A NEW HEURISTIC ALGORITHM FOR THE LINEAR ARRANGEMENT PROBLEM

Andrew J. McAllister

Faculty of Computer Science
University of New Brunswick
Fredericton, New Brunswick
Canada, E3B 5A3
tel: (506) 452-6328
fax: (506) 453-3566
email: AndrewM@unb.ca

Abstract. A new algorithm for the linear arrangement problem is described. The goal is to produce linear arrangements of software model diagrams such that the total length of all connections is reduced as much as possible. The algorithm uses the same general numbering strategy as existing algorithms for the highly related problems of bandwidth and profile reduction but is based on a new heuristic that addresses the unique requirements of the linear arrangement problem. Extensive testing is performed with graphs derived from software model diagrams and from structural engineering. The testing indicates that three refinements to the new algorithm improve the arrangements produced. The new algorithm produces linear arrangements with lower total weighted edge length for both classes of test graphs in comparison with several bandwidth and profile reduction algorithms, and for the software model diagrams in comparison with an eigenvalue-based linear arrangement algorithm. The new heuristic is also shown to require slightly less execution time than the frontal increase minimization heuristic used by several bandwidth and profile reduction algorithms, and far less execution time than the eigenvalue-based algorithm.

Key words. graph, linear arrangement, bandwidth, profile, diagram layout, algorithm

AMS subject classifications. 05C78, 05C85, 68R10

1. Introduction. Let G be a graph with node set V of size n and edge set E of size m. Let $c(uv)$ be an integer greater than zero denoting the weight on the edge uv from node u to node v. Any edge with unspecified weight is assumed to have a weight of 1. A *linear arrangement* (or simply *arrangement*) of G is a one-to-one function f from V to the integer set $\{1,2,...,n\}$. Intuitively, a linear arrangement orders the nodes of G in a straight line with unit distance between nodes in adjacent positions. Let $w(uv, f)$ denote the *length* of an edge uv with respect to arrangement f, defined as $w(uv, f) = |f(u)-f(v)|$ where $f(v)$ denotes the *position* of v in the arrangement. The *weighted length* of the same edge is defined by $c(uv)w(uv, f)$. Let $t(G, f)$ denote the total weighted edge length for G with respect to arrangement f, defined by the following:

$$t(G, f) = \sum_{uv \in E} c(uv)w(uv, f)$$

The *linear arrangement problem* (LAP) is to find f for a given G so that $t(G, f)$ is minimized.

The LAP arises in a number of application areas, including error-correcting codes [12] and VLSI layout [14, 21]. Chung describes the LAP in terms of pins connected with various numbers of wires [3]. In this context the problem is to arrange the pins in a straight line to minimize total wire length.

The motivation for the current research arises in connection with an alternative style for laying out software diagrams, including for example entity-relationship models [2] and data flow diagrams [6]. Such diagrams involve objects connected with lines. Thus drawing a diagram is analogous to drawing a general graph. Traditional layout involves placement of objects (i.e. nodes) anywhere in two dimensions as in Figure 1. Automatic layout in this context involves considerable complexity (see for example [16]). One potential alternative is to arrange the objects in a linear fashion as in Figure 2. Connections (i.e. edges) between objects are drawn between the vertical "tails" associated with the objects rather than between the objects themselves. Layout is

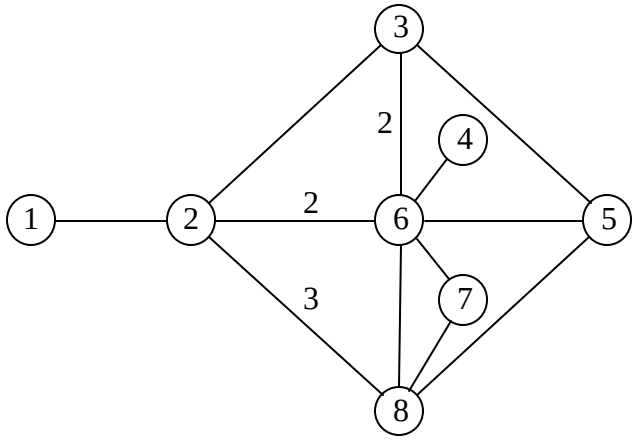

FIG. 1. *An example graph with two-dimensional layout.*

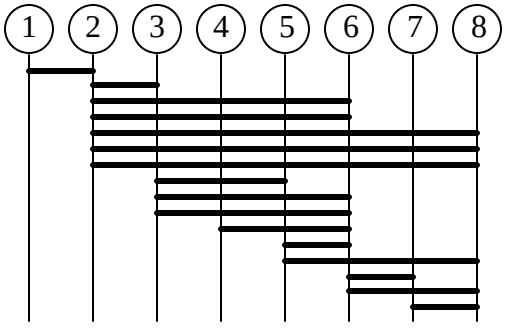

FIG. 2(a). *Nodes arranged in a linear fashion.*

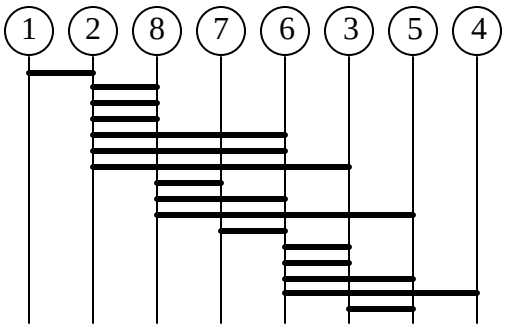

FIG. 2(b). *An alternative linear arrangement.*

simplified in the sense that it is conceptually simple to add or delete any object or connection regardless of the size and complexity of the diagram. Multiple connections between two objects can be shown separately as in Figure 2 or by specifying edge weights as in Figure 1, regardless of the layout style used.

The LAP for this alternative layout style involves ordering the objects to reduce $t(G, f)$ as much as possible. Figure 2(b) provides one possible alternative to the ordering in Figure 2(a) that reduces $t(G, f)$ from 46 to 30. Reducing $t(G, f)$ is intended to enhance the presentation of a diagram by (a) placing highly related objects in close proximity to one another and (b) reducing "visual clutter" associated with long connecting lines. While these diagram presentation issues provide motivation, this paper focuses only on the LAP.

Harper [12] provides a solution to the LAP for the class of n-cubes. Solutions are also available for trees [1, 3, 17]. For general graphs, however, the LAP is known to be NP-hard [7]. There is a need for heuristic algorithms to address this problem in polynomial time.

An eigenvalue-based algorithm is available for finding linear arrangements of VLSI components (e.g. logic gates) to reduce the total length of the wires connecting the components [14, 21]. This problem can be represented as a graph where each node represents a VLSI component and each edge uv represents the wire(s) connecting components u and v. $c(uv)$ represents the number of wires between components u and v. $c(uv)$ is considered to be zero if $uv \notin E$. An arrangement is generated by first creating a square matrix B. Each element b_{ij} of B is defined by:

$$b_{ij} = {}_{ij}\left(\sum_{ik \in E} c(ik)\right) - c(ij)$$

where ${}_{ij}$ is 1 if $i = j$, 0 otherwise. The smallest non-trivial eigenvalue of B is found and the corresponding eigenvector elements are sorted. The indices of the sorted elements are assigned as the resultant f.

