Computing an optimal orientation of a balanced decomposition tree for linear arrangement problems

Reuven Bar-Yehuda
Computer Science Dept.
Technion
Haifa, Israel
http://www.cs.technion.ac.il/reuven/
reuven@cs.technion.ac.il

Guy Even
Dept. of Electrical Engineering-Systems
Tel-Aviv University
Tel-Aviv, Israel
http://www.eng.tau.ac.il/guy/
guy@eng.tau.ac.il

Jon Feldman
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA, USA
http://theory.lcs.mit.edu/jonfeld/
jonfeld@theory.lcs.mit.edu

Joseph (Seffi) Naor
Computer Science Dept.
Technion
Haifa, Israel
http://www.cs.technion.ac.il/users/wwwb/cgi-bin/facultynew.cgi?Naor.Joseph
naor@cs.technion.ac.il
Abstract

Divide-and-conquer approximation algorithms for vertex ordering problems partition the vertex set of graphs, compute recursively an ordering of each part, and “glue” the orderings of the parts together. The computed ordering is specified by a decomposition tree that describes the recursive partitioning of the subproblems. At each internal node of the decomposition tree, there is a degree of freedom regarding the order in which the parts are glued together.

Approximation algorithms that use this technique ignore these degrees of freedom, and prove that the cost of every ordering that agrees with the computed decomposition tree is within the range specified by the approximation factor. We address the question of whether an optimal ordering can be efficiently computed among the exponentially many orderings induced by a binary decomposition tree.

We present a polynomial time algorithm for computing an optimal ordering induced by a binary balanced decomposition tree with respect to two problems: Minimum Linear Arrangement (MINLA) and Minimum Cutwidth (MINCW). For 1/3-balanced decomposition trees of bounded degree graphs, the time complexity of our algorithm is $O(n^{2.2})$, where $n$ denotes the number of vertices.

Additionally, we present experimental evidence that computing an optimal orientation of a decomposition tree is useful in practice. It is shown, through an implementation for MINLA, that optimal orientations of decomposition trees can produce arrangements of roughly the same quality as those produced by the best known heuristic, at a fraction of the running time.

Communicated by T. Warnow; submitted July 1998; revised September 2000 and June 2001.

Guy Even was supported in part by Intel Israel LTD and Intel Corp. under a grant awarded in 2000. Jon Feldman did part of this work while visiting Tel-Aviv University.
1 Introduction

The typical setting in vertex ordering problems in graphs is to find a linear ordering of the vertices of a graph that minimizes a certain objective function [GJ79, A1.3, pp. 199-201]. These vertex ordering problems arise in diverse areas such as: VLSI design [HL99], computational biology [K93], scheduling and constraint satisfaction problems [FD], and linear algebra [R70]. Finding an optimal ordering is usually NP-hard and therefore one resorts to polynomial time approximation algorithms.

Divide-and-conquer is a common approach underlying many approximation algorithms for vertex ordering problems [BL84, LR99, Ha89, RAK91, ENRS00, RR98]. Such approximation algorithms partition the vertex set into two or more sets, compute recursively an ordering of each part, and “glue” the orderings of the parts together. The computed ordering is specified by a decomposition tree that describes the recursive partitioning of the subproblems. At each internal node of the decomposition tree, there is a degree of freedom regarding the order in which the parts are glued together. We refer to determining the order of the parts as assigning an orientation to an internal decomposition tree node since, in the binary case, this is equivalent to deciding which child is the left child and which child is the right child. We refer to orderings that can be obtained by assigning orientations to the internal nodes of the decomposition trees as orderings that agree with the decomposition tree.

Approximation algorithms that use this technique ignore these degrees of freedom, and prove that the cost of every ordering that agrees with the computed decomposition tree is within the range specified by the approximation factor. The questions that we address are whether an optimal ordering can be efficiently computed among the orderings that agree with the decomposition tree computed by the divide-and-conquer approximation algorithms, and whether computing this optimal ordering is a useful technique in practice.

Contribution. We present a polynomial time algorithm for computing an optimal orientation of a balanced binary decomposition tree with respect to two problems: Minimum Linear Arrangement (minla) and Minimum Cutwidth (mincw). Loosely speaking, in these problems, the vertex ordering determines the position of the graph vertices along a straight line with fixed distances between adjacent vertices. In minla, the objective is to minimize the sum of the edge lengths, and in mincw, the objective is to minimize the maximum cut between a prefix and a suffix of the vertices.
The corresponding decision problems for these optimization problems are NP-Complete [GJ79, problems GT42,GT44]. For binary 1/3-balanced decomposition trees of bounded degree graphs, the time complexity of our algorithm is \(O(n^{2.2})\), and the space complexity is \(O(n)\), where \(n\) denotes the number of vertices.

This algorithm also lends itself to a simple improvement heuristic for both minla and mincw: Take some ordering \(\pi\), build a balanced decomposition tree from scratch that agrees with \(\pi\), and find its optimal orientation. In the context of heuristics, this search can be viewed as generalizing local searches in which only swapping of pairs of vertices is allowed [P97]. Our search space allows swapping of blocks defined by the global hierarchical decomposition of the vertices. Many local searches lack quality guarantees, whereas our algorithm finds the best ordering in an exponential search space.

The complexity of our algorithm is exponential in the depth of the decomposition tree, and therefore, we phrase our results in terms of balanced decomposition trees. The requirement that the binary decomposition tree be balanced does not incur a significant setback for the following reason. The analysis of divide-and-conquer algorithms, which construct a decomposition tree, attach a cost to the decomposition tree which serves as an upper bound on the cost of all orderings that agree with the decomposition tree. Even et al. [ENRS00] presented a technique for balancing binary decomposition trees. When this balancing technique is applied to minla the cost of the balanced decomposition tree is at most three times the cost of the unbalanced decomposition tree. In the case of the cutwidth problem, this balancing technique can be implemented so that there is an ordering that agrees with the unbalanced and the balanced decomposition trees.

Interestingly, our algorithm can be modified to find a worst solution that agrees with a decomposition tree. We were not able to prove a gap between the best and worst orientations of a decomposition tree, and therefore the approximation factors for these vertex ordering problems has not been improved. However, we were able to give experimental evidence that for a particular set of benchmark graphs the gap between the worst and best orientations is roughly a factor of two.

**Techniques.** Our algorithm can be interpreted as a dynamic programming algorithm. The “table” used by the algorithm has entries \(\langle t, \alpha \rangle\), where \(t\) is a binary decomposition tree node, and \(\alpha\) is a binary string of length \(\text{depth}(t)\), representing the assignments of orientations to the ancestors of \(t\) in the decomposition tree. Note that the size of this table is exponential in the
depth of the decomposition tree. If the decomposition tree has logarithmic
depth (i.e. the tree is balanced), then the size of the table is polynomial.

