uphill moves make it possible to escape bad local minima, thus extending the search space.

Petit [19] has conducted extensive tests of many algorithms for the MinLA problem, on several classes of graphs. We quote his conclusion:

> "...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." [19]

Our algorithm, described in the following sections, produces results whose quality is similar to that of simulated annealing, but its running time is much better.

3 The Median Iteration

We now describe the *median iteration*, a rapid randomized algorithm for decreasing the cost of a linear arrangement. The heart of the 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 $p : V \longrightarrow \mathbb{R}$, and its *cost* is:

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

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

It is obvious that $LP_p(G)$ is minimal when all the vertices are placed at the same location. Hence, the minimal linear placement problem is not interesting. What will be useful for us is a certain process of minimizing $LP_p(G)$, which we shall use to improve linear arrangements.

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 $med_p^G(i)$.*
Since $med_p^G(i)$ is a usual median, it satisfies:

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

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

When equality holds in these two inequalities, the median can be any point inside an open interval, in which case we take $med_p^G(i)$ to be the middle point of this interval.

The main observation is the following:

Proposition 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, $\epsilon \geq 0$, and let $\delta > 0$. Observe now 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. □

We can now 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 $i \in V$ **do**
      $p(i) \leftarrow med_p^G(i)$

  % When a valid linear arrangement is needed:
  Sort nodes according to respected entries in p
  for every $i \in V$ **do**
    $p(i) \leftarrow \langle$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 at a single point, leaving us very little information. But, for relatively small k, we will get many small clusters of vertices placed together. This provides important information about 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 are placed at the same point is random.

We call this method *median iteration*. Since the median can be computed in linear time, 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 we take k to be fixed (a typical value would be $k = 50$), the total time complexity is $O(|E| + n \log n)$. When the initial p is a linear arrangement, so that 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 $O(|E|)$ running time.

The median iteration is a simple way to reduce the cost of linear arrangement. It addresses the global structure of the ordering, while making local decisions randomly. This kind of distinction between local and global characteristics is quite reasonable in the context of the MinLA problem, due to the locality property (Proposition 1), since we can perform later local refinements without increasing the cost of the arrangement.

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

Interestingly, by differentiating Hall's energy with respect to v_i, we obtain a version of Proposition 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 position of its neighbors, whereas to minimize the cost of a linear arrangement, it would be better to place each vertex in the median position of its neighbors. Experiments with the median iteration, as described in Section 5, validate its ability to improve upon spectral sequencing.

4 The 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., [20, 5–7].

Multi-scale techniques transform a high-dimensional problem in an iterative fashion into ones of increasingly lower dimensions,via a process called *coarsening*. On the coarsest scale the problem is solved exactly, following which a *refinement* process starts, whereby 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 must be tailored for the particular problem at hand, so 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., [14]. More recently, Walshaw [21] 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 [12, 16].

4.1 Segment graphs

One of the most prominent requirements of a good multi-scale algorithm is that it keep the inherent structure of the problem unchanged during coarsening. In our case, the form of the MinLA problem, as described in Section 2, will not be preserved during reasonable notions of coarsening, but we can define a more general problem that is preserved during coarsening, and it will contain the original problem as a special case. In fact, this general problem emerges naturally from trying to find an arrangement for a more general entity, that we shall call an *s-graph*, where the *s* stands for *segment*.

Definition 41 (s-graph) *An* s-graph *is a graph $G(V, E)$, whose vertices can be thought of as line segments; a vertex i is associated with a nonnegative real number l_i, denoting its length. Each edge $\langle i, j\rangle$ is associated with two coordinates, ${}_P\langle i, j\rangle$ and $\langle i, j\rangle_P$, denoting the positions of its endpoints inside vertices i and j, respectively. Clearly we have, $0 \leqslant {}_P\langle i, j\rangle \leqslant l_i$, $0 \leqslant \langle i, j\rangle_P \leqslant l_j$.*

Fig. 4 shows an example of an s-graph. This is a 3-clique (a triangle), so the vertex set is $\{1, 2, 3\}$. The vertex lengths are $l_1 = 3$, $l_2 = 4$, $l_3 = 2$. Weights of the edges are $w_{13} = 2$, $w_{12} = 4$, $w_{23} = 3$. Finally, the coordinates of the edges are ${}_P\langle 1, 3\rangle = 1$, $\langle 1, 3\rangle_P = 0$, ${}_P\langle 1, 2\rangle = 2$, $\langle 1, 2\rangle_P = 1$, ${}_P\langle 2, 3\rangle = 2$, $\langle 2, 3\rangle_P = 1$.