Several heuristic algorithms are available for the related problems of *bandwidth reduction* and *profile reduction*. The bandwidth of a graph with respect to a given arrangement is the length of the longest edge. The goal of bandwidth reduction is to find an arrangement with the lowest possible bandwidth. The bandwidths of Figures 2(a) and 2(b) are 6 and 4, respectively. Assume G is stored in the form of an $n \times n$ matrix M where $m_{ij} \neq 0$ iff $v_i v_j \in E$. Storage requirements are reduced and many matrix operations can be performed more efficiently if M is in small bandwidth form.

Let $p(v, f)$ denote the *profile* of node v with respect to arrangement f, defined by the following:

$$p(v, f) = \max(0, (\max_{uv \in E} f(v) - f(u)))$$

Intuitively, $p(v, f)$ is the length of the longest edge extending leftward from v, or zero if no such edge exists. Let $p(G, f)$ denote the profile of G with respect to arrangement f, defined by the following:

$$p(G, f) = \sum_{v \in V} p(v, f)$$

The goal of profile reduction is to find an arrangement that reduces $p(G, f)$ as much as possible. The $p(G, f)$ values for Figures 2(a) and 2(b) are 18 and 17, respectively. Reversing the arrangement in Figure 2(b) reduces $p(G, f)$ to 14.

The primary application of profile reduction is to aid in solution of large linear systems of the form $Mx = b$, where the n n matrix M corresponding to G is sparse and structurally symmetric (m_{ij} 0 iff m_{ji} 0). The solution of a linear system tends to be more efficient when the rows and columns of M are ordered so that the profile of the associated G is reduced.

A number of heuristic algorithms for bandwidth and profile reduction are available. The reverse Cuthill-McKee (RCM) algorithm, a modification by George [8] of the algorithm defined by Cuthill and McKee [4], is perhaps the most widely used. This algorithm was originally developed for bandwidth reduction. The Gibbs-Poole-Stockmeyer (GPS) algorithm [10] was developed to address both bandwidth and profile reduction, whereas profile reduction is the primary goal of the Gibbs-King (GK) [9] and Snay [18] algorithms. These algorithms share the same simple general strategy, as follows:

Step 1. Select a starting node and place this node in position 1.

Step 2. For each remaining position 2 through n, select one of the unplaced nodes for placement in the current position.

The algorithms differ in the methods used for selecting a starting node and in the criteria used for selecting nodes in Step 2.

While bandwidth and profile reduction are similar to the LAP, there are two very significant differences. First, bandwidth and profile are measured using only a subset of edges whereas the LAP is concerned with all edge lengths. Second, edge weights are irrelevant for bandwidth and profile and are ignored by the algorithms mentioned above. Edge weights are, however, important for the LAP.

Bandwidth and profile reduction algorithms tend to be efficient in the sense that several have been implemented to have linear time complexity with respect to m [15]. Thus these algorithms offer a potential advantage over the comparatively expensive calculation of eigenvalues and eigenvectors. This paper describes a new algorithm that uses the same general strategy as the bandwidth and profile reduction algorithms but is based on a new heuristic that addresses the somewhat different requirements of the LAP. The objective is to provide performance similar to the eigenvalue-based heuristic in terms of reducing $t(G, f)$ and similar to the bandwidth and profile reduction algorithms in terms of execution time.

The rest of the paper is organized as follows. Section 2 provides the rationale for and definition of a new heuristic. The implementation and time complexity of an algorithm based on the new heuristic are presented in Section 3. Several refinements to the new algorithm are proposed in Section

4. Section 5 provides experimental results to assess the practical value of the proposed refinements and to compare the new algorithm with the existing algorithms described above. The results are compared in terms of $t(G, f)$ and execution time. Section 6 presents brief conclusions.

2. A new heuristic. One of the most commonly used heuristics for bandwidth and profile reduction is referred to as *frontal increase minimization* (FIM). This section presents the rationale for a new heuristic by first describing the FIM approach and then offering insights into the ordering process based on examples.

Assume the two-step general strategy described in Section 1 is used. A variety of methods can be used for selecting a starting node in Step 1. For now assume the use of a simple strategy suggested by Cuthill and McKee [4], namely to select a node with minimum degree. The *degree* of node v is denoted as $d(v)$, defined as the number of nodes adjacent to v. Node u is *adjacent* to v iff uvE. Other alternatives for starting node selection are considered in Section 4.

FIM is one possible strategy for node placement in Step 2. Assume $2 \le i \le n$ and the number of placed nodes is currently i-1. This means i-1 placements have occurred and placement i is about to occur in position i. Let P_i denote the set of i-1 nodes placed so far and let $U_i = (V-P_i)$ denote the set of currently unplaced nodes. Let $F_i = \{uU_i \mid vP_i \text{ and } uvE\}$ denote the *front* at placement i. The FIM strategy is to select for placement i a node that minimizes the size of F_{i+1}. In other words, select a node that is adjacent to the fewest nodes in U_i-F_i.

Several profile reduction algorithms incorporate the FIM strategy. They differ in terms of the subset of U_i considered for each placement. For example the Snay algorithm selects the node for placement i from all unplaced nodes in or adjacent to F_i. A similar FIM variation is discussed by Marro [15] where the node for placement i is selected from the nodes in F_i. The GK algorithm uses FIM in conjunction with a precalculated *level structure* [9]. A level structure partitions V into levels $L_1, L_2, \ldots, L_k$ where for v in any level, all nodes adjacent to v are at most one level away. For each of L_1 to L_k, all nodes in the current level are placed before moving on to the next level. The GK algorithm determines the order of placement for nodes within a given level using FIM.

Consider applying the FIM strategy to the example graph in Figure 1. Assume node 1 is the starting node and the node for placement i is selected from the nodes in F_i. The resultant ordering is shown in Figure 3. Edges are displayed in the same style as Figure 2. $t(G, f)$ is 39 and $p(G, f)$ is reduced to 13. This example illustrates the following observations:

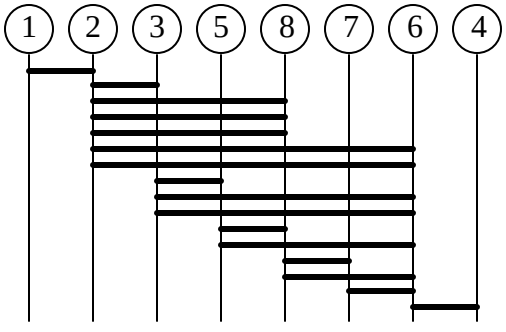

FIG. 3. *An arrangement resulting from FIM.*

Observation 1. Reduction of $p(G, f)$ tends also to reduce $t(G, f)$, however the two goals are not always complementary. Between Figures 2(b) and 3, the former provides the smaller $t(G, f)$ while the latter provides the smaller $p(G, f)$.

Observation 2. The FIM strategy tends to delay placement of a highly connected node until most of its adjacent nodes are placed. For example node 6 is almost the last node to be placed in Figure 3. This makes sense for reducing $p(G, f)$ since only the longest of the edges extending leftward from such a node impacts $p(G, f)$. It seems, however, that a better strategy for reducing $t(G, f)$ is to distribute the nodes adjacent to a highly connected node v as equally as possible on either side of v. This is illustrated by the placement of node 6 in Figure 2(b) but is perhaps better illustrated by the simple example in Figure 4. Minimum $t(G, f)$ is clearly obtained by the arrangement (1, 2, 3, 4, 5). Switching any one of the nodes 1, 2, 4 or 5 to the other side of the highly connected node 3 increases $t(G, f)$.