The contents of a table entry \( \langle t, \alpha \rangle \) is as follows. Let \( M \) denote the set of
leaves of the subtree rooted at \( t \). The vertices in \( M \) constitute a contiguous
block in every ordering that agrees with the decomposition tree. Assigning
orientations to the ancestors of \( t \) implies that we can determine the set \( L \) of
vertices that are placed to the left of \( M \) and the set \( R \) of vertices that are
placed to the right of \( M \). The table entry \( \langle t, \alpha \rangle \) holds the minimum local cost
associated with the block \( M \) subject to the orientations \( \alpha \) of the ancestors of
\( t \). This local cost deals only with edges incident to \( M \) and only with the cost
that these edges incur within the block \( M \). When our algorithm terminates,
the local cost of the root will be the total cost of an optimal orientation,
since \( M \) contains every leaf of the tree.

Our ability to apply dynamic programming relies on a locality property
that enables us to compute a table entry \( \langle t, \alpha \rangle \) based on four other table
entries. Let \( t_1 \) and \( t_2 \) be the children of \( t \) in the decomposition tree, and
let \( \sigma \in \{0, 1\} \) be a possible orientation of \( t \). We show that it is possible to
easily compute \( \langle t, \alpha \rangle \) from the four table entries \( \langle t_i, \alpha \cdot \sigma \rangle \), where \( i \in \{1, 2\} \),
\( \sigma \in \{0, 1\} \).

The table described above can viewed as a structure called an orientations tree
of a decomposition tree. Each internal node \( \hat{t} = \langle t, \alpha \rangle \) of this
orientations tree corresponds to a node \( t \) of the decomposition tree, and a
string \( \alpha \) of the orientations of the ancestors of \( t \) in the decomposition tree.
The children of \( \hat{t} \) are the four children described above, so the value of each
node of the orientations tree is locally computable from the value of its four
children. Thus, we perform a depth-first search of this tree, and to reduce
the space complexity, we do not store the entire tree in memory at once.

Relation to previous work. For minla, Hansen [Ha89] proved that decom-
position trees obtained by recursive \( \alpha \)-approximate separators yields an
\( O(\alpha \cdot \log n) \) approximation algorithm. Since Leighton and Rao presented an
\( O(\log n) \)-approximate separator algorithm, an \( O(\log^2 n) \) approximation fol-

ows. Even et al. [ENRS00] gave an approximation algorithm that achieves
an approximation factor of \( O(\log n \log \log n) \). Rao and Richa [RR98] im-
proved the approximation factor to \( O(\log n) \). Both algorithms rely on com-
puting a spreading metric by solving a linear program with an exponential
number of constraints. For the cutwidth problem, Leighton and Rao [LR99]
achieved an approximation factor of \( O(\log^2 n) \) by recursive separation. The
approximation algorithms of [LR99, Ha89, ENRS00] compute binary 1/3-
balanced decomposition trees so as to achieve the approximation factors.

The algorithm of Rao and Richa [RR98] computes a non-binary non-balanced decomposition tree. Siblings in the decomposition tree computed by the algorithm of Rao and Richa are given a linear order which may be reversed (i.e. only two permutations are allowed for siblings). This means that the set of permutations that agree with such decomposition trees are obtained by determining which “sibling orderings” are reversed and which are not. When the depth of the decomposition tree computed by the algorithm of Rao and Richa is super-logarithmic and the tree is non-binary, we cannot apply the orientation algorithm since the balancing technique of [ENRS00] will create dependencies between the orientations that are assigned to roots of disjoint subtree.

**Empirical Results.** Our empirical results build on the work of Petit [P97]. Petit collected a set of benchmark graphs and experimented with several heuristics. The heuristic that achieved the best results was Simulated Annealing. We conducted four experiments as follows. First, we computed decomposition trees for the benchmark graphs using the HMETIS graph partitioning package [GK98]. Aside from the random graphs in this benchmark set, we showed a gap of roughly a factor of two between worst and best orientations of the computed decomposition trees. This suggests that finding optimal orientations is practically useful. Second, we computed several decomposition trees for each graph by applying HMETIS several times. Since HMETIS is a random algorithm, it computes a different decomposition each time it is invoked. Optimal orientations were computed for each decomposition tree. This experiment showed that our algorithm could be used in conjunction with a partitioning algorithm to compute somewhat costlier solutions than Simulated Annealing at a fraction of the running time. Third, we experimented with the heuristic improvement algorithm suggested by us. We generated a random decomposition tree based on the ordering computed in the second experiment, then found the optimal orientation of this tree. Repeating this process yielded improved the results that were comparable with the results of Petit. The running time was still less that Simulated Annealing. Finally, we used the best ordering computed as an initial solution for Petit’s Simulated Annealing algorithm. As expected, this produced slightly better results than Petit’s results but required more time due to the platform we used.
Organization. In Section 2, we define the problems of minla and mincw as well as decomposition trees and orientations of decomposition trees. In Section 3, we present the orientation algorithm for minla and mincw. In Section 4 we propose a design of the algorithm with linear space complexity and analyze the time complexity of the algorithm. In Section 5 we describe our experimental work.

2 Preliminaries

2.1 The Problems
Consider a graph $G(V,E)$ with non-negative edge capacities $c(e)$. Let $n = |V|$ and $m = |E|$. Let $[i..j]$ denote the set $\{i, i + 1, \ldots, j\}$. A one-to-one function $\pi : V \rightarrow [1..n]$ is called an ordering of the vertex set $V$. We denote the cut between the first $i$ nodes and the rest of the nodes by $\text{cut}_\pi(i)$, formally,

$$\text{cut}_\pi(i) = \{(u, v) \in E : \pi(u) \leq i \text{ and } \pi(v) > i\}.$$  

The capacity of $\text{cut}_\pi(i)$ is denoted by $c(\text{cut}_\pi(i))$. The cutwidth of an ordering $\pi$ is defined by

$$cw(G, \pi) = \max_{i \in [1..n-1]} c(\text{cut}_\pi(i)).$$

The goal in the Minimum Cutwidth Problem (MINCW) is to find an ordering $\pi$ with minimum cutwidth.

The goal in the Minimum Linear Arrangement Problem (MINLA) is to find an ordering that minimizes the weighted sum of the edge lengths. Formally, the the weighted sum of the edge lengths with respect an ordering $\pi$ is defined by:

$$la(G, \pi) = \sum_{(u,v) \in E} c(u,v) \cdot |\pi(u) - \pi(v)|.$$  

The weighted sum of edge lengths can be equivalently defined as:

$$la(G, \pi) = \sum_{1 \leq i < n} c(\text{cut}_\pi(i))$$

2.2 Decomposition Trees
A decomposition tree of a graph $G(V,E)$ is a rooted binary tree with a mapping of the tree nodes to subsets of vertices as follows. The root is mapped to $V$, the subsets mapped to every two siblings constitute a partitioning of
the subset mapped to their parent, and leaves are mapped to subsets containing a single vertex. We denote the subset of vertices mapped to a tree node \( t \) by \( V(t) \). For every tree node \( t \), let \( T(t) \) denote the subtree of \( T \), the root of which is \( t \). The inner cut of an internal tree node \( t \) is the set of edges in the cut \( (V(t_1), V(t_2)) \), where \( t_1 \) and \( t_2 \) denote the children of \( t \). We denote the inner cut of \( t \) by \( \text{in-cut}(t) \).