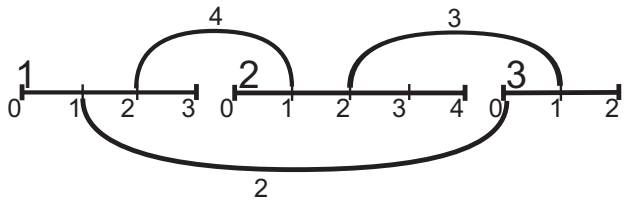

Fig. 4. An s-graph

In a linear arrangement of an s-graph, we place the vertices on a line, in such a way that no two of them intersect. Since we seek an arrangement with short edges, we can safely rule out unused gaps between consecutive vertices. (Though, in the figures we have placed small gaps between consecutive vertices, to make the drawings clearer.) 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 it satisfies $p^\pi_G(i) = \sum_{j,\pi(j)<\pi(i)} l_j$. We will often omit the G in $p^\pi_G(i)$.*

In Fig. 4 we are showing a linear arrangement for which the order of vertices is determined by the bijection: $\pi(1) = 1$, $\pi(2) = 2$, $\pi(3) = 3$. The exact locations of the vertices are: $p^\pi(1) = 0$, $p^\pi(2) = 3$, $p^\pi(3) = 7$.

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 to be:*

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

We will often omit the G in $len^\pi_G(\langle i, j\rangle)$.

Thus, in Fig. 4, $len^{\pi}(\langle 1,3\rangle) = 6$, $len^{\pi}(\langle 1,2\rangle) = 2$, $len^{\pi}(\langle 2,3\rangle) = 3$.

We now generalize the notions of the cost and local-cost of a linear arrangement (see Defs. 22 and 23), to handle s-graphs:

Definition 44 *Given an s-graph G, the* cost *of the linear arrangement π is defined to be :*

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

Definition 45 *Let $G(V,E)$ be an s-graph, $L \cup S \cup R = V$, where L, S and R are mutually disjoint, and let π be a permutation of V such that $\pi(L) = \{1,\ldots,l\}$, $\pi(S) = \{l+1,\ldots,l+s\}$, and $\pi(R) = \{l+s+1,\ldots,n\}$. Denote by a and 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, is defined to be:*

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

Given a regular graph G (not an s-graph), if we set the length of each vertex to be 1 and for every $\langle i,j\rangle$ we take ${}_P\langle i,j\rangle = \langle i,j\rangle_P = 1$, the definitions of the generalized cost and local-cost of a linear arrangement coincide with the regular case (Defs. 22 and 23). Hence, the newer definitions generalize the older ones, and we can keep using the same notations. Moreover, it is easy to verify that Proposition 1 and Lemma 1 hold for s-graphs too, with no modification.

The dynamic programming algorithm can be applied to s-graphs, with slight modifications. We are really computing the best local arrangement cost of a subset $S \subset V$ positioned in locations $\{1,\ldots,|S|\}$. What is needed is a simple change, causing the algorithm to compute the local arrangement according to Def. 45, instead of Def. 23. 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 to be $d_{ij} = p(j) + \langle i,j\rangle_P - p(i) -_P \langle i,j\rangle$. Adding the weighted median of the directed distances between i and its neighbors to the location of i (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

Let G be an s-graph with n nodes. A single coarsening step would be to replace G with a smaller 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, so that in some crucial way both will describe approximately the same entity, but on different scales. Moreover, a good linear arrangement of G^R should correspond to a good linear arrangement of G.

Our attitude to coarsening is to place restrictions on the possible arrangements. These will reduce the search space, so that the restricted problem will be of a "lower dimension", and will involve a smaller graph. We will require that certain pairs of vertices be placed adjacently, in the hope that they will end up being fairly close

in the minimal arrangement of G. In this optimistic situation there is a solution in the restricted subspace that is quite close to the optimal solution, up to some local refinement that does not change the global structure of the arrangement but affects only local portions thereof.

Throughout this section, we illustrate the coarsening process using the graph G, shown in Fig. 5. All its edges have weight 1.

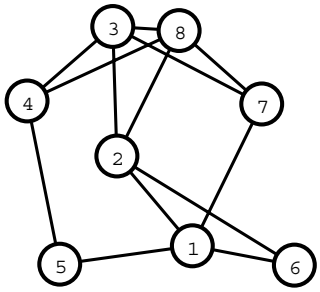

Fig. 5. The graph G

Restricting the search space We would like to be able to restrict the search space in a way that satisfies two conflicting goals:

1. The degrees of freedom should be significantly reduced, enabling representation by a much smaller graph.
2. The minimal arrangement should be affected only locally, while preserving its global structure. Formally, if π^* is the minimal arrangement, we would like an arrangement π to exist, satisfying the restrictions, such that for every vertex i, $|\pi(i) - \pi^*|$ is bounded by some constant.