Observation 3. Edge weight has no impact on $p(G, f)$ and thus is not used by the FIM (or other) profile reduction algorithms. Edge weight is, however, an important factor in determining $t(G, f)$. Consider $F_3 = \{3, 6, 8\}$ during the creation of Figure 3. The FIM strategy selects node 3 for position 3 since this node is connected to only one unplaced non-front node (5) while nodes 6 and 8 are connected to three and two such nodes, respectively. However, since $c_{28} = 3$ is greater than $c_{23} = 1$, it is more important in reducing $t(G, f)$ that node 8 (rather than node 3) should be placed close to node 2 as in Figure 2(b). Similarly, when reducing $t(G, f)$ for Figure 4, nodes 2 and 4 should be preferred over nodes 1 and 5 for placement next to node 3 in the resultant arrangement.

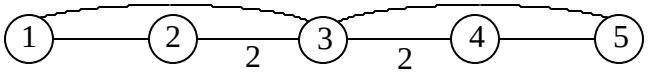

FIG. 4.

Let $d'(v)$ denote the *weighted degree* of v, defined by the following:

$$d'(v) = \sum_{uv \in E} c(uv)$$

For a single vU_i define $tl_i(v)$ as:

$$tl_i(v) = \sum_{u \in P_i,\, uv \in E} c(uv)$$

It follows that $tl_i(v) > 0$ iff vF_i. Define $tr_i(v)$ as $d'(v) - tl_i(v)$. For example in Figure 1 $d(2) = 4$ and $d'(2) = 7$. Assuming node 1 as the starting node, then $tl_2(2) = 1$ and $tr_2(2) = 6$.

Assume $2 \le i \le n$ and vU_i. $tl_i(v)$ and $tr_i(v)$ provide measures of how highly connected v is to P_i and (if v is selected for placement in position i) U_{i+1}, respectively. Based on the three observations above, a method for selecting a node v for each placement is desired so that on average $|tr_i(v) - tl_i(v)| = |d'(v) -$

$2tl_i(v)|$ tends to be as small as possible. This objective is referred to in this paper as *equal distribution*. A new selection factor $sf_i(v)$ is defined to aid in accomplishing this objective, as follows:

$$sf_i(v) = d'(v) - 2tl_i(v)$$

This new selection factor is used with the same two-step general strategy described above. For each placement i in Step 2, select vF_i with minimum $sf_i(v)$. This strategy produces the arrangement in Figure 2(b) for the graph in Figure 1. There are two components to the rationale for using $sf_i(v)$.

(1) As i increases and v remains unplaced, $tl_i(v)$ increases and $tr_i(v)$ decreases with each placement of a node adjacent to v. If $tl_i(v) < tr_i(v)$ (i.e. $sf_i(v) > 0$) then each such placement tends to move the partial f closer to the state where placing v helps to achieve equal distribution. If $tl_i(v) > tr_i(v)$ (i.e. $sf_i(v) < 0$) then each such placement tends to move f further from that state. It follows that (a) the likelihood of placing v should increase as $sf_i(v)$ decreases, and (b) if $sf_i(v) > sf_i(u)$ then u should be preferred over v for placement i.

(2) Let tpu_i denote the total weight of all edges between P_i and U_i, defined as follows:

$$tpu_i = \sum_{v \in F_i} tl_i(v)$$

An alternative definition of $t(G, f)$ for a completed f is:

$$t(G, f) = \sum_{i=2}^{n} tpu_i$$

It follows that $t(G, f)$ is minimized if f is constructed so the average tpu_i is minimized. Furthermore $t(G, f)$ should tend to be *reduced* if the node for each placement i is selected so that tpu_{i+1} is minimized. For placement 1 it is obvious that selecting v with minimum $d'(v)$ as the starting node minimizes tpu_2. If $2in$ and vF_i is selected for placement i then $tpu_{i+1} = tpu_i - tl_i(v) + tr_i(v) = tpu_i + sf_i(v)$. Since tpu_i is a constant for placement i, then tpu_{i+1} is minimized by selecting v with minimum $sf_i(v)$.

3. Implementation and time complexity. Marro [15] shows that FIM implementations with linear time complexity with respect to m are possible based on the following two insights:

(1) For $u \in U_i$, $tl_i(u)$ differs from $tl_{i-1}(u)$ iff v is placed in position i-1 and $u \in ADJ(v)$. Therefore only those $tl_i(u)$ values that change based on placement i-1 need to be adjusted prior to placement i.

(2) It is possible to select a node with minimum $sf_i(u)$ in constant time if several lists of nodes are maintained. Each list contains all nodes $u \in F_i$ that have the same value for $sf_i(u)$. An array of lists $sflist$ is maintained so that the list for any specific $sf_i(u)$ value (denoted $sflist[sf_i(u)]$) can be accessed directly. A separate variable $minsf$ also keeps track of the current minimum $sf_i(u)$ value. Thus a node with minimum $sf_i(u)$ can be selected in constant time by choosing the first node in $sflist[minsf]$.

The Algorithm LA (short for "linear arrangement") defined below results from applying Marro's insights to the new heuristic. Note that $tl_i(u)$ and $sf_i(u)$ values are maintained throughout the

algorithm, rather than calculated for each distinct i value. The expressions $tl(u)$ and $sf(u)$ are used in the algorithm definition to represent their current values.

ALGORITHM LA
Input: A general graph G.
Output: A linear arrangement f of G that approximates the minimum $t(G, f)$.
Method:

1. For each $v \in V$:
   1.1 Calculate $d'(v)$ and $ADJ(v)$.
   1.2 $tl(v) \leftarrow 0$.
2. $F \leftarrow U \leftarrow V$.
3. For i = *Min_Possible_sf_Value* to *Max_Possible_sf_Value*: $sflist[i] \leftarrow$ empty list.
4. $minsf \leftarrow Max\_Possible\_sf\_Value + 1$.
5. Select a starting node $v \in U$. $f_1 \leftarrow v$. $U \leftarrow U - v$.
6. For i = 2 to n:
   6.1 **Update:** For each $u \in (ADJ(v) \cap U)$:
      6.1.1 If $tl(u) > 0$, then: $sf(u) \leftarrow d'(u) - 2tl(u)$, Remove u from $sflist[sf(u)]$.
            Else: $F \leftarrow F - u$.
      6.1.2 $tl(u) \leftarrow tl(u) + c(uv)$. $sf(u) \leftarrow d'(u) - 2tl(u)$.
      6.1.3 Insert u at tail of $sflist[sf(u)]$.
      6.1.4 If $sf(u) < minsf$, then: $minsf \leftarrow sf(u)$.
   6.2 **Select:** $v \leftarrow$ First node in $sflist[minsf]$. Remove v from $sflist[minsf]$.
   6.3 **Place:** $f_i \leftarrow v$. $U \leftarrow U - v$. $F \leftarrow F - v$.
   6.4 If $sflist[minsf]$ is empty, then:
      6.4.1 If $F = \emptyset$, then: $minsf = Max\_Possible\_sf\_Value + 1$.
            Else: Search for the smallest $minsf$ such that $sflist[minsf]$ is non-empty.

