DECREASING CHANNEL WIDTH LOWER BOUNDS BY CHANNEL LENGTHENING

Fook-Luen Heng
William W. Lin
Andrea S. LaPaugh
Ron Y. Pinter

CS-TR-218-89

May 1989

Decreasing Channel Width Lower Bounds by Channel Lengthening

Fook-Luen Heng†
*William W. Lin**
Andrea S. LaPaugh

Department of Computer Science
Princeton University
Princeton, New Jersey 08544

Ron Y. Pinter

IBM Scientific Center
Technion City
Haifa 32000, Israel

ABSTRACT

Under the 2-layer Manhattan model for channel routing, the problem of minimizing the channel *width*, i.e., finding the smallest possible number of tracks required, is NP-complete [Szy]. So, when a quick estimate of the width is needed (during placement, for example), one often uses a lower bound metric. *Density* is a commonly used metric, but another interesting metric is *flux* [BBL]. When allowed to move some of the terminals in order to minimize the width, one usually tries to minimize a lower bound metric, hopefully thereby lowering the width.

We consider the problem of decreasing a given metric to a target value by adding empty columns. The empty columns add spaces in the corresponding positions on the top and bottom rows, so there are no changes in either the relative terminal orderings or the vertical alignments. This addition of columns does not affect the channel's density, but it can decrease the flux. Unfortunately, flux is not monotonically non-increasing as columns are added, so we derive from flux a new measure, *smooth-flux*. We consider the problem of adding a minimum number of columns to achieve a given target value for smooth-flux. We show that this problem is equivalent to the Weighted Clique Cover problem on interval graphs, and we present a polynomial-time algorithm for the problem.

---

† Dr. Heng's present address is IBM's Thomas J. Watson Research Center in Yorktown Heights, NY.

* Dr. Lin's present address is the David Sarnoff Research Center in Princeton, NJ.

Section 1: Introduction

Channel routing is a method for the detailed routing of VLSI circuits [Bur]. The input to a channel routing problem consists of two rows, called the *top* and the *bottom*, whose terminals must be connected across an intervening channel using a wiring model. Optionally, there may be specified sets of nets that must enter the channel from the *left* or leave the channel from the *right*. The objective is to connect all terminals that belong to a common net (for each net), using as few horizontal *tracks* as possible overall. The smallest number of tracks that can possibly be used is called the channel's *width*.

How well one can route depends on the routing model one uses. A model specifies the number of layers one may use to route signals, and it specifies in what ways (if any) wire segments may cross or overlap. Among the various options are knock-knee routing, which allows wires on different layers to share a corner, e.g., in [RBM], [PrLi], and [MPS]; unit-vertical-overlap routing, which allows wires on different layers to overlap in the vertical direction for a single unit segment, e.g., in [GaHa]; and unrestricted routing, in which wires may overlap arbitrarily, e.g., in [Ham] and [BrBr].

In this paper, we shall consider only the 2-layer Manhattan model, which is commonly used. This model allows only 2 layers for routing, with one layer reserved for vertical wire segments and the other layer reserved for horizontal wire segments; and all segments must be wholly vertical or horizontal. Wire segments that intersect may *cross* (not affecting one another), or they may be connected electrically via contact cuts.

In general, channel routing under the 2-layer Manhattan model is known to be NP-complete [Szy]. Interestingly, some heuristics route most instances using a number of tracks very close to the *density* (the maximum, over all horizontal positions along the channel, of the number of nets crossing that position) of the instance [RiFi] [YoKu]. It has been noted, however, that some instances require many more tracks to route than would be indicated by the density [BrRi]. This has led to a new lower bound called *flux*, which was introduced by Baker, Bhatt, and Leighton [BBL]. They showed that the channel's width is bounded from above by a function that is linear in both density and flux.

In classical channel routing problems, the terminals on the top and bottom rows are considered to be at fixed positions. It is interesting to consider the problems that arise when we allow the terminals to move. One of the principal difficulties in tackling these problems is that it is hard to judge the "goodness" of a terminal configuration, since it is an NP-complete problem to determine a configuration's width. One solution is, instead of trying to minimize the channel's width directly, to minimize some lower bound metric for the width, hopefully lowering the width simultaneously. For instance, these papers use density as their channel width metric: Gopal, Coppersmith, and Wong [GCW] showed how to achieve the density bound if the relative ordering of terminals on each of the top and bottom are fixed, but the combined ordering can be arbitrarily changed; Atallah and Hambrusch [AtHa] looked at the problem of minimizing the density when the top terminals are fixed, but the bottom terminals are free to be placed in any position among a set of fixed positions; Kobayashi and Drozd [KoDr] studied the problem of minimizing the density when terminals within the same cell may be interchanged, but no mixing of cells is allowed; Johnson, LaPaugh and Pinter [JLP, LaPi] examined the problem of minimizing the density in the lateral placement of components located along the top and bottom of a channel.

There are some circumstances where density may not be the right metric to use. If the vertical alignment of terminals in a channel is fixed — i.e., the ordering of the terminals within the rows is fixed and we may not place an empty space in the top row without placing an empty space in the corresponding

position on the bottom row (or vice versa) — then we cannot change the density, which can only be changed by altering the order or alignment of terminals. If we want to lower the width in this case, our only allowed action is to add an empty column (place empty spaces in corresponding positions on both the top and the bottom), thereby reducing some metric other than density. Flux is an interesting metric to use because it captures the idea that routing space may be reduced by maneuvering signals in empty columns; adding empty columns can decrease flux. In this paper, we consider the problem of adding the minimum number of columns to a channel to decrease flux by a desired amount.