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

If the initial arrangement π was not too far from the minimal arrangement, the restrictions just introduced affect the solution only locally, as desired.

To activate this method on the graph G of Fig. 5, we have to transform it into an equivalent s-graph (which will also be denoted by G, for convenience), and then find an initial linear arrangement for it. Here, we arbitrarily order the vertices by their names and denote this linear arrangement by π. The result is shown in Fig. 6. As the reader can verify, its cost is $LA_\pi(G) = 43$. We should remark that, in general, smarter

orderings, such as that produced by the median iteration, work much better than an arbitrary initialization.

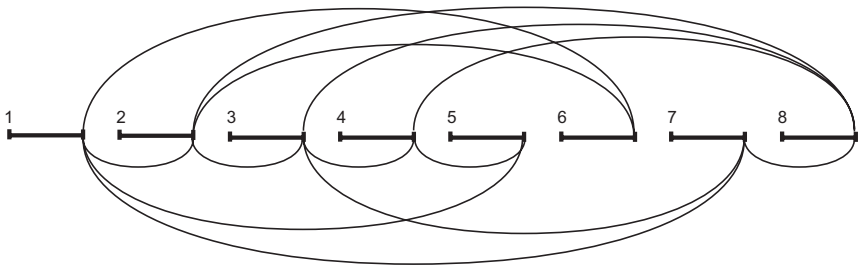

Fig. 6. Linear arrangement of the graph G of Fig. 5. Here we use the equivalent s-graph. Every vertex is of length 1.

As a consequence of the initial linear arrangement π, the set of restricted vertex pairs in the example is $R = \{(1,2),(3,4),(5,6),(7,8)\}$. This restricts every linear arrangement to place vertex 2 immediately after vertex 1, vertex 4 immediately after vertex 3, and so on.

We now show how to formulate the restricted problem as a linear arrangement problem on a much smaller graph.

The coarsening step Given an s-graph $G(V,E)$ and a set $R \subset V \times V$ of restricted vertex pairs, we construct 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.

G^R is produced by contracting pairs of restricted vertices. Vertex lengths and edge weights are preserved, by accumulating them. Positions of the new edges are calculated by averaging those of old edges.

Here is the formal construction of G^R. To ease the technicalities, we will be assuming that if $\langle i,j\rangle \notin E$, then $w_{ij} = 0$ and ${}_P\langle i,j\rangle = \langle i,j\rangle_P = 0$. Also, for simplicity, we assume that n is even, so that 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:

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

- 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 positions of the endpoints of the coarse edge $\langle (v_1, v_2), (v_3, v_4) \rangle$ are the weighted averages of the endpoints of the corresponding fine edges:

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

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

When n is odd, there exists a single unrestricted vertex, v, in which case we add a vertex (v, v) to V^R, where $l_{(v,v)} = l_v$. Definitions regarding the edges adjacent to (v, v) should be slightly modified.

Hence, for our example graph G, as arranged in Fig. 6, we will get the coarse graph G^R shown in Fig 7. We denote the linear arrangement of G^R by φ. The reader can verify that its cost is $LA_\varphi(G^R) = 40$.

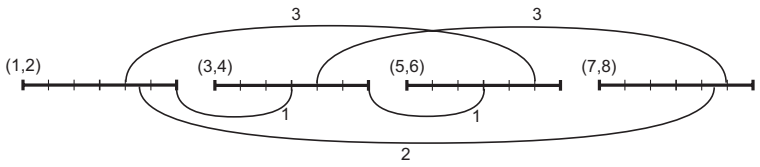

Fig. 7. The graph G^R that is obtained by coarsening the graph of Fig. 5. The length of each vertex is 2, and weights and coordinates of edges were calculated so as to preserve the information of the original graph.