Assuming a starting node is selected as any node with minimum $d'(v)$ then Steps 1 to 5 collectively are (m). Since each $v \in V$ is placed in only one position, then $ADJ(v)$ is examined in Step 6.1 only once per ordering for each v. In Step 1.1 each edge uv results in the addition of u to $ADJ(v)$ and v to $ADJ(u)$. It follows that the total number of nodes in all $ADJ(v)$ for $v \in V$ is $2m$. Since each of Steps 6.1.1 to 6.1.4 is (1), then the Update step is (m). Each of Steps 6.2 and 6.3 is (1).

The impact of Step 6.4.1 on the time complexity of the algorithm depends on (a) the number of searches required for G, and (b) the average number of iterations required per search. The number of searches obviously cannot exceed n-1. The example graph pattern in Figure 5(a) shows that the number of searches required can be as small as zero. Following each placement, $F = \emptyset$ is always true in Step 6.4.1. The example graph pattern in Figure 5(b) shows that in the worst case the number of searches for G is (n). Assume the pattern continues to the right for some arbitrarily large number of nodes, with odd numbered nodes in the top row and even numbered nodes in the bottom. Node 1 is the starting node and Algorithm LA places the nodes in ascending order by node number. Assume node $i > 3$ has just been placed in Step 6.3. Prior to beginning Step 6.4 $minsf = 0$, $sflist[minsf]$ is empty, $F = \{i+1\}$, $sf(i+1) = 2$ and $sflist[2] = (i+1)$. Therefore a search occurs in Step 6.4.1 which increases $minsf$ from 0 to 2. Since this occurs for each value of i, the number of searches required is (n).

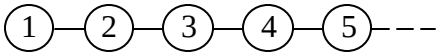

FIG. 5(a). *Number of searches required is zero.*

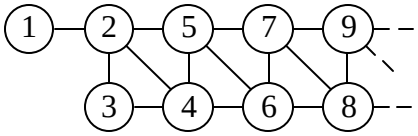

FIG. 5(b). *Number of searches required is O(n).*

Let $d'max(G)$ denote the maximum $d'(v)$ for any $v \in V$. The maximum number of *minsf* values to be searched for any single search is the difference between the largest and smallest possible values for $sf(v)$, which is $2d'max(G)$. Assuming a linear search is used in Step 6.4.1, the worst case time complexity for Step 6.4 is $O(d'max(G)\ n)$ and for Algorithm LA is $O(m + d'max(G)\ n)$. An alternative to a linear search is to maintain the lists in a priority queue, which reduces the worst case time complexity to $O(m + \log_2(d'max(G))\ n)$. A linear search, however, is considered to be a reasonable strategy in Step 6.4.1 for graphs resulting from software model diagrams. The graphs tend to be sparse (i.e. relatively few edges per node) and $c(uv)$ values tend to be small. For example, the largest $c(uv)$ is 3 for any edge in the 20 sample graphs described in Section 5. These two factors tend to keep $|sf(v)|$ values small, which means that if u is placed in Step 6.3 (i.e. $minsf = sf(u)$) and $v \in F$ in Step 6.4 then $sf(v)$-*minsf* tends to be small. The practical effectiveness of a linear search for this step is evaluated with empirical data in Section 5.

4. Refinements. This section introduces four possible refinements to Algorithm LA together with their impact on time complexity. The potential value of the refinements is evaluated in Section 5 based on empirical results. The refinements include a tie breaking strategy, two alternative methods for selection of a starting node and a strategy where the node placed in position i is not always in F_i.

4.1. Tie breaking strategy: It is possible in Step 6.2 of Algorithm LA for there to be several nodes in *sflist*[*minsf*]. Each *sflist* is maintained as a queue where insertions are performed at the tail and removals are from the head of the list. Thus when several nodes have the same minimum value for $sf_i(u)$ the tie is broken by selecting the node for which $sf_i(u)$ has been equal to *minsf* the longest. This is the tie breaking strategy used by Marro [15]. An alternative strategy is suggested by Figure 6. Assume in Figure 6(a) that node 1 has been placed in position i-1 and either 2 or 3 is to be selected for position i. $minsf = sf_i(2) = sf_i(3) = 1$. However, $(tl_i(2)/d'(2) = 1/3) < (tl_i(3)/d'(3) = 2/5) < 0.5$ and thus in this sense node 3 is closer to achieving equal distribution than is node 2. One can argue that node 3 should be selected first. Figure 6(b) depicts a similar situation where node 4 is placed in position i-1 and $minsf < 0$. $minsf = sf_i(5) = sf_i(6) = -1$. In this case $0.5 < (tl_i(6)/d'(6) = 3/5) < (tl_i(5)/d'(5) = 2/3)$ and thus in this sense node 5 is farther past achieving equal distribution than is node 6. One can argue that node 5 should be selected first. The following tie breaking strategy results:

If $minsf > 0$, then:
    v ← Any node in *sflist*[*minsf*] with the largest $d'(v)$.
Else:

v Any node in *sflist*[*minsf*] with the smallest *d'*(*v*).

Modifying Step 6.2 in Algorithm LA to use this strategy necessitates examining the entire *sflist*[*minsf*] for each selection. Marro shows that time complexity is increased by a factor of $\log_2(n)$.

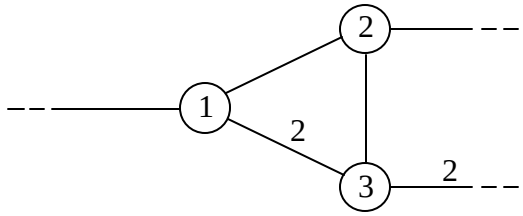

FIG. 6(a). *Tie breaking when minsf > 0.*

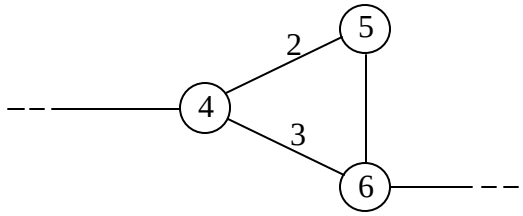

FIG. 6(b). *Tie breaking when minsf < 0.*

4.2 Selection of starting node: The results provided by profile reduction algorithms can vary significantly depending on the choice of starting node [15]. It is reasonable to investigate whether a similar variance is shown by Algorithm LA and, if so, to determine an effective method for selecting a starting node. One of the most widely used methods was first introduced for the GPS algorithm [10]. This method is used directly for other algorithms [9, 15] and is very similar to the method used by Snay [18]. The GPS algorithm finds a *pseudodiameter* of *G*, an end-node of which is used as the start node. The algorithm for finding a pseudodiameter involves creating several rooted level structures. Let *LS*(*v*) denote a level structure rooted at *v*. *LS*(*v*) can be generated as follows:

1. L_1 *v*.
2. For $i > 1$, L_i is the set of all nodes adjacent to any node in L_{i-1} not yet assigned to a level.

The *depth* of a level structure is the number of levels and the *width* is the number of nodes in the largest level. The algorithm for finding a pseudodiameter first generates *LS*(*v*) for *v* with minimal *d*(*v*). Then *LS*(*u*) is generated for each node *u* in the deepest level of *LS*(*v*). If any *LS*(*u*) is found to be deeper than *LS*(*v*), *u* replaces *v* and the process begins again. If no *LS*(*u*) is found to be deeper than *LS*(*v*), then *u* is selected as the node whose associated *LS*(*u*) has the smallest width. The process terminates with *u* and *v* the endpoints of a pseudodiameter. The endpoint with the smallest degree is the starting node.