Every DFS traversal of a decomposition tree induces an ordering of \( V \) according to the order in which the leaves are visited. Since in each internal node there are two possible orders in which the children can be visited, it follows that \( 2^{n-1} \) orderings are induced by DFS traversals. Each such ordering is specified by determining for every internal node which child is visited first. In “graphic” terms, if the first child is always drawn as the left child, then the induced ordering is the order of the leaves from left to right.

### 2.3 Orientations and Optimal Orientations

Consider an internal tree node \( t \) of a decomposition tree. An orientation of \( t \) is a bit that determines which of the two children of \( t \) is considered as its left child. Our convention is that when a DFS is performed, the left child is always visited first. Therefore, a decomposition tree, all the internal nodes of which are assigned orientations, induces a unique ordering. We refer to an assignment of orientations to all the internal nodes as an orientation of the decomposition tree.

Consider a decomposition tree \( T \) of a graph \( G(V, E) \). An orientation of \( T \) is optimal with respect to an ordering problem if the cost associated with the ordering induced by the orientation is minimum among all the orderings induced by \( T \).

### 3 Computing An Optimal Orientation

In this section we present a dynamic programming algorithm for computing an optimal orientation of a decomposition tree with respect to \( \text{min-la} \) and \( \text{min-cw} \). We first present an algorithm for \( \text{min-la} \), and then describe the modifications needed for \( \text{min-cw} \).

Let \( T \) denote a decomposition of \( G(V, E) \). We describe a recursive algorithm \( \text{orient}(t, \alpha) \) for computing an optimal orientation of \( T \) for \( \text{min-la} \). A say that a decomposition tree is oriented if all its internal nodes are assigned orientations. The algorithm returns an oriented decomposition tree isomorphic to \( T \). The parameters of the algorithm are a tree node \( t \) and an assignment of orientations to the ancestors of \( t \). The orientations of the
ancestors of \( t \) are specified by a binary string \( \alpha \) whose length equals \( \text{depth}(t) \). The \( i \)th bit in \( \alpha \) signifies the orientation of the \( i \)th node along the path from the root of \( T \) to \( t \).

The vertices of \( V(t) \) constitute a contiguous block in every ordering that is induced by the decomposition tree \( T \). The sets of vertices that appear to the left and right of \( V(t) \) are determined by the orientations of the ancestors of \( t \) (which are specified by \( \alpha \)). Given the orientations of the ancestors of \( t \), let \( L \) and \( R \) denote the set of vertices that appear to the left and right of \( V(t) \), respectively. We call the partition \((L, V(t), R)\) an ordered partition of \( V \).

Consider an ordered partition \((L, V(t), R)\) of the vertex set \( V \) and an ordering \( \pi \) of the vertices of \( V(t) \). Algorithm \( \text{orient}(t, \alpha) \) is based on the local cost of edge \((u, v)\) with respect to \((L, V(t), R)\) and \( \pi \). The local cost applies only to edges incident to \( V(t) \) and it measures the length of the projection of the edge on \( V(t) \). Formally, the local cost is defined by

\[
\text{local cost}_{L, V(t), R, \pi}(u, v) = \begin{cases} 
  c(u, v) \cdot |\pi(u) - \pi(v)| & \text{if } u, v \in V(t) \\
  c(u, v) \cdot \pi(u) & \text{if } u \in V(t) \text{ and } v \in L \\
  c(u, v) \cdot (|V(t)| - \pi(u)) & \text{if } u \in V(t) \text{ and } v \in R \\
  0 & \text{otherwise.}
\end{cases}
\]

Note that the contribution to the cut corresponding to \( L \cup V(t) \) is not included.

Algorithm \( \text{orient}(t, \alpha) \) proceeds as follows:

1. If \( t \) is a leaf, then return \( T(t) \) (a leaf is not assigned an orientation).
2. Otherwise (\( t \) is not a leaf), let \( t_1 \) and \( t_2 \) denote children of \( t \). Compute optimal oriented trees for \( T(t_1) \) and \( T(t_2) \) for both orientations of \( t \). Specifically,

   (a) \( T^0(t_1) = \text{orient}(t_1, \alpha \cdot 0) \) and \( T^0(t_2) = \text{orient}(t_2, \alpha \cdot 0) \).
   (b) \( T^1(t_1) = \text{orient}(t_1, \alpha \cdot 1) \) and \( T^1(t_2) = \text{orient}(t_2, \alpha \cdot 1) \).
3. Let \( \pi_0 \) denote the ordering of \( V(t) \) obtained by concatenating the ordering induced by \( T^0(t_1) \) and the ordering induced by \( T^0(t_2) \). Let \( \text{cost}_0 = \sum_{e \in E} \text{local cost}_{L, V(t), R, \pi_0}(e) \).
4. Let \( \pi_1 \) denote the ordering of \( V(t) \) obtained by concatenating the ordering induced by \( T^1(t_2) \) and the ordering induced by \( T^1(t_1) \). Let \( \text{cost}_1 = \sum_{e \in E} \text{local cost}_{L, V(t), R, \pi_1}(e) \). (Note that here the vertices of \( V(t_2) \) are placed first.)
5. If \( \text{cost}_0 < \text{cost}_1 \), the orientation of \( t \) is 0. Return the oriented tree \( T' \) the left child of which is the root of \( T^0(t_1) \) and the right child of which is the root of \( T^0(t_2) \).

6. Otherwise (\( \text{cost}_0 \geq \text{cost}_1 \)), the orientation of \( t \) is 1. Return the oriented tree \( T' \) the left child of which is the root of \( T^1(t_2) \) and the right child of which is the root of \( T^1(t_1) \).

The correctness of the \textit{orient} algorithm is summarized in the following claim which can be proved by induction.

\textbf{Claim 1:} Suppose that the orientations of the ancestors of \( t \) are fixed as specified by the string \( \alpha \). Then, Algorithm \textit{orient}(\( t, \alpha \)) computes an optimal orientation of \( T(t) \). When \( t \) is the root of the decomposition tree and \( \alpha \) is an empty string, Algorithm \textit{orient}(\( t, \alpha \)) computes an optimal orientation of \( T \).