In Section 2, we distill the essential properties of flux to describe a general class of channel width metrics; then we give an example of a well-behaved metric, *smooth-flux*, that belongs to this class. *Smooth-flux* is derived from flux but has desirable properties not held by flux. In Section 3, we present the problem of decreasing a channel width metric to some target value, using as few extra columns as possible. We show that, for our class of metrics, this problem is equivalent to the weighted clique cover problem on interval graphs. In Section 4, we present an algorithm that solves the problem. In Section 5, we consider the computational complexity of the lower bound metric *smooth-flux*. We conclude with some extensions of our problem and some suggestions for future research.

Section 2: Smooth-Flux

We consider a two-dimensional grid model for placement and routing. All terminals lie at intersection points in the grid, and all wire segments run along the grid lines. Initially, the grid lines are marked with integers, as in a Cartesian coordinate system. Each vertical grid line corresponds to a column in the layout, so adding a column in the layout is equivalent to moving terminals and either shifting or stretching wire segments. In our grid model, we add columns by inserting new grid lines between those already present. A new grid line added between the lines numbered i and $i+1$ is given a real number coordinate x such that $i < x < i+1$.

The channel width metric *flux* is affected by the horizontal extent of nets and the vertical alignment of terminals. To define flux, Baker, Bhatt, and Leighton [BBL] consider *horizontal cuts* within the channel, where a horizontal cut isolates from all other terminals those terminals placed on a given row between two given points. Horizontal cuts may either be on the top or bottom row. Flux bounds the number of tracks needed to connect (to each other and to terminals outside the cut) the terminals in a horizontal cut. This is in contrast to a vertical cut, used in calculating density, which isolates all the terminals to the left of some point from those to the right of that point.

We formally define flux as follows. A *trivial net* is one comprising exactly two terminals, both of which lie in the same column — a trivial column. The flux f is the largest integer for which some horizontal cut spanning $2f^2$ nontrivial columns splits at least $2f^2 - f$ nontrivial nets. A net is *unsplit* by a cut if all of this net's terminals lie either inside the cut or outside the cut; otherwise, the net is *split*. (See Figure 1.)

Unfortunately, flux is somewhat anomalous, making it inappropriate as a measure to be minimized. Specifically, flux is is not *monotonic*; that is, we may add an empty column and actually *increase* the value of the flux (see Figure 2). This anomalous behavior occurs due to the coarse granularity between allowed cut sizes in calculating flux — all cuts are of size $2i^2$, for some integer i.

For our purposes, we want a metric that is well-defined and fairly easily computed for any given *window*, where a window is a cut that isolates a set of contiguous horizontal positions from all other positions. Notice that a horizontal cut is a window that contains only positions on the top of the channel or on the bottom of the channel. In general, a window may contain positions from both the top and bottom of

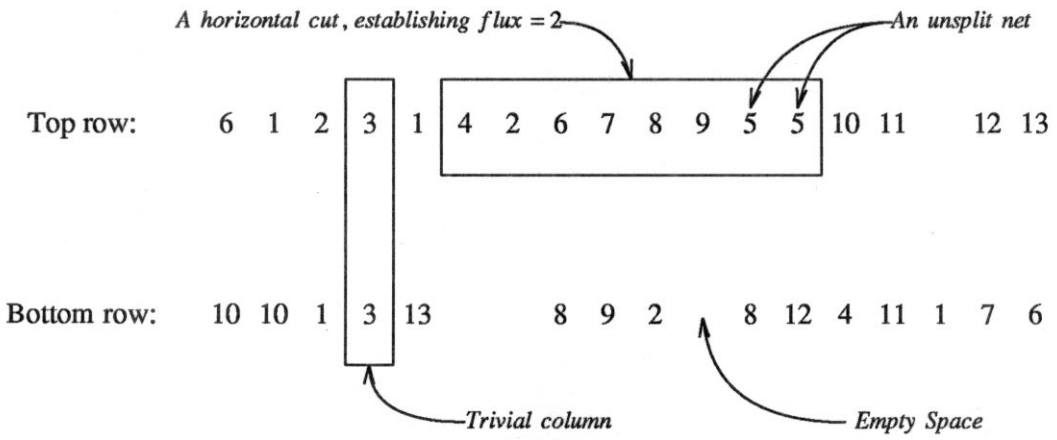


Figure 1. Flux terminology.

the channel. For a window w, define $l(w)$ to be the coordinate of the leftmost terminal or empty space (column) in w and $r(w)$ to be the rightmost such coordinate. When we speak of decreasing the metric for a window w by adding empty columns, we consider the total number of columns spanned by w to be variable; but $l(w)$ and $r(w)$ are fixed. The following conditions must be satisfied by our metric:

(M1) No sub-window of w may have its metric value increased by the addition of empty columns; that is, we require the metric to be monotonically non-increasing as columns are added.

(M2) We must be able efficiently to calculate how many extra columns must be added to a given window in order to lower the metric value for that window to a given target value.

(M3) We must be able to compute the region within the window in which the extra columns must be added in order to achieve the target value. We call this the window's *critical extent.*

The above conditions do not rule out the possibility of a window whose extent is not contiguous, but we do not know of any channel width metric that bounds the width for non-contiguous windows.

There may be a number of metrics that satisfy our above conditions. However, the following metric, derived from flux, is the only one we know of that satisfies the conditions, can be decreased by adding empty columns, and provides a useful lower bound for channel width. We define a revised flux metric, *smooth-flux*, that may be calculated for any horizontal cut.

Define:

C = *the horizontal cut (window) under consideration.*

n_c = *the number of nontrivial columns in* C.

e = *number of empty spaces (positions not containing a terminal) within* C.

S = *number of split nets within* C *(does not include trivial nets).*

U = *number of unsplit (one-sided) nets in* C.