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, and π a restricted linear arrangement of G. Now let φ be a linear arrangement of the respective coarse graph G^R. The two arrangements are* corresponding *if for every $(v_1, v_2) \in R$, we have:*

$$\pi(v_1) = 1 + \sum_{\varphi((i,j)) < \varphi((v_1, v_2))} |(i, j)|$$

where

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

Equivalently, in the inverse direction, 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 $(i, j) \in R$ implies that $i \neq j$, so we can simply write the condition for correspondence as follows: for every $(v_1, v_2) \in R$:

$$\pi(v_1) = 2 * \varphi((v_1, v_2)) - 1$$

Using the close relationship between lengths of vertices in G and G^R, we observe that for two corresponding linear arrangements π and φ, the actual locations of the vertices are related, for each $(v_1, v_2) \in R$, by:

$$p_G^{\pi}(v_1) = p_{G^R}^{\varphi}((v_1, v_2)) \tag{2}$$

It is straightforward to verify in our example that π and φ, are corresponding linear arrangements. Notice that: $p_G^{\pi}(1) = p_{G^R}^{\varphi}((1,2)) = 0$, $p_G^{\pi}(3) = p_{G^R}^{\varphi}((3,4)) = 2$, $p_G^{\pi}(5) = p_{G^R}^{\varphi}((5,6)) = 4$, $p_G^{\pi}(7) = p_{G^R}^{\varphi}((7,8)) = 6$.

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

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

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

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

Lemma 2. *Let G be a graph and R a set of restricted vertex pairs. Let π and φ be 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 A.

In the example graph three edges lie within a restricted pair, and they are $\langle 1, 2\rangle$, $\langle 3, 4\rangle$, $\langle 7, 8\rangle$. Thus, the internal cost is $C(R) = 3$. In agreement with Lemma 2, $LA_{\pi}(G) = LA_{\varphi}(G^R) + C(R) = 40 + 3$.

The coarsening process can help in finding good linear arrangements, and this is demonsrated in Fig. 8. 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 φ^* of G^R, shown in Fig. 8(c). Its cost is $LA_{\varphi^*}(G^R) = 24$.

After solving the problem for the coarse graph, we interpolate the resulting arrangement to the original graph, obtaining the corresponding linear arrangement π^* of G, as shown in Fig. 8(d). In accordance with Lemma 2, $LA_{\pi^*}(G) = LA_{\varphi^*}(G^R) + C(R) = 24 + 3$, a significant decrease in the arrangement cost, which may also be appreciated visually.

We should remark that in the more general case the coarse graph would still be too large to facilitate finding its optimal linear arrangement. Hence, as we shall explain, our strategy is to continue recursively with the coarsening process until we reach a reasonably small graph.

We conclude that given a set R of restricted vertex pairs, 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 that have structure similar to those of the minimum cost arrangements. Hence, we may hope that a good arrangement of G^R corresponds to a reasonably good arrangement of G, and this arrangement can become a truly good one by the local refinement process we now describe.