An optimal orientation for \textsc{mincw} can be computed by modifying the local cost function. Consider an ordered partition \((L, V(t), R)\) and an ordering \( \pi \) of \( V(t) \). Let \( V_i(t) \) denote that set \( \{v \in V(t) : \pi(v) \leq i\} \). Let \( E(t) \) denote the set of edges with at least one endpoint in \( V(t) \). Let \( i \in [0,|V(t)|] \). The \( i \)th local cut with respect to \((L, V(t), R)\) and an ordering \( \pi \) of \( V(t) \) is the set of edges defined by,

\[
\text{local}_{cut_{(L,V(t),R)}}(i) = \{(u,v) \in E(t) : u \in L \cup V_i(t) \text{ and } v \in (V(t)-V_i(t)) \cup R\}.
\]

The local cutwidth is defined by,

\[
\text{local}_{cw}(L,V(t),R,\pi) = \max_{i \in [0,|V(t)|]} \sum_{e \in \text{local}_{cut_{(L,V(t),R)}}(i)} c(u,v).
\]

The algorithm for computing an optimal orientation of a given decomposition tree with respect to \textsc{mincw} is obtained by computing \( \text{cost}_\alpha = \text{local}_{cw}(L,V(t),R,\pi_\alpha) \) in steps 3 and 4 for \( \sigma = 0,1 \).

\section{Designing The Algorithm}

In this section we propose a design of the algorithm that has linear space complexity. The time complexity is \( O(n^{2.2}) \) if the graph has bounded degree and the decomposition tree is 1/3-balanced.
4.1 Space Complexity: The Orientation Tree

We define a tree, called an orientation tree, that corresponds to the recursion tree of Algorithm orient\((t, \alpha)\). Under the interpretation of this algorithm as a dynamic program, this orientation tree represents the table. The orientation tree \(\widehat{T}\) corresponding to a decomposition tree \(T\) is a “quadary” tree. Every orientation tree node \(\widehat{t} = \langle t, \alpha \rangle\) corresponds to a decomposition tree node \(t\) and an assignment \(\alpha\) of orientations to the ancestors of \(t\). Therefore, every decomposition tree node \(t\) has \(2^{\text{depth}(t)}\) “images” in the orientations tree. Let \(t_1\) and \(t_2\) be the children of \(t\) in the decomposition tree, and let \(\sigma \in \{0, 1\}\) be a possible orientation of \(t\). The four children of \(\langle t, \alpha \rangle\) are \(\langle t_i, \alpha \cdot \sigma \rangle\), where \(i \in \{1, 2\}\), \(\sigma \in \{0, 1\}\). Figure 1 depicts a decomposition tree and the corresponding orientation tree.

The time complexity of the algorithm presented in Section 3 is clearly at least proportional to the size of the orientations tree. The number of nodes in the orientations tree is proportional to \(\sum_{v \in V} 2^{\text{depth}(v)}\), where the \(\text{depth}(v)\) is the depth of \(v\) in the decomposition tree. This implies the running time is at least quadratic if the tree is perfectly balanced. If we store the entire orientations tree in memory at once, our space requirement is also at least quadratic. However, if we are a bit smarter about how we use space, and never store the entire orientations tree in memory at once, we can reduce the space requirement to linear.

The recursion tree of Algorithm orient\(\text{root}(T), \phi\) is isomorphic to the orientation tree \(\widehat{T}\). In fact, Algorithm orient\(\text{root}(T), \alpha\) assigns local costs to orientation tree nodes in DFS order. We suggest the following linear space implementation. Consider an orientation tree node \(\widehat{t} = \langle t, \alpha \rangle\). Assume that when orient\((t, \alpha)\) is called, it is handed a “workspace” tree isomorphic to \(T(t)\) which it uses for workspace as well as for returning the optimal orientation. Algorithm orient\((t, \alpha)\) allocates an “extra” copy of \(T(t)\), and is called recursively for each of it four children. Each call for a child is given a separate “workspace” subtree within the two isomorphic copies of \(T(t)\) (i.e. within the workspace and extra trees). Upon completion of these 4 calls, the two copies of \(T(t)\) are oriented; one corresponding to a zero orientation of \(t\) and one corresponding to an orientation value of 1 for \(t\). The best of these trees is copied into the “workspace” tree, if needed, and the “extra” tree is freed. Assuming that one copy of the decomposition tree is used throughout the algorithm, the additional space complexity of this implementation satisfies the following recurrence:

\[
\text{space}(\widehat{t}) = \text{sizeof}(T(t)) + \max_{\text{child of } \widehat{t}} \text{space}(t').
\]
Figure 1: A decomposition tree and the corresponding orientation tree. The four node decomposition tree is depicted as a gray shadow. The corresponding orientation tree is depicted in the foreground. The direction of each non-root node in the orientation tree corresponds to the orientation of its parent (i.e., a node depicted as a fish swimming to the left signifies that its parent’s orientation is left). The label of an edge entering a node in the orientation tree signifies whether the node is a left child or a right child and the orientation of its parent.
Since $\text{sizeof}(T(t)) \leq 2^{\text{depth}(T(t))}$, it follows that

$$\text{space}(\hat{t}) \leq 2 \cdot 2^{\text{depth}(T(t))}.$$  

The space complexity of the proposed implementation of $\text{orient}(T, \alpha)$ is summarized in the following claim.

**Claim 2:** The space complexity of the proposed implementation is $O(2^{\text{depth}(T)})$.

If the decomposition tree is balanced, then $\text{space}(\text{root}(T)) = O(n)$.

### 4.2 Time Complexity

Viewing Algorithm $\text{orient}(t, \alpha)$ as a DFS traversal of the orientation tree $\hat{T}(t)$ corresponding to $T(t)$ also helps in designing the algorithm so that each visit of an internal orientation tree node requires only constant time. Suppose that each child of $t$ is assigned a local cost (i.e. $\text{cost}(\text{left}(\sigma)), \text{cost}(\text{right}(\sigma))$, for $\sigma = 0, 1$). Let $t_1$ and $t_2$ denote the children of $t$. Let $(L, V(t), R)$ denote the ordered partition of $V$ induced by the orientations of the ancestors of $t$ specified by $\alpha$. The following equations define $\text{cost}_0$ and $\text{cost}_1$:

$$\begin{align*}
\text{cost}_0 &= \text{cost}(\text{left}(0)) + \text{cost}(\text{right}(0)) \\
n &\quad + |V(t_2)| \cdot c(V(t_1), R) + |V(t_1)| \cdot c(L, V(t_2)) \\
\text{cost}_1 &= \text{cost}(\text{left}(1)) + \text{cost}(\text{right}(1)) \\
n &\quad + |V(t_1)| \cdot c(V(t_2), R) + |V(t_2)| \cdot c(L, V(t_1))
\end{align*}$$

We now describe how Equation 1 can be computed in constant time. Let $\text{in\_cut}(t)$ denote the cut $(V(t_1), V(t_2))$. The capacity of $\text{in\_cut}(t)$ can be pre-computed by scanning the list of edges. For every edge $(u, v)$, update $\text{in\_cut}(\text{lca}(u, v))$ by adding $c(u, v)$ to it.

We now define outer cuts. Consider the ordered partition $(L, V(t), R)$ corresponding to an orientation tree node $\hat{t}$. The left outer cut and the right outer cut of $\hat{t}$ are the cuts $(L, V(t))$ and $(V(t), R)$, respectively. We denote the left outer cut by $\text{left\_cut}(\hat{t})$ and the right outer cut by $\text{right\_cut}(\hat{t})$.