R = *number of redundant terminals in* $C = n_c - e - S - U$

Note that R counts the repeated terminals for the nets in S and U.

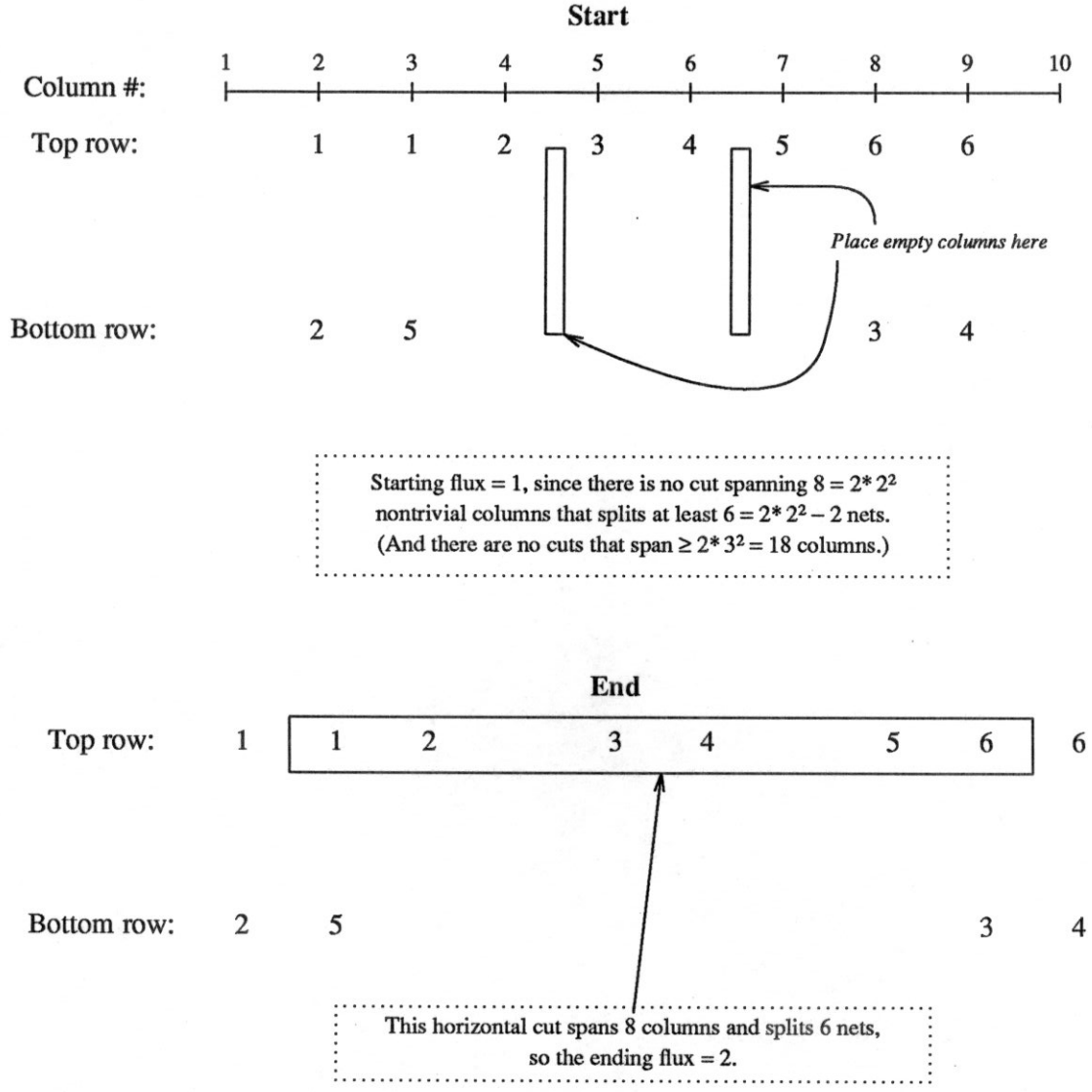


Figure 2. An example where adding empty columns increases the *flux*.

Definition:

Smooth-flux of the cut (window) C, $smooth\text{-}flux(C)$, equals the smallest possible integer f that satisfies the following equation:

$$fe + f(f+1) + (f-1)(U+R) \geq S \tag{1}$$

The smooth-flux of a channel is the maximum of all the smooth-flux values of the windows within the channel. Note that equation (1) is basically identical to the equation defining $flux$, except for the inclusion of the term $(f-1)(U+R)$. Solving the quadratic inequality above, we find the smooth-flux to be

$$f = \left\lceil \frac{-(e+U+R+1) + \sqrt{(e+U+R+1)^2 + 4(U+R+S)}}{2} \right\rceil \tag{2}$$

Or, since $S+e+U+R = n_c$,

$$f = \left\lceil \frac{-(n_c-S+1) + \sqrt{(n_c-S+1)^2 + 4(n_c-e)}}{2} \right\rceil$$

Lemma 1:

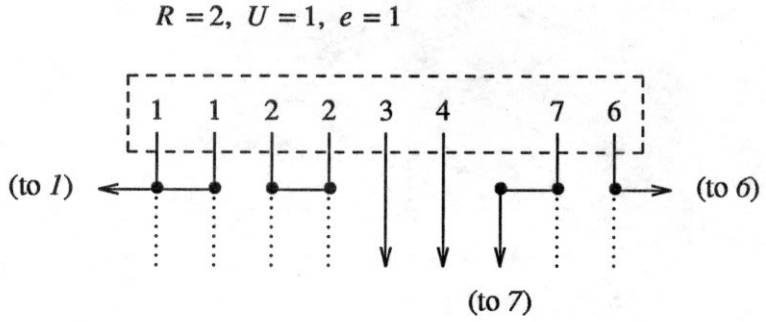
For any cut C, $smooth\text{-}flux(C)$ bounds from below the number of tracks needed to route all the split nets in C.