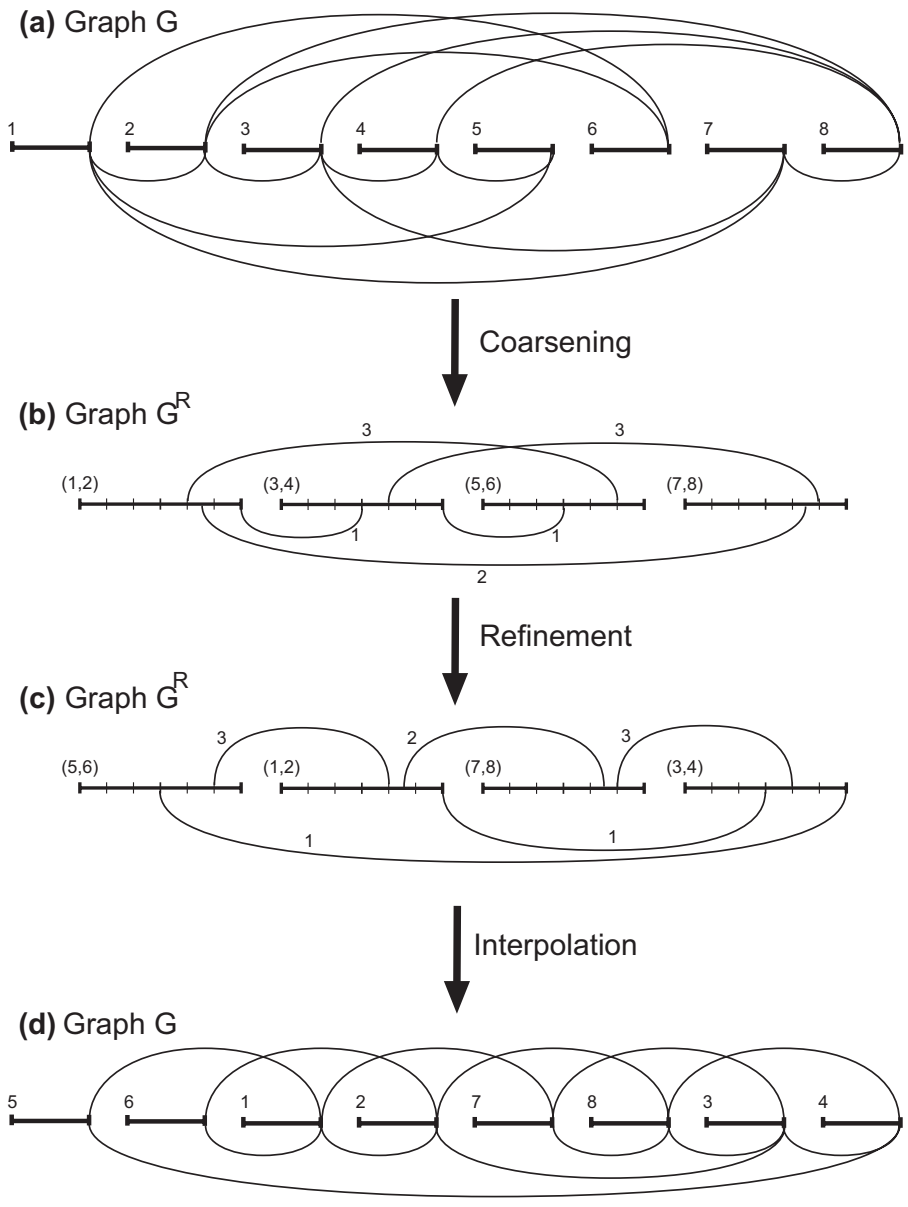

Fig. 8. Improving an initial linear arrangement by optimizing the corresponding arrangement induced on a smaller coarse graph

4.3 Local refinement

Given a linear arrangement π of a graph G, we seek a method that improves the quality of π. Now, in our approach π will typically possess a satisfactory global structure, as a result of minimizing the problem on the restricted search space. Hence, using a method like the median iteration of Section 3 to improve π would not be a good idea, since it is unable to make wise local decisions, which are what is needed here. Consequently, we have constructed a local refinement process that specializes in locally optimizing arrangements.

The idea is to take each k consecutive vertices in π and to rearrange them such that their local arrangement cost (Def. 45) is minimized. Such an optimal arrangement is achieved using a close variant of the dynamic programming algorithm of Section 2, in a time proportional to 2^k. 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. To improve the results the process can be repeated several times.

The local refinement process is depicted in Fig. 9. Taking k to be constant (in our experiments we usually set $k = 6$), the time complexity is $O(|E|)$, and the space requirements are also linear in the size of the input.

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
% $repetitions[= 5]$ – number of iterations of the algorithm
% 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 $repetitions$ **do**
  for $j = 1$ to $n - interval + 1$ **do**
    MinLA_local($G(V, E)$, $Ordering$, j, $j + interval - 1$)
  end for
end for

Fig. 9. The local refinement process

4.4 Putting it all together

We can now put all this together to obtain the multi scale algorithm.

Preprocessing stage Prior to running the algorithm we want to obtain, rapidly, a reasonable linear arrangement. To that end, 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 median iteration is superior to a simple greedy initialization followed by the median iteration. However, since we have developed an extremely fast multi-scale implementation of spectral sequencing [16], it was convenient to use it. Still, if the algorithm in [16] is too complicated, the reader may replace it with a greedy vertex-by-vertex algorithm, such as those appearing in [19, 17].

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

In the most optimistic case, the optimal solution on each scale is reachable from the best restricted solution by local optimization, so that we are guaranteed to find it. Our hope is that in a realistic case a pretty good solution is reachable in this fashion, which, of course, depends on the quality of the refinement process and on the restrictions we impose.

Fig. 10 shows the V-cycle algorithm. It uses several functions. The first of these, 'coarsen(G, π, G^R, π^R)', receives an s-graph G and a linear arrangement of its vertices, π, and produces a coarse graph G^R and a linear arrangement its vertices, π^R. Here, π and π^R are corresponding linear arrangements as per Def. 46. The way to construct G^R and π^R is described in Subsection 4.2. The function 'interpolate(G^R, π^R, π)', receives a coarse graph G^R and a linear arrangement π^R, and produces the corresponding linear arrangement of the fine graph, π.

The recursion in the V-cycle continues as long as G is 'large enough', and a good termination condition would be when the graph is small enough to facilitate finding the optimum linear arrangement directly. In all the graphs that we have tested, local refinement was useless after more than five levels of recursion, due to the good global structure of the arrangement. But in any case, the coarsest graphs are so small that the precise point of termination has but a negligible effect on the running time.

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

Iterating the V-cycle The V-cycle can benefit from starting out with a better arrangement. Thus, repeating the V-cycle can improve its results. In fact, this kind of iteration is common in multi-scale algorithms; see [20]. Iterating the V-cycle can only decrease the cost of the arrangement. In fact, our experience with the algorithm is that after a few iterations the process seems to converge, after which improvement is very slow, if at all.