We describe how outer cuts are computed for leaves and for interior nodes. The capacity of the outer cuts of a leaf $\hat{t}$ are computed by considering the edges incident to $t$ (since $t$ is a leaf we identify it with a vertex in $V$). For every edge $(t, u)$, the orientation of the orientation tree node along the path from the root to $\hat{t}$ that corresponds to $\text{lca}(t, u)$ determines whether the edge belongs to the left outer cut or to the right outer cut. Since the least common ancestors in the decomposition tree of the endpoints of every edge
are precomputed when the inner cuts are computed, we can compute the outer cuts of a leaf $f$ in $O(\deg(t))$ time.

The outer cuts of a non-leaf $f$ can be computed from the outer cuts of its children as follows:

$$
c(\text{left cut}(f)) = c(\text{left cut}(left(0))) + c(\text{left cut}(right(0))) - c(\text{in cut}(t))
$$

$$
c(\text{right cut}(f)) = c(\text{right cut}(left(0))) + c(\text{right cut}(right(0))) - c(\text{in cut}(t))
$$

Hence, the capacities of the outer cuts can be computed while the orientation tree is traversed. The following reformulation of Equation 1 shows that $cost_0$ and $cost_1$ can be computed in constant time.

$$
cost_0 = cost(left(0)) + cost(right(0)) + |V(t_2)| \cdot c(right cut(f_2)) + |V(t_1)| \cdot c(left cut(f_1))
$$

$$
cost_1 = cost(left(1)) + cost(right(1)) + |V(t_1)| \cdot c(right cut(f_2)) + |V(t_2)| \cdot c(left cut(f_1))
$$

**Minimum Cutwidth.** An adaptation of Equation 1 for mincw is given below:

$$
cost_0 = \max \{ \text{local cw}(L, V(t_1), V(T_2) \cup R) + c(L, V(t_2)), \text{local cw}(L \cup V(t_1), V(T_2), R) + c(V(t_1), R) \}
$$

$$
cost_1 = \max \{ \text{local cw}(L, V(t_2), V(T_1) \cup R) + c(L, V(t_1)), \text{local cw}(L \cup V(t_2), V(T_1), R) + c(V(t_2), R) \}
$$

These costs can be computed in constant time using the same technique described for minla.

Now we analyze the time complexity of the proposed implementation of Algorithm $\text{orient}(t, \alpha)$. We split the time spent by the algorithm into two parts: (1) The pre-computation of the inner cuts and the least common ancestors of the edges, (2) the time spent traversing the orientation tree $\hat{T}$ (interior nodes as well as leaves).

Precomputing the inner cuts and least common ancestors of the edges requires $O(\alpha \cdot depth(T))$ time, where $depth(T)$ is the maximum depth of the decomposition tree $T$. Assume that the least common ancestors are stored and need not be recomputed.

The time spent traversing the orientation tree is analyzed as follows. We consider two cases: Leaves - the amount of time spent in a leaf $\hat{f}$ is linear in
the degree of $t$. Interior Nodes - the amount of time spent in an interior node $t$ is constant. This implies that the complexity of the algorithm is linear in

$$|\tilde{T} - \text{leaves}(\tilde{T})| + \sum_{t \in \text{leaves}(\tilde{T})} \deg(t).$$

Every node $t \in T$ has $2^{\text{depth}(t)}$ “images” in $\tilde{T}$. Therefore, the complexity in terms of the decomposition tree $T$ equals

$$\sum_{t \in T - \text{leaves}(T)} 2^{\text{depth}(t)} + \sum_{t \in \text{leaves}(T)} 2^{\text{depth}(t)} \cdot \deg(t).$$

If the degrees of the vertices are bounded by a constant, then the complexity is linear in

$$\sum_{t \in T} 2^{\text{depth}(t)}.$$ 

This quantity is quadratic if $T$ is perfectly balanced, i.e. the vertex subsets are bisected in every internal tree node. We quantify the balance of a decomposition tree as follows:

**Definition 1:** A binary tree $T$ is $\rho$-balanced if for every internal node $t \in T$ and for every child $t'$ of $t$

$$\rho \cdot |V(t)| \leq |V(t')| \leq (1 - \rho) \cdot |V(t)|$$

The following claim summarizes the time complexity of the algorithm.

**Claim 3:** If the decomposition tree $T$ is $\rho$ balanced, then

$$\sum_{t \in T} 2^{\text{depth}(t)} \leq n^\beta,$$

where $\beta$ is the solution to the equation

$$\frac{1}{\rho^\beta + (1 - \rho)^\beta} = \frac{1}{2} \quad (4)$$

Observe that if $\rho \in (0, 1/2]$, then $\beta \geq 2$. Moreover, $\beta$ increases as $\rho$ increases.

**Proof:** The quantity which we want to bound is the size of the orientation tree. This quantity satisfies the following recurrence:

$$f(T(t)) \triangleq \begin{cases} 1 & \text{if } t \text{ is a leaf} \\ 2f(T(t_1)) + 2f(T(t_2)) & \text{otherwise.} \end{cases}$$
Define the function $f^{*}_{\rho}(n)$ as follows

$$f^{*}_{\rho}(n) = \max\{f(T) : T \text{ is } \alpha\text{-balanced and has } n \text{ leaves}\}.$$ 

The function $f^{*}_{\rho}(n)$ satisfies the following recurrence:

$$f^{*}_{\rho}(n) \triangleq \begin{cases} 
1 & \text{if } n = 1 \\
\max \left\{2f^{*}_{\rho}(n') + 2f^{*}_{\rho}(n - n') : n' \in [\lceil \rho n \rceil, (n - \lfloor \rho n \rfloor)] \right\} & \text{otherwise}
\end{cases}$$

We prove that $f^{*}_{\rho}(n) \leq n^3$ by induction on $n$. The induction hypothesis, for $n = 1$, is trivial. The induction step is proven as follows:

$$f^{*}_{\rho}(n) = \max \left\{2f^{*}_{\rho}(n') + 2f^{*}_{\rho}(n - n') : n' \in [\lceil \rho n \rceil, (n - \lfloor \rho n \rfloor)] \right\} \leq \max \left\{2(n')^3 + 2(n - n')^3 : n' \in [\lceil \rho n \rceil, (n - \lfloor \rho n \rfloor)] \right\} \leq \max \left\{2(x)^3 + 2(n - x)^3 : x \in [\rho n, n - \rho n] \right\} = 2(\rho n)^3 + 2(n - \rho n)^3 = 2n^3 \cdot (\rho^3 + (1 - \rho)^3) = n^3$$

The first line is simply the recurrence that $f^{*}_{\rho}(n)$ satisfies; the second line follows from the induction hypothesis; in the third line we relax the range over which the maximum is taken; the fourth line is justified by the convexity of $x^3 + (n - x)^3$ over the range $x \in [\rho n, n - \rho n]$ when $\beta \geq 2$; in the fifth line we rearrange the terms; and the last line follows from the definition of $\beta$. \qed

We conclude by bounding the time complexity of the orientation algorithm for 1/3-balanced decomposition trees.

