1987

Planar Linear Arrangements of Outerplanar Graphs

Greg N. Frederickson
Purdue University, gnf@cs.purdue.edu

Susanne E. Hambrusch
Purdue University, seh@cs.purdue.edu

Report Number:

87-671

PLANAR LINEAR ARRANGEMENTS OF OUTERPLANAR GRAPHS

Greg N. Frederickson
Susanne E. Hambrusch

CSD-TR-671
March 1987

Planar Linear Arrangements of Outerplanar Graphs

Greg N. Frederickson and Susanne E. Hambrusch***

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907, USA.

March 1987

Abstract
Given an n-vertex outerplanar graph G, we consider the problem of arranging the vertices of G on a line such that no two edges cross and various cost measures are minimized. We present efficient algorithms for generating layouts in which every edge (i,j) of G does not exceed a given bandwidth $b(i,j)$, the total edge length and the cutwidth of the layout is minimized, respectively. We present characterizations of optimal layouts which are used by the algorithms. Our algorithms combine sublayouts by solving two processor scheduling problems. Although these scheduling problems are NP-complete in general, the instances generated by our algorithms are polynomial in n.

Keywords
Analysis of algorithms, bandwidth, cutwidth, layout problems, outerplanar graphs, scheduling problems.

* This work was supported by the Office of Naval Research under Contract N00014-86-K-0689 and by the National Science Foundation under Grant DCR-83-20124.
** This work was supported by the Office of Naval Research under Contracts N00014-84-K-0502 and N00014-86-K-0689, and by the National Science Foundation under Grant DMC-84-13496.

1. Introduction

The problem of arranging the vertices of an n-vertex, connected graph $G = (V,E)$ on a line subject to minimizing various cost measures is an interesting combinatorial problem arising in the area of VLSI layout design [F, OMKF, PDS, T]. Commonly considered cost measures are the bandwidth, the total edge length, and the cutwidth [C1, C2, CMST, L, S1, Y1]. In this paper we consider linear arrangement problems when no two edges of G are allowed to cross in the layout. The graphs that can be laid out under this assumption are exactly the outerplanar graphs [Y2]. We present efficient algorithms that minimize the bandwidth, the total edge length, and the cutwidth of a layout, respectively. A *layout* of G is a one-to-one mapping f of the vertices of G to the set $\{0,1, \cdots, n-1\}$. We next give the definition of the problems considered.

In the *Constant Bandwidth (CB) Problem* we are given G and an integer b and must determine if there exists a layout such that for every edge $(i,j) \in E$ $|f(i)-f(j)| \le b$. In the *Variable Bandwidth (VB) Problem* every edge (i,j) has its own bandwidth $b(i,j)$ and we must determine if there exists a layout such that for every edge (i,j) $|f(i)-f(j)| \le b(i,j)$. In both problems a layout should be generated if one exists. In the *Total-Edge-Length (TEL) Problem* we must determine the layout of G minimizing $\sum_{(i,j) \in E} |f(i)-f(j)|$. In the *Minimum Cutwidth (MC) Problem* we must determine a layout of G that minimizes the cutwidth. The *cutwidth* is defined as the maximum number of edges crossing any position $x+.5$ on the line, $x \in \{0,1, \cdots, n-1\}$.

In this paper we present an algorithm that solves the CB problem in $O(n^2)$ time using $O(n)$ space. For the VB and TEL problems we present algorithms that run in time $O(\delta^2 n + n^2)$ using $O(n)$ space, respectively, where δ is the number of biconnected components containing a central articulation point a^*. For the MC problem we present two algorithms: one running in time $O(n^3)$ and using $O(n)$ space, and one running in time $O(dn^2)$ and using $O(n^2)$ space, where d is maximum degree of any vertex representing an articulation point in the block-cutpoint tree of G.

The algorithms for all four problems are of a similar structure and they determine an optimal layout by sweeping through the block-cutpoint tree H of graph G. The *block-cutpoint tree* $H = (V_H, E_H)$ is defined as follows [H1]. Let $B_1, \cdots, B_k$ be the biconnected components of G and let $a_1, \cdots a_l$ be the articulation points in G. Then, $V_H = \{x_1, \cdots x_k, a_1, \cdots a_l\}$ and $E_H = \{(x_i, a_j) \mid a_j$ is a vertex in the biconnected component $B_i\}$. The algorithms differ in how they combine sublayouts during the sweep and in how they select the root of the block-cutpoint tree. Sublayouts are combined by solving two processor scheduling problems, with a different type of scheduling problem for each of bandwidth, edge-length, and cutwidth. Each one of the three scheduling problems that arises is NP-complete, but can be solved by a pseudo-polynomial time general algorithm which in turn yields an algorithm polynomial in n for the instances generated by our layout problems.

The selection of the root of the block-cutpoint tree for the CB, VB, and MC problems is closely related to the existence of a central biconnected component or a central articulation point in G. A central biconnected component (resp. articulation point) is a biconnected component (resp. vertex) for which the removal of its edges (resp. of the vertex) results in connected components with at most $n/2$ vertices each. The central biconnected component or articulation point can be determined in $O(n)$ time. For the CB problem we present a characterization of an optimal layout that allows us to select either the central biconnected component or the central articulation point. For the VB and TEL problems we present partial characterizations of an optimal layout. If G has a central biconnected component, the both problems can be solved in $O(n^2)$ time. Otherwise, the algorithms run in $O(\delta^2 n + n^2)$ time, where δ is the number of biconnected components containing the central vertex.

When considering non-planar layouts of general graphs all four problems are NP-complete [GJ]. The bandwidth problems are already NP-complete for trees [GGJK], but the total edge length and the minimum cut problem can be solved in polynomial time in the case of trees [C1, S1, Y1]. The TEL problem and the MC problem have recently shown to be NP-complete for 2-outerplanar graphs and outerplanar multigraphs in the case of non-planar layouts [MS, S2]. Planar layouts of trees minimizing the cutwidth are described in [DT, Y1].

The paper is organized as follows. In Section 2 we present layout characterizations for the CB, VB, and TEL problems, respectively. These characterizations will be used by our algorithms. Section 3 discusses the scheduling problems arising in the four layout problems and describes their solutions. Sections 4, 5, 6, and 7 present the algorithms for the CB, VB, TEL, and MC problems, respectively.

2. Layout Characterizations

In this section we present characterizations of optimal layouts for the CB, VB, and TEL problems which will be used by the algorithms. We start with some definitions used throughout the paper. We call a layout *feasible* if it satisfies the constraints imposed by the problem. A *central biconnected component* $B^* = (V^*, E^*)$ is a biconnected component for which the removal of the edges in E^* from G results in connected components of at most $n/2$ vertices each. A *central articulation point* a^* is an articulation point whose removal results in connected components of size at most $n/2$ vertices each. It is easy to show that a graph that has no central biconnected component has a central articulation point.

Let L be a planar layout of G. Edge (x,y) *dominates* edge (u,v) if $(x,y) \neq (u,v)$ and $f(x) \leq f(u) < f(v) \leq f(y)$. Let E' be a set of edges. We say (x,y) dominates E' if it dominates every edge of E'. We call an edge a *dominating* edge if it is not dominated by any other edge. We say that a vertex v is dominated by an edge (u,w) if $f(u) < f(v) < f(w)$. A vertex that is not dominated by any edge is called an *exposed* vertex.

The following lemma characterizes a feasible layout for the CB, VB, and TEL problems when the graph contains a central biconnected component B^*.

Lemma 2.1. Let G be an outerplanar graph that contains a central biconnected component B^*. If there exists a feasible layout for the CB, VB, and TEL problems, respectively, then there exists one in which an edge of the central biconnected component B^* is a dominating edge.

Proof. Let L be a feasible layout in which every edge of B^* is dominated by a least e edges, where e is a minimum among all feasible layouts. Suppose $e > 0$. We show how to transform L into another feasible layout L' such that one edge of B^* is dominated by at most $e-1$ edges.