The generation of a level structure is equivalent to a breadth-first search of *G* and is achieved in (*m*) time. It is difficult to characterize the number of level structures that must be generated to find a pseudodiameter. However, it can be shown that the number of level structures generated has an upper bound of *n* as follows. Assume *LS*(*v*) is generated. For *LS*(*v*) to be generated again, *v* must

be in the deepest level of some $LS(u)$ where $LS(u)$ is deeper than $LS(v)$ and $u \quad v$. The minimum distance between u and v must be greater than the depth of $LS(v)$, which is impossible. Therefore $LS(v)$ is generated only once and the worst case time complexity for finding a pseudodiameter is $O(mn)$.

Turner [20] suggests another method for selecting a starting node: generate f using an arbitrary start node and then generate a second arrangement starting with the rightmost node in f. Probability theory is used to show that this strategy has a high probability of producing an arrangement that is close to optimal when used with existing profile reduction algorithms. This strategy doubles execution time.

4.3 Deferred placement. A potential weakness of Algorithm LA is that f_i is always selected from F_i. In certain cases it is advantageous to select f_i from $(U_i\text{-}F_i)$. This case arises in "star" pattern graphs, such as the example in Figure 7. This pattern is also sometimes referred to as a "claw". Examination of the sample graphs described in Section 5 shows that similar patterns are quite common in graphs derived from software models. Algorithm LA assigns node 4 in Figure 7 to f_2 regardless of which other node is selected as the starting node since $F_2 = \{4\}$. However, as per the discussion in Section 2, the equal distribution objective is best achieved if node 4 is placed closer to the middle of the arrangement. This situation is characterized by selection of a node v for placement in position i where the following three conditions are met:

(1) $tl(v)$ is less than half of $d'(v)$, i.e. $sf(v) > 0$;
(2) v is highly connected, which can be interpreted as $d(v)$ is higher than average; and
(3) At least two nodes adjacent to v remain unplaced.

If Condition (3) is not true, then placing v in a higher position means that v will be positioned to the right of all nodes in $ADJ(v)$. Taken together, these three conditions indicate that *deferring* placement of v is likely better in terms of the equal distribution objective. In other words, if v is selected in Step 6.2 of Algorithm LA and v meets the three conditions above, then v is not assigned to f_i. Instead, remember v as a deferred node. Place subsequent nodes starting in position i as if v had been placed. This means that while a deferred node exists, the ith node selected must be placed in position i-1. When $sf(v)$ becomes zero or less, then v is placed. If a subsequent node u is identified for deferral before v is placed, then place immediately whichever of u or v has the smallest selection factor and retain the other as a deferred node.

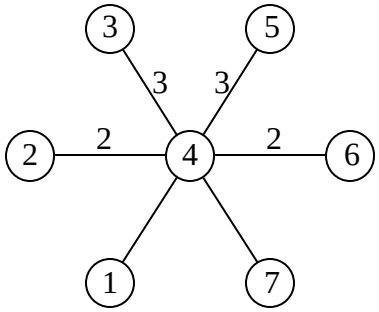

FIG. 7. *A "star" pattern graph.*

The deferred placement strategy is realized by Algorithm DLA (short for "deferred linear arrangement"). In the definition of this algorithm, w denotes a deferred node and $dl(v)$ denotes number of nodes in $ADJ(v)$ that have so far been selected for placement, defined as $dl(v) = |ADJ(v) (V\text{-}U)|$. The expression ceiling($2m/n$) represents the smallest integer not less than the average number of edges per node, and approximates the average $d(v)$.

ALGORITHM DLA
Input: A general graph G.
Output: A linear arrangement f of G that approximates the minimum $t(G, f)$.
Method:

1. For each $v \in V$:
   1.1 Calculate $d'(v)$ and $ADJ(v)$.
   1.2 $tl(v) \leftarrow 0$. $dl(v) \leftarrow 0$.
2. $F \leftarrow \emptyset$. $U \leftarrow V$. $w \leftarrow -1$.
3. For i = *Min_Possible_sf_Value* to *Max_Possible_sf_Value*: *sflist*[i] $\leftarrow$ empty list.
4. *minsf* $\leftarrow$ *Max_Possible_sf_Value* + 1.
5. Select a starting node $v \in U$. $f_1 \leftarrow v$. $U \leftarrow U - v$.
6. For i = 2 to n:
   6.1 **Update deferred node:** If $w \in ADJ(v)$:
      6.1.1 $tl(w) \leftarrow tl(w) + c(wv)$. $sf(w) \leftarrow d'(w) - 2tl(w)$.
      6.1.2 $dl(w) \leftarrow dl(w) + 1$.
   6.2 **Update:** For each $u \in (ADJ(v) \cap U)$:
      6.2.1 If $tl(u) > 0$, then: $sf(u) \leftarrow d'(u) - 2tl(u)$, Remove u from *sflist*[$sf(u)$].
            Else: $F \leftarrow F \cup u$.
      6.2.2 $tl(u) \leftarrow tl(u) + c(uv)$. $sf(u) \leftarrow d'(u) - 2tl(u)$.
      6.2.3 $dl(u) \leftarrow dl(u) + 1$.
      6.2.4 Insert u at tail of *sflist*[$sf(u)$].
      6.2.5 If $sf(u) <$ *minsf*, then: *minsf* $\leftarrow sf(u)$.
   6.3 **Place deferred node:** If $w \neq -1$ and $sf(w) \neq 0$, then: $f_{i-1} \leftarrow w$, $w \leftarrow -1$.
   6.4 **Select:** $v \leftarrow$ First node in *sflist*[*minsf*]. Remove v from *sflist*[*minsf*].

6.5 If $sf(v) > 0$ and $d(v) >$ ceiling$(2m/n)$ and $(d(v)-dl(v)) > 1$, then:
  6.5.1 **Defer placement of *v*:** If $w = -1$, then:
    $w \leftarrow v$.
  Else (there already is a deferred node):
    If $sf(v) < sf(w)$, then: $f_{i-1} \leftarrow v$.
    Else: $f_{i-1} \leftarrow w$, $w \leftarrow v$.
  Else (placement of v is not deferred):
  6.5.2 **Place:** If $w = -1$, then: $f_i \leftarrow v$.
    Else: $f_{i-1} \leftarrow v$.
6.6 $U \leftarrow U - v$. $F \leftarrow F - v$.
6.7 If *sflist*[*minsf*] is empty, then:
  6.7.1 If $F = \emptyset$, then: *minsf* = *Max_Possible_sf_Value* + 1.
    Else: Search for the smallest *minsf* such that *sflist*[*minsf*] is non-empty.

Algorithm LA produces $f = (1, 4, 3, 5, 2, 6, 7)$ with $t(G, f) = 29$ for the graph in Figure 7. Algorithm DLA produces $f = (1, 3, 5, 4, 2, 6, 7)$ with $t(G, f) = 21$, which is close to the optimal result $(1, 2, 3, 4, 5, 6, 7)$ with $t(G, f) = 20$.