Proof:

The following analysis holds for either a horizontal cut on the top or a horizontal cut on the bottom. The argument, similar to that in [BrRi] and [BBL], bounds the number of split nets that may be routed into the correct column by using a particular track.

track 1:

Two split nets may connect via track 1 to points outside the cut, adding 2 more columns available for routing (see nets *1* and *6* in Figure 3). Also, e split nets may be routed into the e columns under the free spaces in C (see net *7* in Figure 3). Finally, we might connect all unsplit nets and all redundant terminals. This would add $U+R$ more columns available for routing split nets on all remaining tracks (see nets *1* and *2* in Figure 3).



6 free columns after track 1 routing

Figure 3. Utilization of track 1.

track 2:

For each available column, we may possibly connect together the terminals of a split net. There are (from step 1 above) at most $e+2+U+R$ columns available. In addition, 2 more split nets may be connected to points outside the cut, thus freeing up two more columns.

etc.

In general, on track i, we may connect at most $e+2i+U+R$ split nets, except on the first track, on which we may connect at most $e+2$ split nets. Thus, the maximum number of split nets connected on f tracks is:

$$\sum_{i=1}^{f} e + \sum_{i=1}^{f} 2i + \sum_{i=2}^{f} (U+R) =$$

$$fe + f(f+1) + (f-1)(U+R)$$

Since all split nets must be connected, this number must be greater than or equal to the total number of split nets, S, hence smooth-flux constitutes a lower bound.

□

Now, we shall show that the smooth-flux metric satisfies our aforementioned conditions for a metric to be appropriate.

Lemma 2:

Smooth-flux satisfies conditions (M1), (M2), and (M3) above.

Proof:

We may compute the number of extra columns needed to lower the smooth-flux value from f to t by simply using the definition. That is, we may substitute t for f and re-write equation (1) above to find the total number of free columns needed:

$$e' \geq \frac{S - t(t+1) - (t-1)(U+R)}{t}$$

The number of columns we must add is simply $e' - e$. This proves that smooth-flux satisfies condition (M2).

Since the smooth-flux value for a window v does not depend on the positions but rather on the *number* of split nets, free columns, unsplit nets, and redundant terminals, we may place the extra empty columns anywhere strictly between $l(v)$ and $r(v)$. Thus, we have shown that smooth-flux satisfies condition (M3).

As to the question of whether or not adding an empty column to a window v can raise the smooth-flux, the answer is no. To see that this is the case, consider any *new* window v' created by the addition of empty columns. Clearly, v' is equivalent to an original window v, plus some empty space(s). From equation (2), the change in smooth-flux with respect to e is:

$$\frac{\partial f}{\partial e} = \frac{1}{2}\left\{-1 + \frac{(e+U+R+1)}{\sqrt{(e+U+R+1)^2 + 4(U+R+S)}}\right\}$$

Since $(e+U+R+1) \leq \sqrt{(e+U+R+1)^2 + 4(U+R+S)}$, $\frac{\partial f}{\partial e} \leq 0$; so if we increase e, f decreases. Therefore, smooth-flux(v')≤smooth-flux(v).

Therefore, we may ignore the *new* windows if we wish to lower the value for smooth-flux; and smooth-flux satisfies condition (M1).

□

Section 3: The Problem of Reducing a Channel Width Metric

We consider the following problem: Given an instance of a channel with top and bottom terminals, an appropriate channel width metric, and an integer T, how many empty columns must we add to reduce the channel width metric to T, and where do we place the empty columns? We consider only channel width metrics that satisfy conditions (M1)-(M3) and for which all critical extents are intervals. In Section 3.1, we formally define this problem; in Section 3.2 we show that this problem is equivalent to Weighted

Clique Cover.

Section 3.1: Formal Problem Definition

We begin by restating the problem we wish to solve in the following, equivalent terms.

SATISFACTION OF WINDOW DEMANDS [*SWD*]

Input: Problem instance P consisting of

- a set of *windows* $W = \{w_1, w_2, \ldots, w_m\}$.
- Each window $w \in W$ has an *extent* $(l(w), r(w))$, where $l(w)$ and $r(w)$ are positive integers, $l(w) < r(w)$. Of the two endpoints, $l(w)$ is the *start* of the extent and $r(w)$ is the *end* of the extent.
- Each window $w \in W$ has a *demand* $\delta(w)$.

Conditions:

- We are allowed to place columns at positions within the windows' extents. We may place a column at any non-integer position. No two columns may be placed in the same position.
- If at least $\delta(w)$ columns are placed within the extent of w, we say that the demand of w is *satisfied.*

Question:

- Where do we place columns so that the demand of every window in W is satisfied and so that we use as few columns as possible?

For an instance of *SWD*, we will refer to any placement of columns that satisfies all the window demands as a *feasible solution* and to any placement which, in addition, uses the minimum number of columns as an *optimal solution.*

The correspondence between *SWD* and the problem of decreasing a channel width metric by adding columns is readily seen. Let $w_{channel}$ be a window derived from an instance of the problem of decreasing the channel metric to a certain value by adding empty columns; $w_{channel}$ is a horizontal cut on either the top row or the bottom row. Then, w_{SWD} is the window corresponding to $w_{channel}$ in the instance of the *SWD* problem:

$$window\ w_{SWD} \leftrightarrow window\ w_{channel}$$

$$extent(w_{SWD}) \leftrightarrow critical\ extent\ of\ w_{channel}$$

$$demand\ \delta(w_{SWD}) \leftrightarrow number\ of\ columns\ needed\ by\ w_{channel}$$

We note that the size of an instance may expand in going from the channel context to the *SWD* context, since the windows are not explicitly given in the former problem. In Section 5, we show that for smooth-flux this expansion is at most quadratic.