Among the e edges dominating B^*, select the edge (u,v) with $f(u)$ a minimum. Note that this implies that the remaining $e-1$ edges dominating B^* are also dominated by (u,v). Let B be the biconnected component containing edge (u,v). It is not hard to show that the removal of all the edges of B that dominate B^* results in two connected components. To obtain from L a layout L' with one edge of B^* dominated by a most $e-1$ edges, remove all the edges of B that dominate B^*, interchange the layouts of the two resulting connected components, and restore the previously removed edges.

The only edge lengths that can change by this operation are of those edges that are removed and then restored. Since B^* is a central biconnected component, any edge (u',v') of B that is removed must have at least $n/2$ vertices between u' and v' in L. These vertices will not lie between u' and v' in L'. Thus there are at most $n/2$ vertices between u' and v' in L' and the length of any edge (u',v') of B did not increase. Any edge that did not dominate B^* in L will not dominate B^* in L'. Furthermore, at least one edge, namely (u,v), that dominated B^* in L will not dominate B^* in L'. Thus L' is another feasible layout in which one edge of B^* is dominated by at most $e-1$ edges, a contradiction to the choice of L. □

The next lemma gives a partial characterization of a feasible layout for the CB, VB, TEL problems when the graph contains no central biconnected component, only a central articulation point a^*. Let $B_1, B_2, \cdots, B_\delta$ be the biconnected components containing vertex a^*.

Lemma 2.2. Let G be an outerplanar graph that contains a central articulation point a^*. If there exists a feasible layout for the CB, VB, and TEL problems, respectively, then there exists one in which an edge of B_i, for some i, $1 \le i \le \delta$, is a dominating edge.

Proof. Let L be a feasible layout and let $B_i = (V_i, E_i)$ be a biconnected component containing an edge (x,y) such that (x,y) dominates every edge in E_j for all $j \ne i$. Assume edge (x,y) is dominated by e edges, where e is a minimum. Suppose $e > 0$. Then a layout L' can be obtained from L in a fashion analogous to that in the proof of Lemma 2.1. Layout L' will be feasible, and have at most $e-1$ edges that dominate a^*. This yields a contradiction. □

The following three lemmas fully characterize an optimal layout for the CB problem. This full characterization allows us to generate an $O(n^2)$ time algorithm for the CB problem. For the CB problem we do not need to work with graph G, but can consider the reduced graph G_r which is defined as follows. Let $B = (V',E')$ be a biconnected component of G, let $|V'|=k$, and let $L(B)$ be a layout of B. The *reduced biconnected component* of B is defined as $B_r = (V',E_r')$ where $E_r' = \{(x_i, x_{(i+1) mod k}) \mid f(x_i) = f(x_{(i+1) mod k})+1\}$. Note that B_r is obtained from B by deleting all the edges except for a cycle of length k. The *reduced graph* G_r is then obtained by replacing every biconnected component of G by its reduced biconnected component. The following lemma holds trivially for the CB problem.

Lemma 2.3. If there exists a feasible layout for the CB problem on graph G_r, then this layout is a feasible layout for the CB problem on graph G.

For the remainder of this section, we assume that G is a reduced graph. We next show that, if G does not contain a central biconnected component, only a central articulation point a^*, then a^* is exposed.

Lemma 2.4. Let G be an outerplanar graph that contains a central articulation point a^*. If there exists a feasible layout for the CB problem on G, then there exists one in which a^* is exposed.

Proof. Let L be a feasible layout in which a^* is dominated by e edges, where e is minimum among all feasible layouts. Suppose $e>0$.

Because of Lemma 2.2 all edges that dominate a^* belong to a biconnected component B that contains a^*. Let (u_1,u_2) be the longest edge of B in layout L. Let V' be all vertices in the same connected component as u_1 and u_2 when a^* is removed from G. Note that V' contains all vertices of B except a^*. Let n' be the cardinality of V'. Since a^* is a central articulation point, $n' \le n/2$. Thus the length of (u_1,u_2) is at least $n-n'+1 > n/2$. Now consider any layout L' in which all vertices in V' are to the left of a^*, and all vertices in $V-(V' \cup \{a^*\})$ are to the right of a^*. Since no edge dominates a^* in L', every edge will be of length at most $\max\{n', n-n'-1\}$ $< n-n'+1$. Thus L' is a feasible layout in which a^* is dominated by no edge, a contradiction. □

From Lemma 2.1 we know that if G contains a central biconnected component $B^* = (V^*, E^*)$, then one edge of B^* is dominating. The following lemma identifies this edge for the CB problem. Let $V^* = \{x_0, \cdots, x_{\alpha-1}\}$ and $E^* = \{(x_i, x_{i+1}) \mid 0 \le i < \alpha\}$. (The addition in the subscripts is done $mod\, \alpha$.) Let $n(x_i)$ be the number of vertices in the connected component containing vertex x_i after the edges (x_{i-1}, x_i) and (x_i, x_{i+1}) have been deleted from G, $0 \le i \le \alpha-1$.

Lemma 2.5. Let G be an outerplanar graph with a central biconnected component B^* and let $n(x_c) + n(x_{c+1}) \ge n(x_i) + n(x_{i+1})$ for all i. If there exists a feasible layout for the CB problem on G, then there exists one in which the edge (x_c, x_{c+1}) of B^* is dominating.

Proof. Let L be a solution in which the edge (x_i, x_{i+1}) of B^* is dominating. Suppose $i \ne c$. We show how to transform L into another layout L' in which the edge (x_c, x_{c+1}) is dominating.

We first obtain an intermediate layout $\hat{L}$. $\hat{L}$ is obtained from L by deleting the edges (x_c, x_{c+1}) and (x_i, x_{i+1}), interchanging the layouts of the two resulting connected components, and restoring the two previously removed edges. In layout $\hat{L}$ edge (x_c, x_{c+1}) is dominating, but the length of the edge may exceed bandwidth b. Such a situation is shown in Figure 2.1, where layouts L_1, L_2, L_5, and L_6 consist of $n/8 - 1$ vertices, and layouts L_3 and L_4 consist of $n/4 - 1$ vertices each. In $\hat{L}$ the edge (x_{c+1}, x_c) violates the bandwidth $b = 3n/4$. Note that the edge (x_c, x_{c+1}) is the only edge in $\hat{L}$ for which the length could have increased. (By an argument similar to the one given in the proof of Lemma 2.1, it can be shown that the length of the edge (x_i, x_{i+1}) did not increase.)

In order to obtain L' we perform the following operation on $\hat{L}$. W.l.o.g. assume that in L $f(x_{i+1}) < f(x_c) < f(x_{c+1}) < f(x_i)$. Move every layout of a subgraph of G formed by the removal of vertex x_{c+1} and whose layout is between $f(x_{c+1})$ and $f(x_{c+2})$ to the left of x_{c+1}. Analogously, move every layout of a subgraph of G formed by the removal of vertex x_c and whose layout is between $f(x_{c-1})$ and $f(x_c)$ to the right of x_c. The length of the edge (x_i, x_{i+1}) in L was at least $\sum_{j \neq i, i+1} n(x_j)$ and the length of the edge (x_c, x_{c+1}) in L' is $\sum_{j \neq c, c+1} n(x_j)$. Since $n(x_c) + n(x_{c+1}) > n(x_i) + n(x_{i+1})$, the edge (x_c, x_{c+1}) does not violate the bandwidth in L'. Furthermore, none of the other edges incident to x_c (resp. x_{c+1}) has a length of more than $n(x_c)$ (resp. $n(x_{c+1})$) in L'. $\square$

3. The Scheduling Problems

Our algorithms for all four layout problems will make use of algorithms for several scheduling problems. These involve scheduling tasks nonpreemptively on two processors, and are in general NP-hard, but all have pseudo-polynomial time algorithms when the relevant input values are integers. In the case of the particular scheduling problem instances generated by our layout problems, the running time of the scheduling algorithms will be strictly polynomial. In this section we present pseudo-polynomial time algorithms for the scheduling problems and show how to reduce the space used over that required in an obvious implementation. We now identify the different scheduling problems that we shall be considering.

In the *Deadline Scheduling Problem* we are given s tasks, with task i requiring time $T(i)$ and having a deadline $D(i)$ by which time it must be completed. We must determine if there is a feasible schedule of the tasks, and if there is one, must find among all feasible schedules a schedule that minimizes the total time on processor 1. One variation of this problem will be to report the set of different times on processor 1 for the set of feasible schedules. A second variation is to generate the schedule that realizes a particular time on processor 1.