**Corollary 4:** If $T$ is a 1/3-balanced decomposition tree of a bounded degree, then the orientation tree $\hat{T}$ of $T$ has at most $n^2.2$ leaves.

**Proof:** The solution of Equation (4) with $\rho = 1/3$ is $\beta < 2.2$. \qed

For graphs with unbounded degree, this algorithm runs in time $O(m \cdot 2^{\text{depth}(T)})$ which is $O(m \cdot n^{1/\log_2(1/1-\rho)})$. Alternatively, a slightly different algorithm gives a running time of $n^2$ in general (not just for constant degree), but the space requirement of this algorithm is $O(n \log n)$. It is based on computing the outer cuts of a leaf of the orientations tree on the way down the recursion. We omit the details.
5 Experiments And Heuristics

Since we are not improving the theoretical approximation ratios for the MINLA or MINCW problems, it is natural to ask whether finding an optimal orientation of a decomposition tree is a useful thing to do in practice. The most comprehensive experimentation on either problem was performed for the MINLA problem by Petit [P97]. Petit collected a set of benchmark graphs and ran several different algorithms on them, comparing their quality. He found that Simulated Annealing yielded the best results. The benchmark consists of 5 random graphs, 3 “regular” graphs (a hypercube, a mesh, and a binary tree), 3 graphs from finite element discretizations, 5 graphs from VLSI designs, and 5 graphs from graph drawing competitions.

5.1 Gaps between orientations

The first experiment was performed to check if there is a significant gap between the costs of different orientations of decomposition trees. Conveniently, our algorithm can be easily modified to find the worst possible orientation; every time we compare two local costs to decide on an orientation, we simply take the orientation that achieves the worst local cost. In this experiment, we constructed decomposition trees for all of Petit’s benchmark graphs using the balanced graph partitioning program HMETIS [GK98]. HMETIS is a fast heuristic that searches for balanced cuts that are as small as possible. For each decomposition tree, we computed four orientations:

1. Naive orientation - all the decomposition tree nodes are assigned a zero orientation. This is the ordering you would expect from a recursive bisection algorithm that ignored orientations.

2. Random orientation - the orientations of the decomposition tree nodes are chosen randomly to be either zero or one, with equal probability.

3. Best orientation - computed by our orientation algorithm.

4. Worst orientation - computed by a simple modification of our algorithm.

The results are summarized in Table 1. On all of the “real-life” graphs, the best orientation had a cost of about half of the worst orientation. Furthermore, the costs on all the graphs were almost as low as those obtained by Petit (Table 5), who performed very thorough and computationally-intensive experiments.
Notice that the costs of the “naive” orientations do not compete well with those of Petit. This shows that in this case, using the extra degrees of freedom afforded by the decomposition tree was essential to achieving a good quality solution. These results also motivated further experimentation to see if we could achieve comparable or better results using more iterations, and the improvement heuristic alluded to earlier.

5.2 Experiment Design

We ran the following experiments on the benchmark graphs:

1. Decompose & Orient. Repeat the following two steps $k$ times, and keep the best ordering found during these $k$ iterations.

   (a) Compute a decomposition tree $T$ of the graph. The decomposition is computed by calling the HMETIS graph partitioning program recursively [GK98]. HMETIS accepts a balance parameter $\rho$. HMETIS is a random algorithm, so different executions may output different decomposition trees.

   (b) Compute an optimal orientation of $T$. Let $\pi$ denote the ordering induced by the orientation of $T$.

2. Improvement Heuristic. Repeat the following steps $k'$ times (or until no improvement is found during 10 consecutive iterations):

   (a) Let $\pi_0$ denote the best ordering $\pi$ found during the Decompose & Orient step.

   (b) Set $i = 0$.

   (c) Compute a random $\rho$-balanced decomposition tree $T'$ based on $\pi_i$. The decomposition tree $T'$ is obtained by recursively partitioning the blocks in the ordering $\pi_i$ into two contiguous sub-blocks. The partitioning is a random partition that is $\rho$-balanced.

   (d) Compute an optimal orientation of $T'$. Let $\pi_{i+1}$ denote the ordering induced by the orientation of $T'$. Note that the cost of $\pi_{i+1}$ is not greater than the cost of the ordering $\pi_i$. Increment $i$ and return to Step 2c, unless $i = k' - 1$ or no improvement in the cost has occurred for the last 10 iterations.

3. Simulated Annealing. We used the best ordering found by the Heuristic Improvement stage as an input to the Simulated Annealing program of Petit [P97].
Preliminary experiments were run in order to choose the balance parameter $\rho$. In graphs of less than 1000 nodes, we simply chose the balance parameter that gave the best ordering. In bigger graphs, we also restricted the balance parameter so that the orientation tree would not be too big. The parameters used for the Simulated Annealing program were also chosen based on a few preliminary experiments.

5.3 Experimental Environment

The programs have been written in C and compiled with a gcc compiler. The programs were executed on a dual processor Intel P-III 600MHz computer running under Red Hat Linux. The programs were only compiled for one processor, and therefore only ran on one of the two processors.

5.4 Experimental Results

The following results were obtained:

1. Decompose & Orient. The results of the Decompose & Orient stage are summarized in Table 2. The columns of Table 2 have the following meaning: “UB Factor” - the balance parameter used when HMETIS was invoked. The relation between the balance parameter $\rho$ and the UB Factor is $(1/2 - \rho) \cdot 100$. “Avg. OT Size” - the average size of the orientation tree corresponding to the decomposition tree computed by HMETIS. This quantity is a good measure of the running time without overheads such as reading and writing of data. “Avg. L.A. Cost” - the average cost of the ordering induced by an optimal orientation of the decomposition tree computed by HMETIS. “Avg. HMETIS Time (sec)” - the average time in seconds required for computing a decomposition tree, “Avg. Orienting Time (sec)” - the average time in seconds required to compute an optimal orientation, and “Min L.A. Cost” - the minimum cost of an ordering, among the computed orderings.

2. Heuristic Improvement. The results of the Heuristic Improvement stage are summarized in Table 3. The columns of Table 3 have the following meaning: “Initial solution” - the cost of the ordering computed by the Decompose & Orient stage, “10 Iterations” - the cost of the ordering obtained after 10 iterations of Heuristic Improvements, “Total Iterations” - The number iterations that were run until there was no improvement for 10 consecutive iterations, “Avg. Time per Iteration” - the average time for each iteration (random decomposition
and orientation), and “Final Cost” - the cost of the ordering computed by the Heuristic Improvement stage.

The same balance parameter was used in the Decompose & Orient stage and in the Heuristic Improvement stage. Since the partition is chosen randomly in the Heuristic Improvement stage such that the balance is never worse than $\rho$, the decomposition trees obtained were shallower. This fact is reflected in the shorter running times required for computing an optimal orientation.