We can obtain better results by carrying out several median iterations (e.g., 10), before entering each new V-cycle. This perturbs the arrangement and provides the next V-cycle with a different starting point, thus 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, median iteration may worsen the arrangement, turning it into one of higher

```
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. 10. The V-cycle (multi-scale) algorithm

cost, which is reminiscent of the uphill moves of simulated annealing. However, median iteration is far better than arbitrary uphill perturbations, as it inherently tends to the global optimum. Thus, it seems that a few sweeps of median iteration can be quite effective, even when they temporarily increase the arrangement cost. We could also adopt another aspect of simulated annealing, by accepting uphill moves with a probability that decreases as the process continues. In practice we gradually reduce the number of sweeps of the median iteration, lessening the effect of uphill perturbations.

The full algorithm appears in Fig. 11.

```
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:
%   k_1[= 40] – no. of sweeps in first median iteration
%   k_2[= 10] – no. of sweeps in the other median iterations
  Spectral_Sequencing(G, Ordering)
  Median_Iteration(G, Ordering, k_1)
  for i = 1 to Iterations do
    Median_Iteration(G, Ordering, k_2)
    V-cycle(G, Ordering)
    Decrease(k_2)
  end for
```

Fig. 11. The full algorithm

5 Results and Related Work

5.1 A case study

We want to demonstrate the benefit of embedding the local refinement process inside the multi-scale scheme. To that end, we have chosen the 10-dimensional hyper-cube, which consist of 1024 vertices and 5120 edges. What we have observed for this example is typical of many graphs we have considered. The optimal linear arrangement of this graph, π^*, can be computed efficiently, and its cost is 523776; see [19].

As the first stage we applied spectral sequencing to the hyper-cube, resulting in an arrangement with cost 659490. We then improved the arrangement using 40 iterations of median iteration. The cost of the resulting arrangement, π^0, 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 improves the initial arrangement rather significantly).

We then tried to improve π^0 in two ways. First, we applied 100 iterations of the Local_Refine function (setting the local constant *repetitions* to 100), and obtained an arrangement with cost 593120. More iterations of Local_Refine did not decrease the cost any further. Second, we applied a single V-cycle to π^0. During this, the Local_Refine function performed only 5 iterations in each of its executions, thus the running time was much faster. The result of this was the optimal linear arrangement π^*, a clear improvement.

This illustrates the effect of embedding the local refinement process in the V-cycle multi-scale scheme, which improves both running time and output quality. In general, embedding a local refinement process in a multi-scale scheme adds global considerations to the refinement process, because local moves on a coarse graph express more global ones on the original graph. Thus, in some sense, the "wisdom" of a multi-scale algorithm can be divided into two parts: One is related to local optimization and it resides in the local optimization process, and the other deals with the global properties of the problem and it resides in the coarsening construction.

5.2 Petit's test suite

We have implemented the algorithm using C++, and use our ACE algorithm [16] for spectral sequencing. The program runs on a 700MHZ Pentium III, under Windows NT. We have tested it on a test suite of graphs compiled by Petit [19], chosen so as to represent several graph families. Table 1 describes the graphs in the set.

Petit computed linear arrangements of these graphs using many algorithms. The best results were obtained by first running spectral sequencing, and then carrying out a refinement 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). We cannot compare his running times with our directly, since the platforms are different. But to get an impression, the running time of SA for the largest graph in Petit's set, the whitaker3, was over 11 hours. We provide the costs computed by SA in Table 1.

For each graph in this set, we ran our multi-scale algorithm (that of Fig. 11) first with a single V-cycle and then with ten (i.e., with $Iterations = 1$ and $Iterations = 10$). The results appear in Table 2, which also provides the results of spectral sequencing and median iteration — the first stage of the algorithm. Recall that this first stage is

randomized; we actually ran it 10 times for each graph and picked the best result. The quality of our results after 10 V-cycle iterations is comparable to that of Petit's SA, but our running time is significantly better. The results may be further improved by executing more iterations and/or by increasing 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 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 median iteration increased the arrangement cost. This happens due to the inability of median iteration to make clever local decisions. Thus, in cases where the arrangement is close to optimal and local improvements are needed, median iteration would be inappropriate.