In the *Cumulative Penalty Scheduling Problem* we are given s tasks, with task i requiring time $T(i)$ and having a delay penalty value of $P(i)$, which is multiplied times the start time of the task. We must determine the schedule with the minimum total penalty, i.e., $\sum_{i=1}^{s} P(i)*A(i)$, where $A(i)$ is the start time of the i-th task in the schedule. Again we consider variations of the problem. One variation is to report the set of different times on processor 1, along with the smallest cost for each. The second variation is, given a particular time on processor 1 realized by some feasible schedule, generate the minimum cost schedule for that time.

In the *Maximum Penalty Scheduling Problem* we are given s tasks, each task requiring unit time, an additive cost $W(i)$ and a dominating cost $C(i)$, where $W(i) \le C(i)$. A cost is determined for each processor, with the cost c of a processor reset to max $\{C(i), c + W(i)\}$ if task i is added to the processor following the previously scheduled tasks. We must determine a schedule that minimizes the maximum cost of the two processors. Once again we consider variations of the problem. One variation is to report the set of different costs on processor 1, along with the corresponding smallest cost on processor 2. The second variation is, given a particular cost on processor 1 realized by some feasible schedule, generate the minimum cost schedule for that cost.

As claimed, each of the above scheduling problems is NP-hard. The NP-completeness of deadline scheduling and cumulative penalty scheduling follows from [GJ], problems SS8 and SS13, respectively. The NP-hardness of maximum penalty scheduling follows from an easy transformation from partition.

Each of the above three problems has a special property that can be used in generating a dynamic programming algorithm. For the deadline scheduling problem, if there is a feasible schedule, then there is a feasible schedule in which the tasks on processor 1 are ordered in terms of nondecreasing deadline, and similarly for processor 2. For the cumulative penalty scheduling problem, an optimal schedule will have the tasks ordered on each processor by nondecreasing value of $T(i)/P(i)$. For the maximum penalty scheduling problem, an optimal schedule will have the tasks ordered on each processor by nondecreasing value of $C(i) - W(i)$. These properties can be established inductively using a simple interchange argument.

Thus a straightforward dynamic programming algorithm that solves any one of these problems need only consider the tasks in order of nondecreasing value of the relevant expression. For tasks $i = 1, \cdots, s$ so ordered, we determine a list K_i of different partial schedules of the first i tasks, with clearly inferior partial schedules eliminated. This list is typically ordered by a crucial value of a schedule, such as the time used so far on processor 1. List K_0 has one entry on it, representing the empty schedule. To generate list K_i given list K_{i-1}, do the following. Generate list K', the list of schedules created by taking each schedule in K_{i-1} and scheduling task i at the next available time on processor 1. Delete any schedule that is not feasible. Similarly, generate list K'' by scheduling task i at the next available time on processor 2 for each schedule in list K_{i-1}, and deleting any infeasible schedules. Then merge K' and K'' ordered on the crucial value, pruning any partial schedule that is clearly inferior to another partial schedule.

We now indicate how this strategy is instantiated for each of the scheduling problems. For deadline scheduling, we represent a partial schedule on list K_i by a pair (t, j) where t is the time scheduled on processor 1, and j is the highest indexed task on processor 1. If $j \neq 0$, then the preceding tasks on processor 1 can be determined by finding a pair $(t-T(j), j')$ on list K_{j-1}, resetting j to j', and repeating this until $j = 0$. Each list is ordered by value t. Let $S(i) = \sum_{k=1}^{i} T(k)$. To generate list K_i, for each pair (t, j) on list K_{i-1}, place pair $(t+T(i), i)$ on list K' if $t + T(i) \leq D(i)$, and place pair (t, j) on list K'' if $S(i) - t \leq D(i)$. When lists K' and K'' are merged to yield K_i, prune any pair (t, j) if there is a pair (t', j') preceding it with $t' = t$.

Let $N = \sum_{i=1}^{s} T(i)$. If the $T(i)$'s are integers, then the preceding algorithm runs in $O(sN)$ time and space. This approach solves all three versions of the deadline scheduling problem. We now discuss how to reduce the space. We handle the first variation of deadline scheduling, in which we are interested in determining only the set of different times on processor 1 for the set of feasible schedules, as follows. We do not keep the j values in the pairs, and do not maintain list K_{i-1} once list K_i is formed. Thus we can determine this set in $O(sN)$ time and $O(N)$ space.

Note that the above methods work in the claimed time and space bounds even if there are additional constraints to the problem that allow no task to be scheduled on processor 1 before a certain time, and no task to be scheduled on processor 2 before a certain time.

We can also solve the second variation of deadline scheduling, determining a feasible schedule that realizes any given time in the set of times on processor 1, using just $O(N)$ space. This is accomplished as follows. Use pairs (t, t_h) in the lists, where t_h is the time on processor 1 that is accounted for by tasks indexed $i = 1, 2, \cdots, \lfloor s/2 \rfloor$. In $O(sN)$ time and $O(N)$ space one can thus find the set of different times on processor 1 for the set of feasible schedules, and for each such time t a corresponding time t_h which is the portion of t accounted for by tasks indexed up to $\lfloor s/2 \rfloor$ in some feasible schedule with time t on processor 1. The problem then reduces to solving two subproblems. First, use the approach recursively to identify a feasible schedule of the first $\lfloor s/2 \rfloor$ tasks with time t_h on processor 1. Let $N_h = \sum_{i=1}^{\lfloor s/2 \rfloor} T(i)$. Then use the approach recursively to identify a feasible schedule of the remaining $\lceil s/2 \rceil$ tasks with time t on processor 1, given that no task on processor 1 can be scheduled before time t_h, and no task on processor 2 can be scheduled before time $N_h - t_h$. The approach used for achieving $O(N)$ space is reminiscent to the one used in [H2].

Lemma 3.1. The deadline scheduling problem and its two variations can be solved in $O(sN)$ time and $O(N)$ space.

Proof. By arguments presented above, the first variation can be solved in the claimed time and space bounds.

The algorithm for the second variation can be shown to take $O(N)$ space, by a simple inductive argument. The time can be shown to be $O(sN)$ as follows. Let $T(s, N)$ be the time to solve a deadline schedule problem of s tasks with total processing time N by the above method. Then for $s = 1$, $T(s, N) \le c$, and for $s > 1$,

$$T(s, N) \le csN + T(\lfloor s/2 \rfloor, N_h) + T(\lceil s/2 \rceil, N - N_h)$$

It follows that $T(s, N) \le c(2s-1)N$.

The original version of the problem can be solved by solving the first variation, selecting the smallest time on processor 1, and then solving the second variation. □

We next discuss how the space-saving strategy is instantiated for cumulative penalty scheduling. We represent a partial schedule on list K_i by a triple (t, c, j) where t is the time scheduled on processor 1, c is the cost of the partial schedule, and j is the highest indexed task on processor 1. Each list is ordered by value t. Recall that $S(i) = \sum_{k=1}^{i} T(k)$. To generate list K_i, for each triple (t, c, j) on list K_{i-1}, place triple $(t+T(i), c + P(i)*t, i)$ on list K', and place triple $(t, c + P(i)*(S(i)-t), j)$ on list K''. When lists K' and K'' are merged to yield K_i, prune any triple (t, c, j) if there is a triple (t', c', j') preceding it with $t' = t$ and $c' \le c$.

As in deadline scheduling, there is a good pseudopolynomial time implementation of the algorithm for cumulative penalty scheduling. Again let $N = \sum_{i=1}^{s} T(i)$. If the $T(i)$'s are integers, then the algorithm as stated runs in $O(sN)$ time and space. We handle the first variation of cumulative penalty scheduling, in which we are interested in determining only the set of different times on processor 1 for the set of feasible schedules, along with the smallest cost for each time, as follows. We do not keep the j values in the triples, and do not maintain list K_{i-1} once list K_i is formed. Thus we can determine this set in $O(sN)$ time and $O(N)$ space. Note that the above methods work even if there are additional constraints to the problem that allow no task to be scheduled on processor 1 before a certain time, and no task to be scheduled on processor 2 before a certain time.