We now present some definitions and properties of solutions to *SWD* that will be useful in our study of the problem.

Definitions:

- *atomic interval I*:

  for integers j and k, with $j < k$, an interval (j,k) such that there are endpoints of windows at both j and k, but there are no window endpoints at any position between j and k. We denote $l(I) = j, r(I) = k$.

- *critical interval*:

  an atomic interval I such that $l(I) = l(u)$ for some $u \in W$ and $r(I) = r(v)$ for some $v \in W$. Windows u and v may possibly be identical.

- $\rho(I)$:

  where I is an atomic interval. This is the set of windows whose extents contain the extent of interval I, i.e., $w \in \rho(I)$ if $l(w) \leq l(I)$ and $r(w) \geq r(I)$.

Lemma 3 presents three properties of feasible solutions that we use implicitly in our proofs.

Lemma 3:

For any instance of the *Satisfaction of Window Demands* problem, the following three properties hold:

(1) A solution exists if all window demands are finite.

(2) We only need to specify in which atomic interval to place a column; the actual numerical position is unimportant.

(3) Given 3 consecutive atomic intervals I_1, I_2, and I_3, if $w \in \rho(I_1)$ and $w \in \rho(I_3)$, then $w \in \rho(I_2)$.

Proof:

Property (1) is added for completeness, to show that a solution to our problem exists. We may simply place exactly $\delta(w)$ columns in the extent of all windows $w \in W$. This may be done since there are an infinite number of positions within any window.

Property (2) comes from the fact that we do not restrict where we may place a column. Placing a column at a particular position will only affect one atomic interval; and for a given atomic interval, any two columns placed within the interval will affect exactly the same set of windows, namely $\rho(I)$. So, from now on, we shall only specify the atomic interval in which we are placing a column.

Property (3) ensures that a window is contiguous and thus may not "skip over" the extent of an atomic interval. By hypothesis, $l(I_1) \leq l(I_2)$ and $r(I_3) \geq r(I_2)$. Since by definition of ρ, $l(w) \leq l(I_1) \leq l(I_2)$ and $r(w) \geq r(I_3) \geq r(I_2)$, $w \in \rho(I_2)$.

□

Section 3.2 Equivalence of Weighted Clique Cover

We now present a graph covering problem which also models our problem of reducing a channel metric. The problem is *Weighted Clique Cover*. We define the problem for a general graph, but we will only be interested in interval graphs for this application.

WEIGHTED CLIQUE COVER (WCC)

Input: Problem instance P consisting of

- a graph $G=(V,E)$.
- Each node v has a weight $wt(v)$ which is a positive integer.

Question:

- Assign non-negative weights to the cliques of G so that for a given node $v \in V$, the sum of the weights on all cliques containing v is greater than or equal to $wt(v)$. Minimize the sum of the clique weights over all cliques.

An interval graph is defined by a set of intervals on the line. Each node represents an interval and there is an edge between two nodes if the corresponding intervals intersect. Thus for any interval graph there is a corresponding set of windows and vice versa. (Note that the mapping from sets of intervals to interval graphs is many-to-one. Furthermore, it suffices to consider only open intervals [Gol].) Let W be a set of windows and $G=(W,E)$ be the corresponding interval graph, where we use W to denote both the set of windows (i.e. intervals) and the set of nodes. Then for any $w \in W$, $\delta(w)=wt(w)$. Note that each position on the line defines a clique of G consisting of exactly those windows which contain the position. We say that the position *induces* the clique. Similarly, any atomic interval I induces a clique in G, with node set $\rho(I)$. We first establish the correspondence between the critical intervals of a set of windows and the maximal cliques of the corresponding interval graph.

Lemma 4:

For any set of windows W defining interval graph $G=(W,E)$ the set of critical intervals is in one-to-one correspondence with the set of maximal cliques.

Proof:

We shall use "window" to mean both the interval of the line and the node corresponding to that interval.

Given a maximal clique K, let l_K be the rightmost start of a window in K and r_K be the leftmost end of a window in K. We know $l_K \leq r_K$ and all the windows in K contain (l_K, r_K). We claim (l_K, r_K) is atomic, and thus a critical interval. Suppose another window w starts or ends within (l_K, r_K). Then w intersects all the windows in K and thus $K \cup \{w\}$ is a clique of G, contradicting that K is maximal.

Given a critical interval $(l(u), r(v))$, it induces a clique K in G. We claim this clique is maximal. Suppose not. Then there is a window w which intersects all the windows in K but is not in $\rho((l(u), r(v)))$. But then $r(v) \leq l(w)$ and (v,w) is not in E, or $r(w) \leq l(u)$ and (u,w) is not in E. In either case $K \cup \{w\}$ is not a clique, a contradiction.

□

Lemma 5:

Any solution to an instance of Weighted Clique Cover on an interval graph can be transformed to a solution that assigns non-zero weights only to maximal cliques.

Proof:

Given a solution to WCC, transform the solution by transferring any non-zero weight of a clique to the weight of a maximal clique that contains it. Then only maximal cliques have non-

zero weight, the total weight over all cliques is unchanged, and for any node w, the sum of weights over all cliques containing w is at least as large as before.

□

Theorem 1:

Weighted Clique Cover on interval graphs is equivalent to Satisfaction of Window Demands.

Proof:

Let W be a set of windows and $G = (W,E)$ be the corresponding interval graph. We may start with either W or G and derive the other, but if we start with G, W is not unique. We must show that there is a solution to the instance of SWD defined by W and δ which places c columns if and only if there is a solution to the instance of WCC defined by G and wt with total clique weight c.