3. Simulated Annealing. The results of the Simulated Annealing program are summarized in Table 4.

5.5 Conclusions

Table 5 summarizes our results and compares them with the results of Pet- tit [P97]. The running times in the table refer to a single iteration (the running time of the Decompose & Orient stage refers only to orientation not including decomposition using HMETIS). Petit ran 10-100 iterations on an SGI Origin 2000 computer with 32 MIPS R10000 processors.

Our main experimental conclusion is that running our Decompose & Orient algorithm usually yields results within 5-10% of Simulated Annealing, and is significantly faster. The results were improved to comparable with Simulated Annealing by using our Improvement Heuristic. Slightly better results were obtained by using our computed ordering as an initial solution for Simulated Annealing.

5.6 Availability of programs and results

Programs and output files can be downloaded from http://www.eng.tau.ac.il/~guy/Projects/Minla/index.html.
| Graph | Naive | Random | Worst | Best | Best/Worst |
|-------|-------|--------|-------|------|------------|
| random A1 | 1012523 | 1042034 | 980631 | 0.94 |
| random A2 | 6989176 | 6998714 | 6972740 | 0.99 |
| random A3 | 14780970 | 14767044 | 14826303 | 0.99 |
| random A4 | 19190232 | 1907549 | 1963952 | 0.98 |
| random G4 | 210817 | 220669 | 282280 | 0.56 |
| hc10 | 523776 | 523776 | 523776 | 1.0 |
| mesh 3x3 | 55084 | 55084 | 55084 | 1.0 |
| bintree10 | 4850 | 4850 | 4850 | 1.0 |
| seat | 729782 | 729782 | 887231 | 0.57 |
| whisker3 | 2173607 | 2423313 | 2333350 | 0.47 |
| cly | 91135 | 91135 | 91135 | 1.0 |
| airfoil | 524201 | 677191 | 337237 | 0.5 |
| cly | 122838 | 108936 | 172398 | 0.48 |
| cly | 152903 | 160114 | 293747 | 0.5 |
| cly | 140178 | 146764 | 246104 | 0.5 |
| white | 37291 | 137291 | 203594 | 0.51 |
| cly | 12191 | 137291 | 203594 | 0.51 |
| cly | 23342 | 23342 | 23342 | 1.0 |
| cly | 2938 | 2938 | 2938 | 1.0 |
| cly | 2475 | 2475 | 2475 | 1.0 |
| white | 695 | 695 | 695 | 1.0 |
| white | 3477 | 3477 | 3477 | 1.0 |

Table 1: A comparison of arrangement costs for different orientations of a single decomposition tree for each graph.
## Table 2: Results of the Decompose & Orient stage. 100 iterations were executed for all the graphs, except for 3elt and airfoil1 - 60 iterations, and whitaker3 - 25 iterations.
| Graph   | Initial solution | 10 Iterations | 20 Iterations | 30 Iterations | Total Iterations | Avg. Time / Iteration (sec) | Final Cost |
|---------|------------------|----------------|---------------|---------------|------------------|-----------------------------|-------------|
| randomA1 | 956837           | 947483         | 941002        | 935014        | 1048             | 2.63                        | 897910      |
| randomA2 | 6791813          | 6759449        | 6730868       | 6717108       | 500              | 9.84                        | 6604738     |
| randomA3 | 14621489         | 14576923       | 14537348      | 14506843      | 200              | 23.73                       | 14365385    |
| randomA4 | 1852192          | 1833589        | 1821592       | 1812344       | 557              | 3.73                        | 1761666     |
| randomG4 | 156447           | 150477         | 149456        | 149243        | 76               | 3.53                        | 149185      |
| hc10    | 523776           | 523776         | -             | -             | 10               | 2.00                        | 523776      |
| mesh33x33 | 35728           | 35492         | 35325         | 35212         | 152              | 1.28                        | 34845       |
| bintree10 | 3740           | 3724         | 3720         | 3718         | 47               | 0.75                        | 3714        |
| 3elt    | 414051           | 400469        | 395594        | 392474        | 724              | 46.49                       | 372107      |
| airfoil1 | 325635           | 313975        | 309998        | 307383        | 630              | 36.94                       | 292761      |
| whitaker3 | 1441887        | 1408337       | 1393045       | 1381345       | 35               | 149.43                      | 1376511     |
| c1y     | 63803            | 63283         | 63085         | 62973         | 80               | 0.73                        | 62903       |
| c2y     | 81731            | 81094         | 80877         | 80679         | 171              | 1.19                        | 80109       |
| c3y     | 130213           | 129255        | 128911        | 128650        | 267              | 1.91                        | 127729      |
| c4y     | 119016           | 118109        | 117690        | 117438        | 160              | 2.14                        | 116621      |
| c5y     | 99772            | 99051         | 98813         | 98661         | 182              | 1.72                        | 98004       |
| gd95d   | 512              | 506           | -             | 14            | 0.00              | 506                         |             |
| gd96a   | 112551           | 110744        | 109605        | 108887        | 977              | 1.07                        | 102294      |
| gd96b   | 1457             | 1427          | 1424          | 1417          | 38               | 0.02                        | 1417        |
| gd96c   | 533              | 520           | 520           | -             | 20               | 0.00                        | 520         |
| gd96d   | 2521             | 2463          | 2454          | 2449          | 62               | 0.03                        | 2436        |

Table 3: Results of the Heuristic Improvement stage. Balance parameters equal the UB factors in Table 2.
| Graph      | Initial solution | Total Running Time (sec) | Final Cost |
|------------|------------------|--------------------------|------------|
| randomA1   | 897910           | 2328.14                  | 884261     |
| randomA2   | 6604738          | 191645                   | 6576912    |
| randomA3   | 14365385         | 58771.8                  | 14289214   |
| randomA4   | 1761666          | 10895.6                  | 1747143    |
| randomG4   | 149185           | 8375.88                  | 146996     |
| hc10       | 523776           | 2545.16                  | 523776     |
| mesh33x33  | 34845            | 278.037                  | 33531      |
| bintree10  | 3714             | 59.782                   | 3762       |
| 3elt       | 372107           | 5756.52                  | 363204     |
| airfoil1   | 292761           | 4552.14                  | 289217     |
| whitaker3  | 1376511          | 29936.8                  | 1200374    |
| c1y        | 62903            | 204.849                  | 62333      |
| c2y        | 80109            | 298.844                  | 79571      |
| c3y        | 127729           | 553.064                  | 127065     |
| c4y        | 116621           | 579.751                  | 115222     |
| c5y        | 98004            | 445.604                  | 96956      |
| gd95c      | 506              | 1.909                    | 506        |
| gd96a      | 102294           | 268.454                  | 99944      |
| gd96b      | 1417             | 2.923                    | 1422       |
| gd96c      | 520              | 1.175                    | 519        |
| gd96d      | 2436             | 4.163                    | 2409       |

Table 4: Results of the Simulated Annealing program. The parameters were $t_0=4$, $t_1=0.1$, $\alpha=0.99$ except for whitaker3, airfoil1 and 3elt for which $\alpha=0.975$ and randomA3 for which $t_1=1$ and $\alpha=0.8$
The table below compares the results and running times of the Decompose & Orient stage, Heuristic Improvement stage, and Simulated Annealing stage for various graphs. The results are derived from Petit's paper [P97].