The asymptotic time complexity of Algorithm DLA is the same as that of LA. Steps 6.1 and 6.2 of DLA require a single examination of $ADJ(v)$, as does Step 6.1 of LA. Step 6.7 of DLA is the same as step 6.4 of LA. All other subparts of Step 6 in DLA are (1).

5. Empirical results. This section illustrates the performance of the new heuristic algorithm on a collection of empirical test cases. The basic Algorithm LA is compared with the refinements described in Section 4 to determine the practical value of the refinements. Comparisons are made with results obtained from the eigenvalue-based approach to linear arrangement as well as existing bandwidth and profile reduction algorithms.

The motivation for this work is in arranging software model diagrams. To test applicability in this domain, the test cases include 20 graphs derived from sample diagrams. The types of diagrams include entity-relationship diagrams, data flow diagrams and object models. The diagrams were obtained from a variety of sources including textbook examples and documentation from commercial software development projects. The examples range in size and complexity from a single page with 12 graphical objects to models made up of multiple diagrams and including over 100 distinct graphical objects.

The graphs are derived from the software models as follows. Each distinct graphical object (e.g. an "entity" or an "object class" icon) is represented as an arbitrarily numbered node. Each line (e.g. an association or a data flow) drawn between two graphical icons corresponds to an edge. A case where two objects are connected by two or more lines results in an edge with a weight equal to the number of distinct connections. A ternary connection (involving three objects) is represented as a separate edge between each pair of participating objects. This set of 20 graphs has been submitted in matrix form to the Rutherford-Boeing sparse matrix collection [5] so the graphs can be made freely available to other researchers. These 20 test cases are designated as type M in Tables 1 and 2.

Fourteen additional test cases are also included from domains often used for evaluation of profile reduction algorithms. Five cases are from the Cannes collection of finite element structures problems in aircraft design and are designated as type C in Tables 1 and 2. Nine cases are from Everstine's

collection of structural engineering problems, designated as type E in Tables 1 and 2. Inclusion of these cases broadens the types of tests used and provides much larger examples than those derived from software models. These fourteen test cases are available from the Rutherford-Boeing sparse matrix collection [5], are used by Marro to compare the performance of several profile reduction algorithms [15] and are referred to collectively as "structures problems" in the ensuing discussion.

Table 1 provides the average edge length defined as $t(G, f)/m$ resulting from several refinements of Algorithm LA for each test case. Section 4 describes four refinements of LA: a tie breaking strategy, two alternative methods of starting node selection and Algorithm DLA. Table 1 indicates that the latter three refinements provide improved arrangements while the tie breaking strategy appears to be of limited value. DLA improves performance for the software models but not for the structures problems. This is not a surprising result. DLA provides an advantage over LA only when specific types of patterns are present in G, such as star patterns similar to Figure 7. Similar patterns are present in many of the software models but tend not to be present in the structures problems.

Given that three of the refinements provide improved results, it is reasonable to investigate whether combinations of refinements provide further gains. The rightmost two columns in Table 1 show the results provided by the two combinations found to provide the best performance. The first combination involves Algorithm DLA executed twice and uses the rightmost node from the first arrangement as the starting node for the second execution. The second combination extends the first by using a pseudodiameter to select the starting node for the first execution. The results are comparable in that the two combinations return arrangements with exactly the same $t(G, f)$ in 26 of 34 cases and there is only one case (#33) with a marked difference in the result. The second combination provides a 15 percent reduction in total average edge length compared with the basic LA algorithm for the 34 test cases.

Table 2 compares five existing algorithms to a refined version of Algorithm LA, namely the second combination of refinements described above. The FIM algorithm selected for comparison is as similar as possible to the refined version of Algorithm LA. The FIM algorithm is executed twice and a pseudodiameter is used to select the starting node for the first execution. Deferred placement does not apply since FIM does not attempt to achieve equal distribution. The FORTRAN code provided by Lewis and Poole is used for the GPS and GK algorithms [13]. The MATLAB environment from The Mathworks, Inc. is used as the implementation of the RCM algorithm [11]. The eigenvalue-based approach is also implemented using MATLAB, which uses a variant of the Arnoldi method for finding eigenvalues and eigenvectors [19].

For the software models, the new algorithm provides the best arrangement for 17 of the 20 test cases and provides an improvement of approximately 19 percent in overall average edge length over the nearest competitor, the eigenvalue-based approach. The situation is not as simple for the structures problems. The new algorithm provides superior performance compared with the four bandwidth and profile reduction algorithms in that a better arrangement is found in 12 of the 14 cases and the overall average edge length is lower. In comparison with the eigenvalue-based approach, however, the new algorithm is less consistent (e.g. case #27), results in a higher overall average edge length, but still returns a better solution in 5 of 14 cases.

Execution times are provided in Figure 8 for the test cases involving $n > 600$. The trials depicted in Figure 8 were executed on an IBM RS/6000 100 MHz model E20. To increase accuracy each timing involves ten consecutive executions. The total execution time is divided by ten to give the times

shown. The two bottommost data sets in Figure 8 compare the most basic versions of the LA and FIM algorithms, which involve single execution, a minimum degree starting node and no deferred placement. Both algorithms select the node for placement i from F_i. These two algorithms are implemented with the same C++ code, with the exception of the selection factor used for each placement. The additional time required for FIM is due to the updates required after each placement of node v. LA requires examination of each $u \in ADJ(v)$ whereas FIM requires in addition examination of $ADJ(u)$ for each such u.

TABLE 1
Average edge length obtained with various refinements of Algorithm LA

#	*n*	*m*	Type	LA	TieBrk	PD	2	DLA	DLA2	PD+ DLA2
1	12	10	M	3.5	3.5	3.5	3.5	3.0	3.0	2.9
2	16	19	M	4.0	4.0	4.0	3.6	3.0	3.0	3.0
3	17	20	M	2.4	2.3	2.4	2.1	2.4	2.1	2.1
4	19	31	M	6.2	4.3	6.2	6.0	6.0	5.9	5.9
5	19	33	M	3.1	3.1	3.1	3.1	3.2	3.2	3.2
6	20	27	M	3.3	3.3	3.3	3.3	3.4	3.4	3.4
7	21	38	M	10.8	10.8	10.8	8.2	7.7	7.7	7.7
8	23	35	M	5.3	4.9	5.3	4.7	5.2	4.7	4.7
9	29	53	M	6.7	6.5	6.7	5.8	6.7	5.8	5.8
10	30	51	M	5.0	5.0	4.2	4.3	4.9	4.3	4.2
11	32	43	M	4.4	4.3	4.4	4.4	4.3	4.1	4.1
12	33	57	M	4.4	4.4	4.2	4.4	4.4	4.4	4.2
13	47	111	M	10.4	10.7	10.4	10.4	10.4	9.4	9.4
14	55	80	M	6.1	6.0	6.1	5.3	6.0	5.4	5.4
15	56	74	M	6.4	6.5	5.4	5.4	5.7	5.1	5.1
16	66	88	M	4.9	4.9	5.3	4.9	4.6	4.6	4.6
17	72	146	M	8.4	8.4	8.4	8.4	7.9	7.9	7.9
18	74	145	M	10.3	10.3	7.3	7.3	9.7	7.1	7.1
19	93	151	M	14.5	13.4	10.8	11.9	12.6	11.7	10.7
20	119	241	M	8.0	7.8	8.0	8.0	8.0	8.0	8.0
21	634	3297	C	39.4	39.0	39.4	39.4	39.4	39.4	39.4
22	715	2975	C	27.0	26.8	26.5	24.7	26.3	24.7	26.2
23	758	2618	E	10.0	10.0	6.1	6.1	10.0	6.1	6.1
24	838	4586	C	39.1	39.2	37.3	33.9	39.1	33.9	33.9
25	869	3208	E	10.9	10.9	10.9	10.7	10.9	10.7	10.7
26	878	3285	E	14.2	14.2	14.2	14.2	14.2	14.2	14.2
27	918	3233	E	34.3	33.6	34.3	34.3	34.3	34.3	34.3
28	992	7876	E	22.7	22.7	22.7	22.7	22.7	22.7	22.7
29	1005	3808	E	27.9	28.2	27.9	27.9	27.9	27.9	27.9
30	1007	3784	E	16.3	16.3	16.3	14.1	16.3	14.1	14.1
31	1054	5571	C	46.8	46.7	46.6	36.2	46.8	36.1	36.1
32	1072	5686	C	48.9	48.9	35.8	36.6	48.9	36.5	33.4
33	1242	4592	E	40.6	42.8	26.8	35.1	40.6	35.1	26.8
34	2680	11173	E	35.7	37.6	24.6	24.6	35.7	24.6	24.0
	Subtotals:		M	128.1	124.4	119.8	115.0	119.1	110.8	109.4