First consider a solution to SWD. Assign to each clique a weight equal to the number of columns added at positions within the atomic interval that induces the clique. The sum of the weights over all cliques is the number of added columns. Also, the sum of weights over all cliques containing a node w equals the number of columns added to the window w.

We now consider a solution to WCC. Unfortunately, not all cliques of G are induced by a position on the line. Using Lemma 5, transform the solution to a solution in which only maximal cliques have non-zero weight. From the new solution to WCC construct a solution to SWD as follows: for each maximal clique K of weight $wt(K)$, add $wt(K)$ columns to the critical interval corresponding to K. By Lemma 4, this critical interval is unique for K, and each critical interval corresponds to some K. Then the total number of columns added is the total clique weight and the number of columns added to any window is the sum of the weights of all cliques containing that window.

□

Corollary to Theorem 1:

For any instance of the *Satisfaction of Window Demands* problem, there is an optimal solution for which columns are placed only in critical intervals.

We know of no previous work on the general Weighted Clique Cover problem; however, the case where every node's weight equals 1 is simply the problem of *Partition into Cliques*, or finding a minimal-sized set of cliques that covers the set of nodes. (Partition into Cliques requires a set of *disjoint* cliques, but any set of cliques which cover a graph can be transformed into a disjoint set by removing redundant nodes.) The problem Partition into Cliques is known to NP-complete for general graphs [GaJo]. It is solvable in polynomial time for several classes of graphs [GaJo] including chordal graphs, which is a superclass of interval graphs. Our contribution is a solution of the Weighted Clique Cover problem with arbitrary node weights for interval graphs.

Section 4: An Algorithm for Adding Columns

In this section, we present an algorithm which solves Satisfaction of Window Demands. In Section 4.1 we describe the algorithm; in Section 4.2 we give an example of its use. Section 4.3 contains the proof that the algorithm finds an optimal solution.

Section 4.1: The Algorithm

Our algorithm to solve *SWD* is based on the observation (Corollary to Theorem 1) that all extra columns can be placed in the critical intervals. The algorithm starts with the leftmost critical interval and proceeds with each critical interval in turn. As it considers a critical interval, it places enough columns in it so that the window demand is satisfied for any window whose extent includes the present critical interval, but whose extent does not include any critical intervals yet to be considered. Thus, it postpones as long as possible the addition of columns.

Before presenting the algorithm, we present some notation:

- $\eta(x)$:

  where x is either an interval or a window, is the number of columns placed within x's extent, for some assignment of columns.

- *Column Assignment:*

  a possible solution to a problem instance. It tells how many columns are placed in each critical interval. An assignment is given as a sequence of numbers $A = \{\eta(I_1), \eta(I_2), \ldots, \eta(I_M)\}$, where $I_1, I_2, \ldots, I_M$ are all the critical intervals, from left to right.

- *(Total) Cost:*

  The (total) cost of a column assignment A is $c(A)$, the total number of columns used by A's constituents, namely $c(A) = \sum_{k=1}^{M} \eta(I_k)$.

One-pass Algorithm:

Let $W = \{w_1, \ldots, w_m\}$ be the set of windows, where $w_i = (l(w_i), r(w_i))$;
$\{\delta(w_i) \mid 1 \le i \le m\}$ is the set of window demands.

Calculate

$I = \{I_1, \ldots, I_M\}$ set of critical intervals, where $I_i = (l_i, r_i)$ and $l_i < l_{i+1}$.

Initialization:

$W_0 = W$

$U_0 = \varnothing$

$\eta_1(I_0) = 0$, where I_0 is a dummy variable

$\delta_0(w) = \delta(w)$, for *all* $w \in W$

for $i = 1$ *to* M **do**

$V_i = \{ v \in W \mid v \in \rho(I_i)$ and $v \notin \rho(I_j)$ for $j > i \}$

$U_i = \rho(I_i) - V_i$

Initialize $\eta_1(I_i) = 0$

end

for $i = 1$ *to* M **and** while W_{i-1} is nonempty **do**

(1) Update window demands,

$$\delta_i(w) = \begin{cases} \delta_{i-1}(w) & \text{, if } w \notin U_{i-1} \\ \max\{0, \delta_{i-1}(w) - \eta_1(I_{i-1})\} & \text{, if } w \in U_{i-1} \end{cases}$$

(2) Compute $\eta_1(I_i)$, the number of columns to be placed in I_i,

$$\eta_1(I_i) = \max_{v \in V_i} \{\delta_i(v)\}$$

(3) Update the current set of windows,

$$W_i = W_{i-1} - V_i$$

end

We shall denote the column assignment produced by the One-pass Algorithm as $A_1 = \{ \eta_1(I_1), \ldots, \eta_1(I_M) \}$.

Theorem 2:

The One-pass Algorithm finds an optimal solution to the Satisfaction of Window Demands Problem in running time $O(m^2)$, where m is the number of windows.

Proof:

We postpone the proof that the One-pass Algorithm correctly finds an optimal solution. It is the proof of Lemma 8 in Section 4.3. Here we analyze the running time.

(A) The initialization:

Sort the endpoints of windows in W and scan the sorted endpoints from left to right in order to compute I. This takes $O(m \log m)$ time.

Let P denote the total size of all $\rho(I_i)$'s, i.e., $P = \sum_{i=1}^{M} |\rho(I_i)|$.

It takes O(P) time to compute V_i, U_i, and $\rho(I_i)$, for $i = 1$ *to* M.

(B) The main loop:

Steps (1), (2), and (3) together take O(P) time.