Graph Name	Size |V|	Size |E|	Degree min/avg./max	Diameter	Cost using 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. The test suit of Petit [19]. For each graph the table shows the cost of the linear arrangement computed by Petit using simulated annealing (SA).

5.3 Experimenting with larger graphs

The linear running time of our method facilitates its application to graphs whose sizes are orders of magnitude larger than those in Petit's test suit. We have used several graphs from finite element discretization, obtained from publicly available collections. The number of edges is between several hundred of thousands to several million. The results appear in Table 3. For these graphs we have executed our program on a dual processor Intel Xeon 1.7GHz. The program is non-parallel and ran on a single processor. As can be seen, the results of the median iteration constitute a significant improvement over spectral sequencing, and running times are comparable. Multi-scale refinement further improves the results, though its running time is much slower. Notice that the running time of the multi-scale refinement is linear in $2^{interval}$ (the constant *interval* is defined in the function Local_Refine (in Fig. 9)), hence changes of this constant have

Graph Name	Spectral Sequencing	Median Iteration	Multi-Scale 1 Iteration	Multi-Scale 10 Iterations	Time of a single V-cycle
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 our algorithm. The table shows the cost of the resulting arrangments and the running time in seconds (separated by a slash), for 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. The rightmost column shows the time for a single V-cycle on its own.

a great influence on the running time. For all graphs, we have used the default value of *interval*, i.e., 6. However, for several graphs the results remain almost the same for smaller values of *interval*. We have applied the multi-scale algorithm with 10 iterative V-cycles (providing also results for 5 V-cycles). For some graphs the results may be further improved by executing more V-cycles.

5.4 Comparison with Bar-Yehuda et al.

Besides Petit's work [19], we would like to discuss another paper that is closely related to our work. This is Bar-Yehuda et al. [2], in which a divide-and-conquer approach to the MinLA problem is presented. Their idea is to divide the vertices into two sets, to recursively arrange each set internally at consecutive locations, and finally to join the two ordered sets, deciding which 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. Each node of the tree gives a degree of freedom as to 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. The authors of [2] propose a dynamic programming algorithm for computing the best possible ordering for a given decomposition tree. They also applied their algorithm iteratively to Petit's test suit, starting each iteration with the result of the previous one. After a few tens of iterations, they obtain high quality results, similar to those of Petit's SA [19] (and of ours).

It is interesting to compare our algorithm with that of [2], since both are hierarchical in spirit: one uses divide-and-conquer (DAC) and the other uses multi-scale (MS). A main difference between MS and DAC can be expressed by the following observation.

Graph Name	Size |V|	|E|	Degree min/max	Spectral Sequencing	Median Iteration	Multi-Scale 5 Iterations	Multi-Scale 10 Iterations	Time of a a single V-cycle
tooth‡	78,136	452,591	3/39	269280342/2.1s	259873995/2s	255465042	254099155	2.1m
ocean‡	143,437	409,593	1/6	158569182/3.8s	148010790/1.7s	141732687	140087494	2.7m
mrngA†	257,000	505,048	2/4	436251618/8.9s	356042268/5.3s	348448986	345708553	4.7m
rotor‡	99,617	662,431	5/125	288471171/4.2s	258975718/3.5s	247583742	245237685	3.3m
598†	110,971	741,934	5/26	383705515/5.6s	358469578/6.6s	340886008	335814037	3.8m
144†	144,649	1,074,393	4/26	865611991/5.27s	800509976/7.92s	772846779	766341851	5.7m
m14b†	214,765	1,679,018	4/40	1120954880/8.8s	1044523506/11.5s	1004606217	994263100	$8\frac{1}{6}$m
mrngB†	1,017,253	2,015,714	2/4	4498133847/ 26.4s	3630765794/23.8s	3558254373	3532246889	19.6m
auto†	448,695	3,314,611	4/37	4805554654/19.4 s	4607058796/29.4s	4501150138	4467232593	20m

† Taken from George Karypis' collection at: `ftp.cs.umn.edu/users/kumar/Graphs`

‡ Taken from Francois Pellegrini's Scotch graph collection, at:
`http://www.labri.u-bordeaux.fr/Equipe/PARADIS/Member/pelegrin/graph`

Table 3. Results of our algorithm on large graphs. Running times are given in seconds(s) or in minutes(m).

In DAC, the nodes are first divided into two groups, which cannot be mixed during the entire process; this is a kind of a global constraint. In contrast, MS does not employ global constraints, except on the coarsest scale, where the vertices are restricted into a small number of sets. Instead, MS imposes many local constraints, restricting small sets of vertices throughout the entire hierarchy. Interestingly, in the heart of the DAC approach of [2] lies a multi-scale algorithm from [14], which is used for building the decomposition tree.