| Graph       | Petit's Results [P97] | Decompose & Orient stage | Heuristic Improvement stage | Simulated Annealing stage |
|-------------|------------------------|---------------------------|-----------------------------|---------------------------|
|             | cost (sec) | time (sec) | cost (sec) | % diff. | time (sec) | cost (sec) | % diff. | time (sec) | cost (sec) | % diff. | time (sec) |
| randomA1    | 900992     | 317       | 956837    | 6.20%   | 5         | 897910    | -0.34%  | 3         | 884261    | -1.86%  | 2328      |
| randomA2    | 6584658   | 481       | 6791813   | 3.15%   | 18        | 6604738   | 0.30%   | 10        | 6576912   | -0.12%  | 191645    |
| randomA3    | 14310861  | 682       | 14621489  | 2.17%   | 71        | 14365385  | -0.38%  | 24        | 14289214  | -1.51%  | 58771     |
| randomA4    | 1753265   | 346       | 1852192   | 5.64%   | 8         | 1761666   | 0.48%   | 4         | 1747143   | -0.35%  | 10896     |
| randomG4    | 150490    | 346       | 156447    | 3.96%   | 8         | 149185    | -0.87%  | 4         | 146996    | -2.32%  | 8376      |
| hc10        | 548352    | 335       | 523776    | -4.48%  | 2         | 523776    | -4.48%  | 2         | 523776    | -4.48%  | 2545      |
| mesh33x33   | 34515     | 336       | 35728     | 3.51%   | 1         | 34845     | 0.96%   | 1         | 33531     | -2.85%  | 278       |
| bintree10   | 4069      | 291       | 3740      | -8.09%  | 1         | 3714      | -8.72%  | 1         | 3762      | -7.54%  | 50        |
| 3elt        | 375387    | 6660      | 414051    | 10.30%  | 61        | 372107    | -0.87%  | 46        | 363204    | -3.25%  | 5757      |
| airfoil1    | 288977    | 5364      | 325635    | 12.69%  | 53        | 292761    | 1.31%   | 37        | 289217    | 0.08%   | 4552      |
| whitaker3   | 119977    | 3197      | 1441887   | 20.18%  | 139       | 1376511   | 14.73%  | 149       | 1200374   | 0.05%   | 2937      |
| c1y         | 63854     | 196       | 63803     | -0.08%  | 2         | 62903     | -1.49%  | 1         | 62333     | -2.38%  | 205       |
| c2y         | 79500     | 277       | 81731     | 2.81%   | 6         | 80109     | 0.77%   | 1         | 79571     | 0.09%   | 299       |
| c3y         | 124708    | 509       | 130213    | 4.41%   | 4         | 127729    | 2.42%   | 2         | 127065    | 1.89%   | 553       |
| c4y         | 117254    | 535       | 119016    | 1.50%   | 4         | 116621    | -0.54%  | 2         | 115222    | -1.73%  | 580       |
| c5y         | 102769    | 416       | 99772     | -2.92%  | 4         | 98004     | -4.64%  | 2         | 96956     | -5.66%  | 446       |
| gd95c       | 509       | 1         | 512       | 0.59%   | 0         | 506       | -0.59%  | 0         | 506       | -0.59%  | 2         |
| gd96a       | 104698    | 341       | 112551    | 7.50%   | 2         | 102294    | -2.30%  | 1         | 99944     | -4.54%  | 268       |
| gd96b       | 1416      | 3         | 1457      | 2.90%   | 0         | 1417      | 0.07%   | 0         | 1422      | 0.42%   | 3         |
| gd96c       | 519       | 1         | 533       | 2.70%   | 0         | 520       | 0.19%   | 0         | 519       | 0.00%   | 1         |
| gd96d       | 2393      | 8         | 2521      | 5.35%   | 0         | 2436      | 1.80%   | 0         | 2409      | 0.67%   | 4         |

Table 5: Comparison of results and running times. Note (a) Time of Decompose & Orient and Heuristic Improvement are given per iteration. (b) Decompose & Orient time does not include time for recursive decomposition.
6 Acknowledgments

We would like to express our gratitude to Zvika Brakerski for his help with using HMETIS, running the experiments, generating the tables, and writing scripts.

References

[BL84] S.N. Bhatt and F.T. Leighton, “A framework for solving VLSI graph layout problems”, JCSS, Vol. 28, pp. 300-343, (1984).

[ENRS00] G. Even, J. Naor, S. Rao, and B. Schieber, “Divide-and-Conquer Approximation Algorithms via Spreading Metrics”, Journal of the ACM, Volume 47, No. 4, pp. 585 - 616, Jul. 2000.

[ENRS99] G. Even, J. Naor, S. Rao, and B. Schieber, “Fast approximate graph partitioning algorithms”, SIAM Journal on Computing, Volume 28, Number 6, pp. 2187-2214, 1999.

[FD] Frost, D., and Dechter, R. ”Maintenance scheduling problems as benchmarks for constraint algorithms” To appear in Annals of Math and AI.

[GJ79] M.R. Garey and D.S. Johnson, “Computers and intractability: a guide to the theory of NP-completeness”, W. H. Freeman, San Francisco, California, 1979.

[GK98] G. Karypis and V. Kumar, “hMeTiS - a hypergraph partitioning package”, http://www-users.cs.umn.edu/~karypis/metis/hmetis/files/manual.ps.

[Ha89] M. Hansen, “Approximation algorithms for geometric embeddings in the plane with applications to parallel processing problems,” 30th FOCS, pp. 604-609, 1989.

[HL99] Sung-Woo Hur and John Lillis, “Relaxation and clustering in a local search framework: application to linear placement”, Proceedings of the 36th ACM/IEEE conference on Design Automation Conference, pp. 360 - 366, 1999.

[K93] Richard M. Karp, “Mapping the genome: some combinatorial problems arising in molecular biology”, Proceedings of the twenty-
fifth annual ACM Symposium on Theory of Computing, pp. 278-285, 1993

[LR99] F.T. Leighton and S. Rao, “Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms”, Journal of the ACM, Volume 46, No. 6, pp. 787 - 832, Nov. 1999.

[P97] J. Petit, “Approximation Heuristics and Benchmarkings for the MinLA Problem”, Tech. Report LSI-97-41-R, Departament de Llenguatges i Sistemes Informàtics. Universitat Politècnica de Catalunya, http://wwwlsi.upc.es/dept/techreps/html/R97-41.html, 1997.

[RR98] S. Rao and A. W. Richa, “New approximation techniques for some ordering problems”, Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, 1998, pp. 211-218.

[RAK91] R. Ravi, A. Agrawal and P. Klein, “Ordering problems approximated: single processor scheduling and interval graph completion”, 18th ICALP, pp. 751-762, 1991.

[R70] D. J. Rose, “Triangulated graphs and the elimination process”, J. Math. Appl., 32 pp. 597-609, 1970.