We can also solve the second variation of cumulative penalty scheduling, determining a feasible schedule that realizes any given time in the set of times on processor 1, and is of minimum cost for that time, using just $O(N)$ space. This is accomplished by using the divide-and-conquer technique discussed for deadline scheduling. Use triples (t, t_h, c) in the lists, where t_h is the time on processor 1 that is accounted for by tasks indexed $i = 1, 2, \cdots, \lfloor s/2 \rfloor$.

Lemma 3.2. The cumulative penalty scheduling problem and its two variations can be solved in $O(sN)$ time and $O(N)$ space.

Proof. The proof is similar to that of Lemma 3.1. □

We next discuss how the space-saving strategy is instantiated for maximum penalty scheduling. We represent a partial schedule on list K_i by a triple (c_1, c_2, j) where c_1 is the cost for processor 1, c_2 is the cost for processor 2, and j is the highest indexed task on processor 1. Each list is ordered by value c_1. To generate list K_i, for each triple (c_1, c_2, j) on list K_{i-1}, place triple $(\max\{C(i), W(i) + c_1\}, c_2, i)$ on list K', and place triple $(c_1, \max\{C(i), W(i) + c_2\}, j)$ on list K''. When lists K' and K'' are merged to yield K_i, prune any triple (c_1, c_2, j) if there is a triple (c_1', c_2', j') preceding it with $c_1' \le c_1$ and $c_2' \le c_2$.

For maximum penalty scheduling we do the following. Let $C = \sum_{i=1}^{s} C(i)$. If the $C(i)$'s are integers, then the algorithm as stated runs in $O(sC)$ time and space. We handle the first variation of maximum penalty scheduling, in which we are interested in determining only the set of different costs on processor 1, along with the corresponding smallest cost on processor 2, as follows. We do not keep the j values in the triples, and do not maintain list K_{i-1} once list K_i is formed. Thus we can determine this set in $O(sC)$ time and $O(C)$ space. Note that the above methods work even if there are additional constraints to the problem such as an initial nonzero cost on processor 1, and an initial nonzero cost on processor 2.

We can also solve the second variation of maximum penalty scheduling, determining a feasible schedule that realizes any given cost in the set of costs on processor 1, and is of minimum cost on processor 2, using just $O(N)$ space. This is accomplished by using the divide-and-conquer technique discussed for deadline scheduling. Use triples $(c_1, c_{1,h}, c_2)$ in the lists, where $c_{1,h}$ is the cost on processor 1 that is accounted for by tasks indexed $i = 1, 2, \cdots, \lfloor s/2 \rfloor$.

Lemma 3.3. The maximum penalty scheduling problem and its two variations can be solved in $O(sC)$ time and $O(C)$ space.

Proof. The proof is similar to that of Lemma 3.1. □

4. The Constant Bandwidth Problem

In this section we present an algorithm that solves the CB problem for a given outerplanar graph G in $O(n^2)$ time using $O(n)$ space. We assume that G has been reduced; i.e., every biconnected component consisting of α vertices, $\alpha>2$, contains exactly α edges. The algorithm starts by constructing the block-cutpoint tree H of G. It then determines whether G contains a central biconnected component B^* or only a central articulation point a^*, and roots H at either B^* or a^*. The layout of G is determined by sweeping up through H from the leaves towards the root. During the sweep, at every vertex x representing a biconnected component B_x, sublayouts of x's grand-children are combined with the layout of the vertices in B_x. The order of the sublayouts is determined by solving a deadline scheduling problem.

We first describe the algorithm for the case when G contains a central biconnected component B^*. For every vertex x in the block-cutpoint tree H let $p(x)$ be its parent. For every vertex x representing a biconnected component B_x let $L(x)$ be the layout of the subgraph of G induced by the vertices represented in the subtree rooted at vertex x. If $B_x \neq B^*$, vertex $p(x)$, which is an articulation point, is also included in $L(x)$ since it is represented in the biconnected component B_x. Note that all the edges between vertex $p(x)$ and another vertex in $L(x)$ are edges in B_x. Since $L(x)$ will, at some later step, be combined with other sublayouts containing vertex $p(x)$ and an edge encountered later has to be dominating, it is necessary to have vertex $p(x)$ assigned to a corner position in layout $L(x)$. W.l.o.g., we assign $p(x)$ to the rightmost position. Hence, in layout $L(x)$, one edge incident to $p(x)$ will be a dominating edge in $L(x)$. Two values, *length*(x) and *slack*(x), are associated with $L(x)$. Let *length*(x) be the number of vertices in the layout $L(x)$ minus 1, and let *slack*(x) be the maximum distance the length of the dominating edge incident to $p(x)$ can be increased in future steps (which equals the maximum number of additional vertices that can be dominated).

We call a biconnected component *trivial* if it consists of a single edge. When sweeping up through the block-cutpoint tree no action is taken for any vertex corresponding to an articulation point unless it is the root. When vertex x corresponds to a biconnected components one of 4 cases can occur:

(1) Vertex x is a leaf

(2) Vertex x is neither a leaf nor the root, and corresponds to a trivial biconnected component B_x

(3) Vertex x is neither a leaf nor the root, and corresponds to a non-trivial biconnected component B_x

(4) Vertex x is the root

Throughout, let the children of x in H be $a_1, \cdots, a_k$, and the children of a_j be $x_{j,1}, \cdots x_{j,m_j}$,

$1 \le j \le k$. Let the vertices in the biconnected component B_x in G be $y_0, \cdots, y_{\alpha-1}$. See Figure 4.1.

Case (1). If vertex x is a leaf, it is handled as follows. We set $length(x)$ to the number of vertices in B_x minus 1, $slack(x)$ to the bandwidth b minus $length(x)$, and $L(x)$ to any layout that has one of the two edges incident to vertex $p(x)$ as a dominating edge.

Case (2). If x is neither a leaf nor the root, and corresponds to a trivial biconnected component B_x, then $k=1$ and B_x consists of the edge $(a_1, p(x))$ with $a_1=y_0$ and $p(x)=y_1$. We set up and solve a deadline scheduling problem to determine which sublayouts will be dominated by the edge $(a_1, p(x))$, where vertex $p(x)$ will be at the rightmost position. The deadline scheduling problem consists of m_1 tasks, where $T(i) = length(x_{1,i})$ and $D(i) = length(x_{1,i}) + slack(x_{1,i})$, for $i = 1, \cdots, m_1$. In the solution to the deadline scheduling problem, which minimizes the total time on processor 1, let the tasks scheduled on processor 1 be $i_1, \cdots, i_q$ and the ones on processor 2 be $j_1, \cdots, j_r$, $q+r=m_1$. Let t_1 be the time on processor 1. Set $length(x) = \sum_{i=1}^{m_1} length(x_{1,i}) - m_i + 2$ and $slack(x) = b - t_1$. Let $L^-(x_{1,i})$ be the layout obtained from $L(x_{1,i})$ by removing vertex y_0 (i.e., the rightmost vertex). Let $L^{r-}(x_{1,i})$ be the layout obtained from $L(x_{1,i})$ by reversing the order of the vertices and removing vertex y_0. Then, layout $L(x)$ is formed by the concatenation of

$$L^{r-}(x_{1,j_r}), \cdots, L^{r-}(x_{1,j_1}), y_0, L^-(x_{1,i_1}), \cdots, L^-(x_{1,i_q}), y_1.$$

Case (3). If x is neither a leaf nor a root, and it corresponds to a non-trivial biconnected component B_x, then vertex $p(x)$ is assigned the rightmost position and there are two choices for a dominating edge. Both choices are considered, and the one resulting in a larger slack is chosen. Assume $p(x)$ corresponds to articulation point y_c in the biconnected component B_x. First determine the slack of the layout L' that has the edge (y_{c+1}, y_c) as a dominating edge. If vertex y_{c+1} is an articulation point in G, let its corresponding vertex in H be a_j. In this case a deadline scheduling problem on m_j tasks is set up, where $T(i) = length(x_{j,i})$ and $D(i) = length(x_{j,i}) + slack(x_{j,i})$ for $i=1,2, \cdots, m_j$. Note that in L' it does not matter whether the layouts associated with the children of vertex a_r in H, $r \ne j$, are placed to the left or to the right of vertex a_r. We determine the slack of layout L' from t_1, the minimum time needed on processor 1 for the deadline scheduling problem set up for a_j, and the lengths of the layouts associated with the other grand-children of x.