	C/E	413.8	416.9	369.4	360.5	413.1	360.3	349.8
Totals:		541.9	541.3	489.2	475.5	532.2	471.1	459.2

Type: M = Software Model, C = Cannes collection, E = Everstine's collection

LA: Basic Algorithm LA. TieBrk: LA with tie break strategy.

PD: LA with pseudodiameter used to select starting node. 2: LA executed twice. DLA: Algorithm DLA

DLA2: DLA executed twice. PD+DLA2: Pseudodiameter added to DLA2

TABLE 2
Existing algorithms compared with the new algorithm

#	*n*	*m*	Type	RCM	FIM	GPS	GK	Eigen	LA: PD+ DLA2
1	12	10	M	2.9	4.1	2.9	3.0	3.4	2.9
2	16	19	M	5.1	4.1	4.3	4.1	4.5	3.0
3	17	20	M	3.5	2.9	2.6	2.7	2.4	2.1
4	19	31	M	4.8	5.0	4.7	4.6	4.0	5.9
5	19	33	M	5.1	3.4	4.8	4.5	3.1	3.2
6	20	27	M	5.1	3.7	4.8	4.5	3.3	3.4
7	21	38	M	12.4	11.4	10.2	10.2	10.2	7.7
8	23	35	M	7.3	5.3	7.2	5.3	6.4	4.7
9	29	53	M	7.8	7.8	7.2	6.6	5.8	5.8
10	30	51	M	6.9	4.6	5.3	4.9	4.7	4.2
11	32	43	M	5.9	4.7	4.6	4.8	6.5	4.1
12	33	57	M	7.2	5.8	6.5	5.9	5.2	4.2
13	47	111	M	12.9	14.0	12.6	10.6	12.1	9.4
14	55	80	M	13.4	6.5	10.7	9.4	7.8	5.4
15	56	74	M	11.5	6.9	8.0	7.9	6.0	5.1
16	66	88	M	9.9	6.2	8.3	8.4	7.6	4.6
17	72	146	M	12.9	10.4	10.8	10.3	10.1	7.9
18	74	145	M	11.5	10.4	11.4	13.0	8.4	7.1
19	93	151	M	23.3	15.3	16.8	15.2	14.8	10.7
20	119	241	M	18.8	10.2	15.5	13.8	9.5	8.0
21	634	3297	C	43.7	53.2	39.9	39.7	33.0	39.4
22	715	2975	C	44.5	31.5	41.9	41.6	35.7	26.2
23	758	2618	E	7.8	7.0	7.3	7.3	6.0	6.1
24	838	4586	C	49.9	39.9	46.0	41.9	31.9	33.9
25	869	3208	E	16.8	12.2	13.7	13.4	11.9	10.7
26	878	3285	E	18.8	17.0	16.2	16.1	15.0	14.2
27	918	3233	E	18.4	50.4	16.3	16.0	12.8	34.3
28	992	7876	E	25.8	24.7	23.0	23.0	23.3	22.7
29	1005	3808	E	32.1	47.1	31.1	29.5	22.7	27.9
30	1007	3784	E	17.9	16.1	16.2	16.0	14.3	14.1
31	1054	5571	C	37.2	43.8	33.6	35.1	27.1	36.1
32	1072	5686	C	52.6	39.9	45.3	46.0	29.7	33.4
33	1242	4592	E	32.1	30.5	35.1	32.4	22.4	26.8
34	2680	11173	E	28.7	39.0	26.7	26.4	23.0	24.0
Subtotals:			M	188.2	142.7	159.2	149.7	135.8	109.4

	C/E	426.3	452.3	392.3	384.4	308.8	349.8
Totals:		614.5	595.0	551.5	534.1	444.6	459.2

Algorithms:

RCM:	Reverse Cuthill-McKee.	FIM:	Frontal increase minimization.
GPS:	Gibbs-Poole-Stockmeyer.	GK:	Gibbs-King.
Eigen:	Eigenvalue-based linear arrangement.		
LA:	DLA executed twice, pseudodiameter used for the first starting node.		

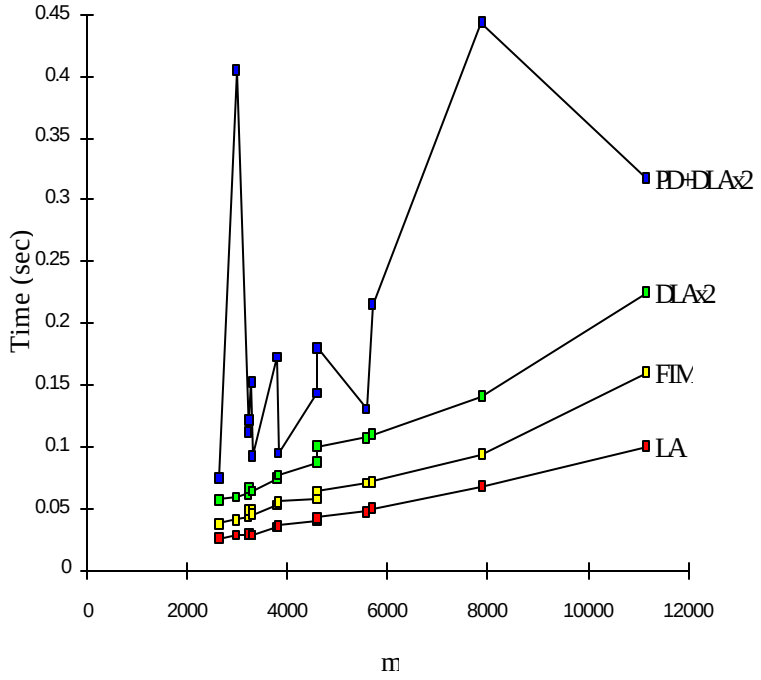

FIG. 8. *Execution times: FIM versus three variations of the new algorithm.*

Both the LA and FIM implementations exhibit growth in execution time suggestive of time complexity that in practical terms approximates (m). Marro [15] provides an analysis that shows FIM to be (m), however this analysis does not mention any step equivalent to Step 6.4.1 in Algorithm LA. This is that only step not shown to be (m) in Section 3. For the 34 test cases, a search is necessary in this step during 57 percent of the placements. On average the number of iterations per search is 1.56. This average seems small enough to justify the use of a linear search in Step 6.4.1, at least for this set of test data. The worst case time complexity of this step is $O(d'max(G)\ n)$. However the average number of iterations required for the test graphs is $0.88n$. Thus it is not surprising that m is a reasonable predictor of execution time for the basic LA and FIM algorithms in Figure 8.