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

6 Discussion

We have presented a multi-scale algorithm for the minimum linear arrangement problem. It produces arrangements on a par with the best known algorithms, but requires significantly less time, allowing it to deal with much larger graphs.

The heart of the algorithm is a novel construction of a hierarchy of coarse graphs, corresponding to increasingly restricted parts of the original problem. A local refinement scheme becomes considerably improved when embedded in the multi-scale construction. In fact, the multi-scale construction is independent of the specific refinement heuristic, and we believe that other heuristics could benefit from being embedded in such a multi-scale scheme. Also, variants of the proposed multi-scale construction can

be used for other vertex ordering problems, such as bandwidth or cutwidth minimization [9].

An interesting property of our multi-scale construction has to do with the way we build coarse graphs. The common approach to coarsening is based solely on the graph's structure. Most frequently such a coarsening is performed by contracting a set of edges, such as those that participate in a maximal matching (see, e.g. [14]). In contrast, our approach to coarsening relies on a specific solution of the problem, on which we impose several restrictions. When we have a globally good approximate solution, our approach has the advantage of utilizing the wisdom of this solution for peforming a coarsening that forces reasonable restrictions. On the other hand, coarsening based on graph structure is most often very local in nature, and may impose globally bad restrictions, like identifying vertices that should be very distant in the optimal solution. Solution-based coarsening is discussed in [21].

The median iteration process we have proposed seems to have value in its own right. It is actually an extremely fast method for decreasing the cost of an arrangement, using a continuous relaxation of the original problem. It was applied successfully to graphs with millions of edges.

Acknowledgement

We would like to thank Chris Walshaw for his useful comments and corrections.

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 S. 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. A. Brandt, "Multilevel Computations: Review and Recent Developments", *Multigrid Methods: Theory, Applications, and Supercomputing (Proc. 3rd Copper Mountain Conference on Multigrid Methods)*, Marcel Dekker, New York, pp. 35–62, 1988.
8. L. Carmel, D. Harel and Y. Koren, "Drawing Directed Graphs Using One-Dimensional Optimization", `www.wisdom.weizmann.ac.il/~liran/dg.pdf`
9. 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. To appear in *ACM Computing Surveys*.
10. M. R. Garey and D. S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

11. K. M. Hall, "An r-dimensional Quadratic Placement Algorithm", *Management Science* **17** (1970), 219-229.
12. D. Harel and Y. Koren, "A Fast Multi-Scale Method for Drawing Graphs Nicely", *Proceedings of Graph Drawing'00* , Lecture Notes in Computer Science, Vol. 1984, Springer Verlag, pp. 183–196, 2000.
13. M. Juvan and B. Mohar, "Optimal Linear Labelings and Eigenvalues of Graphs", *Discrete Applied Math.* **36** (1992), 153–168.
14. 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.
15. S. Kirkpatrick, J. Gelatt and M.P. Vecchi, "Optimization by Simulated Annealing", *Science* **220** (1983), 671-680.
16. 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.
17. A. J. McAllister, "A New Heuristic Algorithm for the Linear Arrangement Problem", Technical Report 99_126a, Faculty of Computer Science, University of New Brunswick, 1999.
18. B. Mohar, "The Laplacian Spectrum of Graphs", *Graph Theory, Combinatorics, and Applications* **2** (1991), 871–898.
19. 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.)
20. S. H. Teng, "Coarsening, Sampling, and Smoothing: Elements of the Multilevel Method", *Algorithms for Parallel Processing*, IMA Volumes in Mathematics and its Applications, Vol. 105, Springer Verlag, pp. 247–276, 1999.
21. C. Walshaw, "Multilevel Refinement for Combinatorial Optimisation Problems", Technical Report 01/IM/73, Comp. Math. Sci., Univ. Greenwich, London, UK, 2001.

A Proof of Lemma 2

We prove the following lemma, which appears in Subsection 4.2

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

For simplicity assume that n, the number of vertices in the fine graph, is even. Thus, $|R| = n/2$ and if $(i,j) \in R$ then $i \neq j$. Otherwise, the proof should be modified accordingly.

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 $len^\pi(\langle i,j\rangle) = p^\pi(j) + \langle i,j\rangle_P - p^\pi(i) -_P \langle i,j\rangle$.

Also, we have $p^\pi(v_2) = p^\pi(v_1) + l_{v_1}$, $p^\pi(v_4) = p^\pi(v_3) + l_{v_3}$.
Now, rewrite T as follows:

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

As observed in Equation 2 in Subsection 4.2, $p^\pi(v_1) = p^\varphi((v_1, v_2))$ and $p^\pi(v_3) = p^\varphi((v_3, v_4))$. By the construction of G^R, we deduce:

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

Hence,

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

□