In a similar fashion determine the slack of the layout L'' that has edge (y_{c-1}, y_c) as a dominating edge. From L' and L'' choose the layout that yields a larger slack, and determine *length* (x), *slack* (x), and $L(x)$.

Case (4). When x is the root of H (i.e., $B_x = B^*$) we determine the final layout. Instead of choosing $p(x)$ as the rightmost vertex of $L(x)$, the rightmost vertex in this case is determined by Lemma 2.5. Let y_c and y_{c+1} be the vertices satisfying $n(y_c)+n(y_{c+1}) \geq n(y_i)+n(y_{i+1})$ for all $0 \leq i < \alpha$. The layout $L(x)$ is obtained by solving two deadline scheduling problems, one for y_c and one for y_{c+1}. Assume w.l.o.g. that y_c is made the rightmost vertex in $L(x)$. Then the layouts of y_c's grand-children corresponding to the tasks scheduled on processor 1 (resp. 2) are placed to the left (resp. right) of y_c. The positioning of layouts of y_{c+1}'s grand-children is done in the reverse order. Layout $L(x)$ is returned as the final layout.

This completes the description of the algorithm for the CB problem when G contains a central biconnected component. Its correctness follows from the lemmas of Section 2 and the discussion. Assume now that H is rooted at the central articulation point a^*. Let $x_1, \cdots, x_\delta$ be the children of a^* in H which represent the biconnected components $B_1, \cdots, B_\delta$ containing vertex a^*. Cases (1) to (3) remain as described above and Case (4) is handled as follows. We set up and solve a deadline scheduling problem on δ tasks using *length* (x_i) and *slack* (x_i), $1 \leq i \leq \delta$. Any feasible solution to the scheduling problem represents a final layout of graph G.

Theorem 4.1. The algorithm determining a feasible layout for the CB problem runs in $O(n^2)$ time and uses $O(n)$ space.

Proof. All the preprocessing steps (i.e., reducing G, creating the block-cutpoint tree H, determining B^* or a^*) can be done in $O(n)$ time. Consider a vertex x in H representing a biconnected component. Let n' be *length* (x), i.e., it is the number of vertices in biconnected components represented by vertices in the subtree rooted at x. For case (1), the algorithm takes $O(n')$ time. For cases (2), (3) and (4), the algorithm takes $O(n')$ time plus the time to solve at most two scheduling problems. By Lemma 3.1, solving one deadline scheduling problem for s sublayouts which contain a total of N vertices of G uses $O(sN)$ time. Thus a scheduling problem at an articulation point a_j takes $O(d(a_j)n(a_j))$ time, where $d(a_j)$ is the in-degree of a_j in H and $n(a_j)$ is the number of vertices in the biconnected components represented by vertices in the subtree rooted at a_j. Let $T(n')$ be the time used by our algorithm for computing the layout $L(x)$. Recall that vertex $x_{j,i}$ is the i-th child of the j-th child of vertex x in H. Then, $T(n')$ is described by the recurrence relation: $T(n') \leq c(n' + \max_{1 \leq j \leq k} d(a_j)n(a_j)) + \sum_{j=1}^{k} \sum_{i=1}^{m_j} T(n(x_{j,i}))$, where c is a constant. The last term reflects the time to solve subproblems for the grandchildren of x. It is easy to see that $T(n') \leq cn'(1 + \sum_{a \in A_H(x)} d(a))$, where $A_H(x)$ is the set of articulation points in the subtree of H rooted at x. Since the total degree of all articulation points in H is $O(n)$, the $O(n^2)$ time bound follows. The $O(n)$ space bound follows from Lemma 3.1. □

5. The Variable Bandwidth Problem

In this section we describe how to modify and extend the algorithm given for the CB problem to solve the variable bandwidth problem. The algorithm for the VB problem can no longer work with the reduced graph G_r since the bandwidth of a removed edge can have an influence on the layout. We call an edge (i,j) of G an *inner edge* if $(i,j) \notin G_r$, and an *outer edge* if $(i,j) \in G_r$. If G contains a central biconnected component B^*, then Lemma 2.1 states that an outer edge of B^* will be a dominating edge. Unlike the CB problem, every outer edge of B^* is now a potential candidate. If a B^* exists, our algorithm determines a feasible layout in $O(n^2)$ time using $O(n)$ space. If only a central articulation point a^* exists, our algorithm determines a feasible layout in $O(\delta^2 n + n^2)$ time using $O(n)$ space, where δ is the number of biconnected components containing a^*. From Lemma 2.2 we know that an outer edge of a biconnected component containing a^* will be dominating, and considering all δ biconnected components causes the increase in the time.

We first describe the algorithm for the case when a central biconnected component B^* exists. As is done for the CB problem, the algorithm constructs the block-cutpoint tree H, finds B^*, and makes B^* the root of H. Let $b(i,j)$ be the bandwidth of edge (i,j). If edge $(i,j) \notin G$, we set $b(i,j) = \infty$. The layout of G is constructed by sweeping up through H. Every vertex x of H corresponding to a biconnected component B_x has the three entries, *length*(x), *slack*(x), and $L(x)$ as defined in the previous section, associated with it. Let the vertices of B_x in G be

$y_0, \cdots, y_{\alpha-1}$ such that edges (y_i, y_{i+1}) are the outer edges.

Case (1). The leaves of H are handled as follows. As in the in the CB algorithm, we set *length* (x) to the number of vertices in B_x minus 1 (i.e., to $\alpha-1$). If B_x is a trivial biconnected component consisting of the vertices y_0 and y_1 with $y_1 = p(x)$, set *slack* (x) to $b(y_0, y_1) - 1$ and $L(x)$ to the layout consisting of vertex y_0 followed by $y_1 = p(x)$.

If B_x is a non-trivial biconnected component, let $p(x)$ correspond to articulation point y_c in G. We now have two choices for the dominating edge, (y_{c-1}, y_c) or (y_{c+1}, y_c). If either layout contains an edge whose length exceeds its bandwidth, the layout is discarded. If both layouts are feasible, then let

$$slack' = \min \{ b(y_{c-i}, y_c) - length(x) + i - 1 \mid 1 \le i \le \alpha-1 \},$$
$$slack'' = \min \{ b(y_{c+i}, y_c) - length(x) + i - 1 \mid 1 \le i \le \alpha-1 \},$$

and $slack(x) = \max \{ slack', slack'' \}$. Set $L(x)$ to the layout yielding the maximum value of $slack(x)$.

Case (2). If x is neither a leaf nor the root, and corresponds to a trivial biconnected component, we proceed in a fashion similar to that in case (2) of the algorithm for the CB problem. The one exception is that $slack(x)$ is set to $b(y_0, y_1) - t_1$.

Case (3). Assume x is neither a leaf nor the root, and corresponds to a non-trivial biconnected component B_x. Recall that $a_1, \cdots, a_k$ are the children of x in H and $y_0, \cdots y_{\alpha-1}$ are the vertices in B_x, $k < \alpha$. For every child a_j of x we solve a deadline scheduling problem of the first variation, choose a particular feasible time on processor 1, and solve a deadline scheduling problem of the second variation for it. The main idea is to construct $L(x)$ in a left to right fashion and to position the sublayouts of every vertex y_i as much as possible to the left of y_i.

As in case (3) of the algorithm for the CB problem, we have two choices for the dominating edge. We discuss how to determine only layout L', the layout that has the edge (y_{c+1}, y_c) as the dominating edge, since determining the layout L'' that has the edge (y_{c-1}, y_c) dominating is analogous. Assume w.l.o.g. that vertex $p(x)$ corresponds to vertex $y_{\alpha-1}$ in G. Hence, the vertices of B_x will be positioned in the order $y_0, y_1, \cdots, y_{\alpha-1} = p(x)$ and when generating layout L' the vertices of B_x are considered in this order. If vertex y_0 corresponds to an articulation point, set up and solve a deadline scheduling problem minimizing the time on processor 1 as done for the CB problem.