Times for Algorithm DLA executed twice are shown as a third data set in Figure 8. Not surprisingly, the times are approximately twice those for LA. Deferred placement seems to have no effect on asymptotic time complexity, as predicted in Section 4.3.

The final (topmost) data set in Figure 8 shows the effect on execution time of generating a pseudodiameter for selection of a starting node. The differences in execution times between the third (DLA2) and the topmost data sets are due entirely to starting node selection. The time required for generation of the pseudodiameter varies between 18 and 85 percent of total execution time for the 14 cases in Figure 8.

Direct comparison of our implementation of Algorithm LA with the RCM, GPS and GK implementations in terms of execution time is not possible since different hardware and software environments are involved. However, Marro shows that execution times for these three algorithms

tend to be comparable with FIM [15]. In addition, Table 3 lists the execution times for the two algorithms implemented using MATLAB. These trials were executed on an 85 MHz Sun Microsystem SPARC 1000. Even allowing for the difference in execution speed of the two machines involved, the times for the RCM algorithm are comparable with the times depicted in Figure 8 for the FIM, LA and DLA algorithms in the sense that all times are less than one-half second. The total time required for the 14 cases with the eigenvalue-based approach, however, is more than 500 times that required with RCM.

TABLE 3
MATLAB execution times in seconds

#	*n*	*m*	RCM	Eigen
21	634	3297	0.06	21.36
22	715	2975	0.07	22.68
23	758	2618	0.04	20.63
24	838	4586	0.12	29.42
25	869	3208	0.07	26.01
26	878	3285	0.05	29.60
27	918	3233	0.05	33.08
28	992	7876	0.16	46.71
29	1005	3808	0.08	47.05
30	1007	3784	0.10	38.10
31	1054	5571	0.10	47.76
32	1072	5686	0.15	48.19
33	1242	4592	0.08	57.24
34	2680	11173	0.23	236.83
		Totals:	1.36	704.66

6. Conclusions. The results in Section 5 provide strong evidence that the heuristic described in this paper is an improvement over existing profile and bandwidth reduction algorithms for the linear arrangement problem, especially when Algorithm LA is refined as per Section 4. The new heuristic is shown to be slightly more efficient in terms of execution time than the FIM approach. The results also show that the most effective linear arrangement algorithm depends on the type of application. Algorithm DLA refined with two executions per graph and the use of a pseudodiameter for starting node selection provides the best overall results for the software model test cases. If execution speed is not a concern for a given structural engineering application, then the eigenvalue-based approach provides somewhat better overall results. For time-critical applications, however, the new algorithm offers competitive performance in reducing total weighted edge length plus a considerable advantage over the eigenvalue-based approach in terms of execution speed.

Acknowledgements. The author thanks Joe Horton for his helpful comments on the first draft version of this paper.

REFERENCES

[1] D. ADOLPHSON AND T.C. HU, *Optimal linear ordering*, SIAM J. Appl. Math., 25 (1973), pp. 403-423.

[2] P. CHEN, *The entity-relationship model: Toward a unified view of data*, ACM Trans. Database Sys., 1 (1976), pp. 9-36.

[3] F.R.K. CHUNG, *On optimal linear arrangements of trees*, Comput. Math. Appl., 10 (1984), pp. 43-60.

[4] E. CUTHILL AND J. MCKEE, *Reducing the bandwidth of sparse symmetric matrices*, in ACM National Conference Proceedings, 24, 1969, pp. 157-172.

[5] I.S. DUFF, R.G. GRIMES AND J.G. LEWIS, *The Rutherford-Boeing sparse matrix collection*, Technical Report TR/PA/97/36, Online: http://www.cerfacs.fr/algor/algo_reports.html, 1997.

[6] C. GANE AND T. SARSON, *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, Englewood Cliffs, NJ, 1979.

[7] M.R. GAREY, D.S. JOHNSON AND L. STOCKMEYER, *Some simplified NP-complete graph problems*, Theoret. Comput. Sci., 1 (1976), pp. 237-267.

[8] A. GEORGE, *Computer implementation of the finite element method*, STAN-CS-71-208,Computer Science Dept., Stanford Univ.,Stanford, Calif., 1971.

[9] N.E. GIBBS, *Algorithm 509: a hybrid profile reduction algorithm*, ACM Trans. Math. Software, 2 (1976), pp. 378-387.

[10] N.E. GIBBS, W.G. POOLE JR AND P.K. STOCKMEYER, *An algorithm for reducing the bandwidth and profile of a sparse matrix*, SIAM J. Numer. Anal., 13 (1976), pp. 236-250.

[11] J.R. GILBERT, C. MOLER AND R. SCHREIBER, *Sparse matrices in MATLAB: design and implementation*, SIAM J. Mat. Anal., 13 (1992), pp. 333-356.

[12] L.H. HARPER, *Optimal assignment of numbers to nodes*, J. SIAM, 12 (1961), pp. 131-135.

[13] J.G. LEWIS AND W.G. POOLE JR, *Implementation of the Gibbs-Poole-Stockmeyer and Gibbs-King algorithms*, ACM Trans. Math. Software, 8 (1982), pp. 180-189.

[14] W. LIU AND A. VANNELLI, *Generating lower bounds for the linear arrangement problem*, Discrete Appl. Math., 59 (1995), pp. 137-151.

[15] L. MARRO, *A linear time implementation of profile reduction algorithms for sparse matrices*, SIAM J. Sci. Stat. Comp., 7 (1986), pp. 1212-1231.

[16] L.B. PROTSKO, P.G. SORENSON, J.P. TREMBLAY AND D.A. SCHAEFER, *Towards the automatic generation of software diagrams*, IEEE Trans. Soft. Eng., 17 (1991), pp. 10-21.

[17] Y. SHILOACH, *A minimum linear arrangement algorithm for undirected trees*, SIAM J. Comp., 8 (1979), pp. 15-32.

[18] R.A. SNAY, *Reducing the profile of sparse symmetric matrices*, Internat. Bull. Géodésique, 50 (1976), pp. 341-352.

[19] D.C. SORENSON, *Implicit application of polynomial filters in a k-step Arnoldi method*, SIAM J. Mat. Anal., 13 (1992), pp. 357-385.

[20] J.S. TURNER, *On the probable performance of heuristics for bandwidth minimization*, SIAM J. Comp., 15 (1986), pp. 561-580.

[21] A. VANNELLI AND G.S. ROWAN, *An eigenvector-based approach for multi-stack VLSI layout*, in Proc. Midwest Symp. on Circuits and Systems, 29, 1986, pp. 435-439.