Thus, the complexity of the One-pass Algorithm is $O(P + m \log m)$ time. Clearly, P is bounded above by $M * m$, where M is the number of critical intervals. This gives a bound on the running time of

$$O(Mm + m \log m) \tag{3}$$

Since the number of critical intervals is not greater than the number of windows, $M \leq m$; so $P = O(m^2)$. Therefore, the running time of the algorithm is $O(m^2)$, i.e., quadratic in the number of windows.

□

Section 4.2: Example Usage of the One-pass Algorithm

Below is an example of the usage of the One-pass Algorithm (see Figure 4). We show the input and the running of the algorithm for an example where the number of critical intervals is three.

Input:

$W = \{w_1, w_2, w_3, w_4\}$

Window	Extent		Demand
	Start	End	
w_1	1	3	3
w_2	2	4	6
w_3	3	7	8
w_4	5	8	4

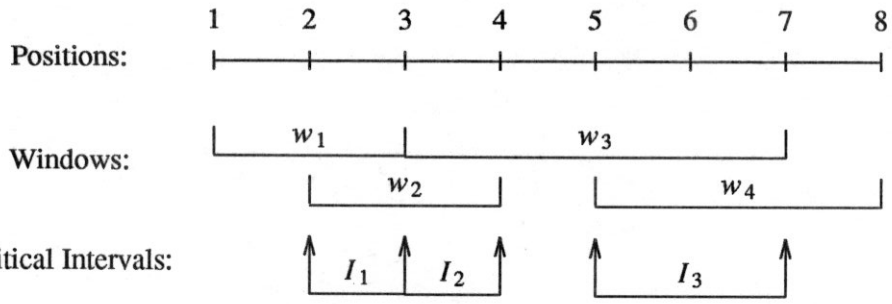


Figure 4. Sample input to the One-pass Algorithm.

The critical intervals are $I_1 = (2,3)$, $I_2 = (3,4)$, and $I_3 = (5,7)$.

Iteration 1: $(i = 1)$

$W_0 = \{w_1, w_2, w_3, w_4\}$

Demands are $\delta_1(w_1) = 3$, $\delta_1(w_2) = 6$, $\delta_1(w_3) = 8$, $\delta_1(w_4) = 4$

We find that $V_1 = \{w_1\}$, $U_1 = \{w_2\}$ and so $\eta_1(I_1) = 3$. We place 3 columns within I_1, then we continue with the algorithm.

Iteration 2: $(i = 2)$

$W_1 = \{w_2, w_3, w_4\}$
Demands are $\delta_2(w_2) = 3$, $\delta_2(w_3) = 8$, $\delta_2(w_4) = 4$

We find that $V_2 = \{w_2\}$, $U_2 = \{w_3\}$ and so $\eta_1(I_2) = 3$. We place 3 columns within I_2, then we continue with the algorithm.

Iteration 3: $(i = 3)$

$W_2 = \{w_3, w_4\}$
Demands are $\delta_3(w_3) = 5$, $\delta_3(w_4) = 4$

We find that $V_3 = \{w_3, w_4\}$, $U_3 = \varnothing$ and so $\eta_1(I_3) = 5$. We place 5 columns within I_3, then we are finished since $i = M$.

Totally, we have used 11 columns (which is optimal), and the assignment $A_1 = \{3, 3, 5\}$.

□

Section 4.3: Proof of Correctness of the One-pass Algorithm

We want to prove that our One-pass Algorithm finds an optimal solution to *SWD*. We begin by proving a lemma that allows us to take two feasible solutions — one with lower cost and one which postpones the addition of columns longer — and produce a feasible solution which has both the lower cost and the postponed addition of columns. We shall use the lemma later to derive the solution produced by the One-pass Algorithm from an arbitrary optimal solution.

Lemma 6:

Let

$$H = \{ \eta(I_1),...,\eta(I_M) \}$$
$$H' = \{ \eta'(I_1),...,\eta'(I_M) \}$$

be two feasible solutions to an instance of the *Satisfaction of Window Demands* problem such that there is a $t < M$ with $\eta(I_j) = \eta'(I_j)$ for $1 \le j < t$ and $\eta(I_t) < \eta'(I_t)$ (i.e., H and H' agree for the first $t-1$ intervals, and H' has more columns at the t^{th} interval). Then we can find another solution,

$$H'' = \{ \eta''(I_1),...,\eta''(I_M) \}$$

such that

$$\eta''(I_j) = \eta'(I_j) = \eta(I_j), \text{ for } j < t,$$
$$\eta''(I_t) = \eta(I_t),$$
$$cost(H'') = cost(H').$$

Proof:

Intuitively, we may take solution H' and *ship* the extra number of columns, $\eta'(I_t) - \eta(I_t)$, in interval I_t to interval I_{t+1} without changing the cost and validity of the solution. We can show this by constructing H'' as follows:

$\eta''(I_j) = \eta'(I_j)$, for $j < t$

$$\eta''(I_t) = \eta(I_t)$$
$$\eta''(I_{t+1}) = \eta'(I_{t+1}) + \eta'(I_t) - \eta(I_t)$$
$$\eta''(I_k) = \eta'(I_k) \ , \text{for } k > t+1$$

There is no change in cost:

$$cost(H'') = \sum_{i=1}^{M} \eta''(I_i)$$
$$= \sum_{j=1}^{t-1} \eta'(I_j) + \eta(I_t) + \eta'(I_{t+1}) + \eta'(I_t) - \eta(I_t) + \sum_{k=t+2}^{M} \eta'(I_k)$$
$$= \sum_{i=1}^{M} \eta'(I_i) = cost(H')$$

Validity of H'':

We must show that for any window w, $\eta''(w) \geq \delta(w)$. Recall that $\eta(w) \geq \delta(w)$ and $\eta'(w) \geq \delta(w)$, since H and H' are both solutions. Let arbitrary window w contain all critical intervals from I_p to I_q, $p \leq q$, and no other critical intervals. Any window which lies completely to the left of I_t $(q < t)$ must have its demand satisfied since $\eta''(w)=\eta'(w)$; any window which lies completely to the right of I_{t+1} $(p > t+1)$ must also have its demand satisfied since $\eta''(w)=\eta'(w)$. We must check that the demand is met for windows of the following three types:

(1) $q = t$
Window w contains I_t but not I_{t+1}.