Assume we have handled vertices $y_0, \cdots, y_{i-1}$ of B_x. Let vertex y_i correspond to vertex a_j in H. We set up and solve a deadline scheduling problem of the first variation for a_j using the *length* and *slack* values of a_j's children. We next determine, $maxleft_i$, the maximum number of additional vertices that can be placed between y_{i-1} and y_i. $Maxleft_i$ is computed by first determining for every edge (y_r, y_i) with $r<i$ the number of additional vertices that can be dominated by this edge without exceeding its bandwidth. Let this value be $eslack(y_r, y_i)$. Let $L(x)$ be the layout that is being built incrementally, let f be the mapping for layout $L(x)$, and let vertex u be the most recently placed vertex in $L(x)$; i.e., u was placed when y_{i-1} was handled and $f(u)$ is maximum. Then, $eslack(y_r, y_i)$ is set to $b(y_r, y_i) - f(u) + f(y_r) - 1$, and $maxleft_i$ is set to the minimum over all $eslack(y_r, y_i)$ values. Knowing $maxleft_i$, we choose among all feasible solutions to the scheduling problem for a_j the one using time t_1 on processor 1 such that t_1 is a minimum and $t_1 \geq n(y_j) - maxleft_i$. Recall that $n(y_i) = n(a_j)$ is the number of vertices in the connected component containing vertex $y_i = a_j$ after the edges of B_x have been deleted. We obtain the schedule associated with t_1 by solving a deadline scheduling problem of the second variation, and place all the sublayouts corresponding to tasks scheduled on processor 2 to the left of vertex y_i, and all the ones scheduled on processor 1 to the right of vertex y_i.

Case (4). When the root of H is considered the algorithm tries every outer edge in turn as the dominating edge. In order to do so, we first solve for every vertex in B^* corresponding to an articulation point a deadline scheduling problem of the first variation. This takes $O(\sum_{i=1}^{k} m_i n(a_i))$ $= O(n^2)$ time and the k lists obtained use a total of $O(\sum_{i=1}^{k} n(a_i)) = O(n)$ space. Determining whether a feasible layout with edge (y_c, y_{c+1}) as a dominating edge exists is done in $O(\alpha + \sum_{j=1}^{k} n(a_j))$ time similar as in case (3). If a feasible layout exists, we obtain the final layout by solving k deadline scheduling problems of the second variation. If no feasible layout exists, the edge (y_{c+1}, y_{c+2}) is considered next. In the worst case all α edges have to considered as dominating edges and $O(\alpha\,(\alpha + \sum_{j=1}^{k} n(a_j))) = O(n^2)$ time is used.

When G contains no central biconnected component B^*, we first root the block-cutpoint tree H at the central articulation point a^*. We determine whether a feasible layout with a^* as an exposed exists and while doing so we also compute results to scheduling problems used again at a later step. Let $x_1, \cdots, x_\delta$ be the children of a^* in H which correspond to the δ biconnected components $B_1, \cdots, B_\delta$ containing a^*. The sweep through the tree towards a^* keeps at every x_i the lists generated when deadline scheduling problems of the first variation are solved at x_i. If B_i contains k_i articulation points, then $k_i - 1$ lists are generated for x_i. The total space needed to store all $\sum_{j=1}^{\delta} k_j$ lists is $O(n)$. Assume no feasible layout with a^* as an exposed vertex exists. We then try every vertex x_i as the root of the block-cutpoint tree, $1 \le i \le \delta$. Obviously, no action needs to be taken if x_i corresponds to a trivial biconnected component. Assume we have unsuccessfully tried $x_1, \cdots, x_{i-1}$. Then x_i is handled as follows. We solve one scheduling problem on $\delta - 1$ tasks using $length(x_j)$ and $slack(x_j)$, $1 \le j \le \delta$, $j \ne i$. We then try every outer edge of the biconnected component B_i as a dominating edge as done in Case (4). This step makes use of the $k_i - 1$ lists stored at x_i. Let α_i be the number of vertices in B_i, $\alpha_i \le k_i$. Then, trying one edge of B_i as dominating takes $O(\alpha_i + n)$ time and all edges take $O(\alpha_i^2 + \alpha_i n)$ time.

In the worst case we need to consider δ non-trivial biconnected components. We have then spent $O(\delta^2 n)$ time on solving a total of δ scheduling problems and $O(\sum_{i=1}^{\delta} \alpha_i + n\alpha_i) = O(n^2)$ time on trying edges as dominating edges. Hence, if G contains only a central articulation point a^*, a feasible layout can be determined in time $T(n) + O(\delta^2 n + n^2)$, where $T(n)$ is the time for initial solution with a^* exposed.

Theorem 5.1. The algorithm determining a feasible layout for the VB problem runs in $O(n^2)$ time if G contains a central biconnected component B^* and in $O(\delta^2 n + n^2)$ time if G contains no B^*, and it uses $O(n)$ space.

Proof. Let x be a non-root vertex in H representing a biconnected component, let n' be $length(x)$, and let $a_1, \cdots, a_k$ be the children of x. For case (1), the algorithm takes $O(n')$ time. For cases (2) and (3) the algorithm takes $O(n')$ time plus the time to solve k scheduling problems, where a scheduling problem at an articulation point a_j takes $O(d(a_j)n(a_j))$ time. Let $T(n')$ be the time used by our algorithm for computing the layout $L(x)$. Then, $T(n')$ is described by the recurrence relation: $T(n') \le c(n' + \sum_{j=1}^{k} d(a_j)n(a_j)) + \sum_{j=1}^{k} \sum_{i=1}^{m_j} T(n(x_{j,i}))$, where c is a constant. It is easy to see that $T(n') \le cn'(1 + \sum_{a \in A_H(x)} d(a))$, where $A_H(x)$ is the set of articulation points in the subtree of H rooted at x. Since the total degree of all articulation points in H is $O(n)$ and case (4) takes $O(n^2)$ time, the algorithm takes $O(n^2)$ time when G contains a central biconnected component. The time bound of $O(\delta^2 n + n^2)$ for the case when G contains only a central articulation point a^* also follows from the above discussion. The $O(n)$ space bounds follows from Lemma 3.1. □

6. The Total-Edge-Length Problem

In this section we show how to obtain in $O(\delta^2 n + n^2)$ time a layout minimizing the total edge length, where δ is the number of biconnected components containing the central vertex a^*. The algorithm will again sweep up through the block-cutpoint tree H towards the root, either B^* or a^*. In the case when a^* is the root, the algoruthm will consider, as done in the VB problem, all children of a^* as possible roots. Every vertex x of H corresponding to a biconnected component has two entries, *length*(x) and $L(x)$, associated with it. Sublayouts are combined during the sweep by solving cumulative penalty scheduling problems. Consider, for example, case (2). To determine which sublayouts to place to the left and to the right of vertex a_1, the degree of vertex a_1 in any biconnected component corresponding to a child of a_1 in H is the delay penalty for the corresponding task. Minimizing the total edge length corresponds to minimizing the penalty in the scheduling problem. The four cases encountered in the sweep when B^* is the root are now as follows.

Case (1). For every leaf x of H we set *length*(x) to the number of vertices in the biconnected component B_x minus 1. If B_x is a non-trivial biconnected component, there are two choices for the layout $L(x)$ and we choose the one with the smaller total edge length.

Case (2). For every non-leaf, non-root vertex x corresponding to a trivial biconnected component we set up and solve a cumulative penalty scheduling problem of the first variation. We set $T(i)$ to *length*$(x_{1,i})$ and $P(i)$ to the degree of vertex a_1 in the biconnected component $B_{x_{1,i}}$. The list returned by the algorithm consists of the pairs (t,c), with every pair representing a solution of cost c using time t on processor 1. From this list, which is of length at most n_1, we select the entry (t_1,c) such that $c+t_1$ is a minimum. The term t_1 added corresponds to the length the edge $(a_1, p(x))$ is stretched. We then obtain the schedule associated with entry (t_1,c). The schedule determines which of the layouts $L(x_{1,i})$, $i=1,2, \cdots, m_1$, will be dominated by the edge $(a_1,p(x))$ and the order of the sublayouts.

Case (3). For every non-leaf, non-root vertex x corresponding to a non-trivial biconnected component there are again two choices for the dominating edge. As done in case (3) in the algorithm for the VB problem, we assume $p(x) = y_{\alpha-1}$ and describe how to obtain the layout that has edge $(y_0,y_{\alpha-1})$ as the dominating edge. Assume vertices $y_0, \cdots, y_{i-1}$ have been considered and a cumulative penalty scheduling problem has been solved if they correspond to an articulation point. Let vertex y_i of B_x correspond to articulation point a_j. Then a scheduling problem of the first variation is set up and solved where $T(i)$ is set to $length(x_{j,s})$ and $P(s)$ to the degree of vertex a_j in $B_{x_{j,s}}$, $s = 1, \cdots ,m_j$. Let p_1 be equal to the number of edges (y_i,y_r) in B_x with $r>i$, and p_2 equal to the number of edges (y_r,y_i) in B_x with $r<i$. From the list generated by the scheduling algorithm we determine, in $O(n_j)$ time, the entry (t_1,c) such that $c + p_1t_1 + p_2(n_j-t_1)$ is a minimum. The schedule associated with this entry determines $L(x)$.

Case (4). For the root of H we consider every outer edge in turn as a dominating edge and choose the one resulting in the smallest total edge length. In order to determine this edge we first set up and solve k scheduling problems of the first variation, one for every articulation point in B^*. Using the k lists generated by the scheduling problems, the time needed for determining the total edge length of the layout that has edge (y_c,y_{c+1}) as a dominating edge is $O(\alpha + \sum_{i=1}^{k} n_i)$ with $\sum_{i=1}^{k} n_i \le n$. Hence, step (4) runs in time $O(\alpha\,(\alpha + \sum_{i=1}^{k} n_i)) = O(n^2)$.

When graph G contains no central biconnected component B^*, we root H at a^*, determine the minimum total edge length for any layout that has a^* exposed, and then try every non-trivial biconnected component containing a^* as a root. The $O(\delta^2 n + n^2)$ time bound for this step is obtained as in the VB problem.

It can be shown that in certain situations not all non-trivial biconnected components containing vertex a^* need to be considered as a root. Let B be a non-trivial biconnected component containing k articulation points, $a^*=a_1, a_2, \cdots, a_k$, and let the degree of vertex a^* in B be two. Then an optimal layout having an edge of B dominating has either vertex a^* exposed or a vertex a_j exposed, where $n(a_j) \geq n/8$. Recall that $n(a_j)$ is the number of vertices in the connected component containing a_j after all edges of B have been removed. This implies that, if the degree of a^* is three or more in a constant number of biconnected components, then the running time of our algorithm is $O(n^2)$. The proof of this layout characterization is along the lines of the proofs given in Section 2. Since it does not improve the worst case time complexity, it is omitted.

Theorem 6.1. The algorithm determining a feasible layout for the TEL problem runs in $O(n^2)$ time if G contains a central biconnected component B^* and in $O(\delta^2 n + n^2)$ time if G contains no B^*, and it uses $O(n)$ space.

7. The Minimum Cutwidth Problem

In this section we present two algorithms for determining a layout of G minimizing the cutwidth. The first one runs in $O(n^3)$ time and uses $O(n)$ space, and the second one runs in $O(dn^2)$ time and uses $O(n^2)$ space, where d is the maximum degree of a vertex representing an articulation point in the block-cutpoint tree H of G. Both algorithms for the MC problem work again with the block-cutpoint tree H. The scheduling problem solved when determining how to combine sublayouts is the maximum penalty scheduling problem.

Assume the block-cutpoint tree H is rooted at some vertex r which corresponds to a biconnected component B_r. Using an algorithm similar to the ones described in the previous section, we can determine in $O(n^2)$ time the minimum cutwidth achieved by a layout that has one edge of B_r dominating. When sweeping up through H towards r, every vertex x corresponding to a biconnected component has, in addition to layout $L(x)$, a weight entry $w(x)$ and a cutwidth entry $c(x)$ associated with it. The entry $w(x)$ corresponds to the degree of vertex $p(x)$ in the biconnected component B_x and represents the additive cost in the scheduling problem. Entry $c(x)$ corresponds to the maximum cutwidth in layout $L(x)$ and it represents the dominating cost. Obviously, $c(x) \geq w(x)$.

Assume vertex y_i of biconnected component B_x is handled in step (3) of the algorithm and y_i corresponds to articulation point a_j in H. We set up and solve a maximum penalty scheduling problem of the first variation using the weight and the cutwidth entries of a_j's children. The list returned by the scheduling algorithm contains the pairs (c_1,c_2) as discussed in Section 2. Let p_1 (resp. p_2) be the number of edges (y_r,y_i) in B_x with $r>i$ (resp. $r<i$). Then the schedule determining the layout is the one associated with the entry (c_1,c_2) in the list for which the maximum of c_1+p_1 and c_2+p_2 is a minimum. The other steps of the algorithm are modified accordingly. Considering every biconnected component as the root of H results in an $O(n^3)$ time algorithm which uses $O(n)$ space.

We next describe an algorithm that solves the MC problem in $O(dn^2)$ time using $O(n^2)$ space. The algorithm first chooses an arbitrary vertex r corresponding to a biconnected component as the root. We then run the $O(n^2)$ time algorithm described above, saving the lists generated for every instance of a scheduling problem and not determining the actual layout (i.e., no scheduling problems of the second variation are solved). We obtain the optimal layout of G by traversing H rooted at r and computing at every node x the minimum cutwidth of any layout having an edge of B_x dominating. This strategy corresponds to rerooting H at every vertex and taking advantage of previously computed sublayouts. Once we know the vertex r^* of H such that one edge of B_{r^*} is dominating in the optimal layout, the final layout is obtained by solving the problem rooted at r^*.

Assume that vertex x has been just been considered as the root and vertex $x_{1,1}$, a grandchild of x, is considered next. (See Figure 4.1 for the naming of the vertices.) From the computations already done for vertex x we obtain the new $w(x)$ and $c(x)$. The new layout $L(x)$ (which is actually not determined) has articulation point a_1 at the rightmost position and does not contain layouts $L(x_{1,2}), \cdots, L(x_{1,m_1})$. We next solve one maximum penalty scheduling problem of the first variation; namely the one set up from the entries $w(x), w(x_{1,2}), \cdots, w(x_{1,m_1})$ and $c(x), c(x_{1,2}), \cdots, c(x_{1,m_1})$. Solving this problem costs $O(d(\sum_{j=2}^{m_1} c(x_{1,j})+c(x)))$ time. The list obtained from this scheduling problem and the m_1-1 lists associated with vertices $x_{1,2}, \cdots, x_{1,m_1}$ contain a total of $O(n)$ entries. From these m_1 lists we determine the best layout that has one edge of $B_{x_{1,1}}$ as a dominating edge. Assuming $B_{x_{1,1}}$ consists of $\alpha_{1,1}$ vertices, this step can be done in $O(n\alpha_{1,1})$ time. When vertices $x_{1,2}, \cdots, x_{1,m_1}$ are considered as roots at some later time in the algorithm, a scheduling problem consisting of m_1 tasks is set up and solved each time.

The total time spent on solving scheduling problems is $O(n \sum_{a \in A_H} d(a)^2) = O(dn^2)$, where A_H is the set of articulation points in H and d is the maximum degree of any vertex representing an articulation point. The time spent on determining the best layout that has one edge of a biconnected component B_x as a dominating edge is $O(n\alpha_x)$, where α_x in the number of vertices in B_x. Hence, the total time spent on this part of the algorithm is $O(n \sum_x \alpha_x) = O(n^2)$. We conclude with the following theorem.

Theorem 7.1. The minimum cutwidth problem can be solved in $O(n^3)$ time using $O(n)$ space or in $O(dn^2)$ time using $O(n^2)$ space, where d is the maximum degree of any vertex representing an articulation point in the block-cutpoint tree H of G.

References

[C1] F.R.K. Chung, 'On Optimal Linear Arrangements of Trees', *Comp. & Maths. with Appls.*, Vol. 10, No. 1, pp 43-60, 1984.

[C2] F.R.K. Chung, 'On the Cutwidth and the Topological Bandwidth of a Tree', SIAM J. Alg. Discr. Meth., Vol. 6, pp 268-277, 1985.

[CMST] M.-J. Chung, F. Makedon, I.H. Sudborough, J. Turner, 'Polynomial Time Algorithms for the Min Cut Problem on Degree Restricted Trees', *SIAM J. Comput.*, Vol. 14, pp 158-177, 1985.

[DT] D. Dolev, H. Trickey, 'Embedding a Tree on a Line', IBM Techn. Report, RJ3368, IBM San Jose, 1982.

[F] A. Feller, 'Automatic Layout of Low-cost Quick Turnaround Random-logic Custom LSI Devices', *Proc. of 13-th Design Automation Conf.*, pp 79-85, 1976.

[GGJK] M.R. Garey, R.L. Graham, D.S. Johnson, D.E. Knuth, 'Complexity Results for Bandwidth Minimization',*SIAM J. on Appl. Math.*, Vol. 34, pp 477-495, 1978.

[GJ] M.R. Garey, R.L. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, W.H. Freeman, 1979.

[H1] F. Harary, Graph Theory, Addison-Wesley, 1969.

[H2] D.S. Hirschberg, 'A Linear Space Algorithm for Computing Maximal Commom Subsequences', *CACM*, Vol. 18, pp 341-344, 1975.

[L] T. Lengauer, 'Upper and Lower Bounds on the Complexity of the Min-Cut Linear Arrangement Problem on Trees', *SIAM J. Alg. Discr. Meth.*, Vol. 3, pp 99- 113, 1982.

[MS] B. Monien, I.H. Sudborough, 'Min Cut is NP-Complete for Edge Weighted Trees, *Proceedings of 13-th ICALP Conf.*, 1986.

[OMKF] T. Ohtsuki, H. Mori, E.S. Kuh, T. Kashiwabara, T. Fujisawa, 'One-dimensional Logic Gate Assignments and Interval Graphs', *IEEE Trans. on Circuits and Systems*, pp 675-684, 1979.

[PDS] G. Persky, D. Deutsch, D. Schweikert, 'A Minicomputer-based System for Automated LSI Layout', *J. Des. Aut. and Fault Tol. Comp.*, pp 217-255, 1977.

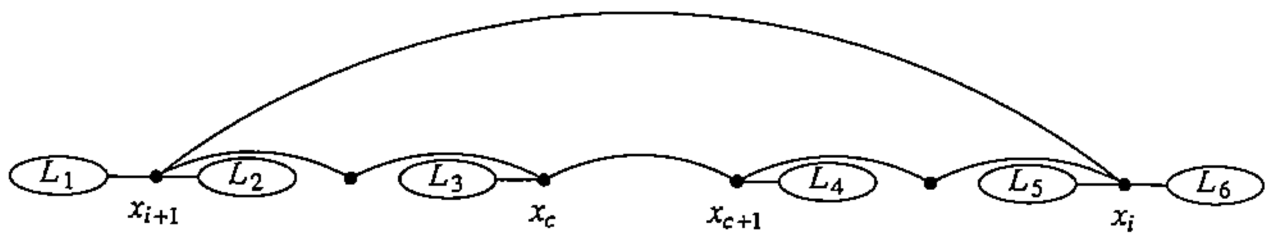
[S1] Y. Shiloach, 'A Minimum Linear Arrangement Algorithms for Undirected Trees', *SIAM J. on Computing*, Nr. 8, pp 15-32, 1979.

[S2] I.H. Sudborough, personal communication, 1986.

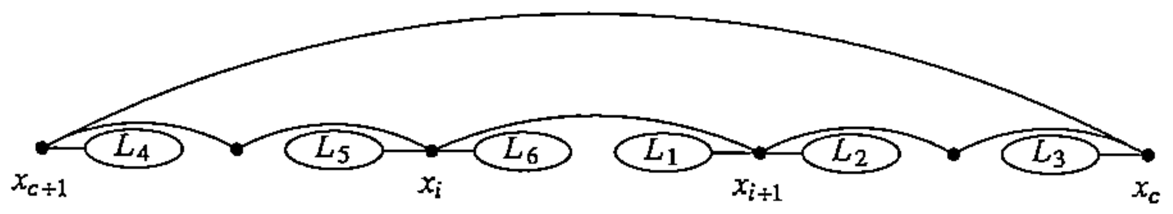
[T] S. Trimberg, 'Automating Chip Layout', *IEEE Spectrum*, pp 38-45, 1982.

[Y1] M. Yannakakis, 'A Polynomial Algorithm for the Min Cut Linear Arrangement on Trees', *JACM*, Vol. 32, pp 950-988, 1985.

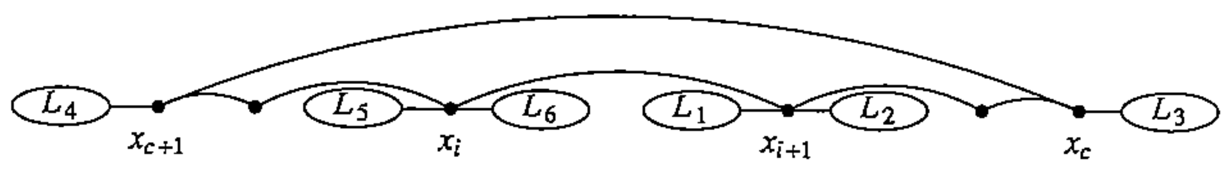
[Y2] M. Yannakakis, 'Four Pages are necessary and Sufficient for Planar Graphs', *Proceedings of 18-th STOC*, pp 104-108, 1986.



(a) layout L

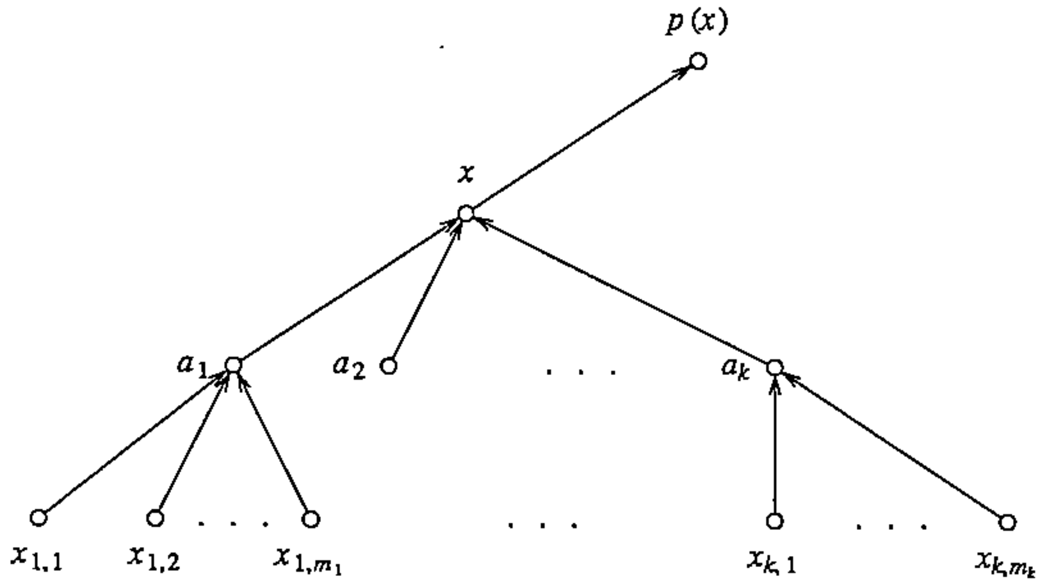


(b) layout $\hat{L}$



(c) layout L'

Edge (x_{c+1}, x_c) violates bandwidth $b = 3n/4$ in $\hat{L}$.
Layouts L_1, L_2, L_5, and L_6 consist of $n/8 - 1$ vertices, L_3 and L_4 of $n/4 - 1$ vertices each.
Figure 2.1



Vertex x, its parent $p(x)$, its children a_i and grandchildren $x_{i,j}$ in tree H

Figure 4.1