(2) $p = t+1$
Window w contains I_{t+1} but not I_t.

(3) $p \leq t < t+1 \leq q$
Window w contains intervals I_t and I_{t+1}.

We now show that $\eta''(w) \geq \delta(w)$ for each type of window.

Type 1 window:

$$\eta''(w) = \eta''(I_p) + \cdots + \eta''(I_t)$$
$$= \eta'(I_p) + \cdots + \eta'(I_{t-1}) + \eta(I_t)$$
$$= \eta(I_p) + \cdots + \eta(I_t) = \eta(w) \geq \delta(w) \ , \text{since } \eta'(I_j) = \eta(I_j) \text{ for } j < t$$

Type 2 window:

$$\eta''(w) = \eta''(I_{t+1}) + \cdots + \eta''(I_q)$$
$$= \eta'(I_{t+1}) + \eta'(I_t) - \eta(I_t) + \eta'(I_{t+2}) + \cdots + \eta'(I_q)$$
$$= \eta'(w) + (positive\ value) > \eta'(w) \geq \delta(w)$$

Type 3 window:

$$\begin{aligned}\eta''(w) &= \eta''(I_p) + \cdots + \eta''(I_t) + \eta''(I_{t+1}) + \cdots + \eta''(I_q) \\ &= \eta'(I_p) + \cdots + \eta(I_t) + \eta'(I_{t+1}) + \eta'(I_t) - \eta(I_t) + \cdots + \eta'(I_q) \\ &= \eta'(w) \geq \delta(w)\end{aligned}$$

□

To complete the proof of Theorem 2, we first show that the column assignment produced by the One-pass Algorithm satisfies all window demands. We then apply Lemma 6 to show that the solution is in fact optimal.

Lemma 7:

For any instance of the *Satisfaction of Window Demands* problem, the assignment $A_1 = \{ \eta_1(I_1), \ldots, \eta_1(I_M) \}$ constructed by the One-pass Algorithm is a feasible solution.

Proof:

Consider a window w whose extent includes critical intervals I_s through I_t, i.e. $r_{s-1} \leq l(w) \leq l_s < r_t \leq r(w) \leq l_{t+1}$. (See Figure 5)

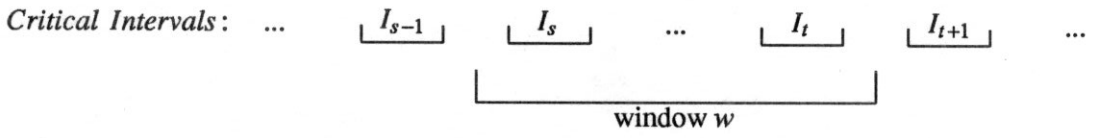


Figure 5. Window extending from I_s to I_t.

Observe the following:

(i) From the initialization, $w \in V_t$, since $w \in \rho(I_t)$ but $w \notin \rho(I_{t+1})$.

(ii) The number of columns placed in w is

$$\eta_1(w) = \eta_1(I_s) + \cdots + \eta_1(I_t) = \sum_{k=s}^{t} \eta_1(I_k).$$

(iii) From step (1) of the One-pass Algorithm,

$$\delta_s(w) = \delta_{s-1}(w),$$

since $w \notin \rho(I_{s-1}) \supseteq U_{s-1}$.

Similarly,

$$\delta_k(w) = \delta_{k-1}(w), \text{ for } 1 \leq k < s.$$

Therefore,

$$\delta_s(w) = \delta_{s-1}(w) = \cdots = \delta_0(w) = \delta(w). \qquad (4)$$

From (i) above, $w \notin V_k$, for $k \neq t$. So from step (1) of the One-pass Algorithm, the window demand for w from the $s+1^{st}$ to the t^{th} iteration is

$$\delta_{k+1}(w) = \max \{ 0, \delta_k(w) - \eta_1(I_k) \},$$

$$since \;\; w \in \rho(I_k) - V_k = U_k \;, \text{ for } s \leq k < t$$

i.e.,

$$\delta_{k+1}(w) \geq \delta_k(w) - \eta_1(I_k), \text{ for } s \leq k < t. \tag{5}$$

Combining equations (4) and (5) gives

$$\delta_t(w) + \eta_1(I_{t-1}) + \cdots + \eta_1(I_s) \geq \delta_s(w) = \delta(w). \tag{6}$$

But from step (2) in the One-pass Algorithm,

$$\eta_1(I_t) = \max_{u \in V_t} \{\delta_t(u)\} \geq \delta_t(w), \tag{7}$$

Therefore, (ii), (6), and (7) give

$$\eta_1(w) = \sum_{k=s}^{t} \eta_1(I_k) \geq \delta(w) \;\; \text{, for } all \; w \in W.$$

Hence, all window demands are satisfied.

□

Lemma 8:

For any instance of the *Satisfaction of Window Demands* problem, the column assignment $A_1 = \{ \eta_1(I_1),...,\eta_1(I_M) \}$ constructed by the One-pass Algorithm is an optimal solution.

Proof:

By Lemma 7, the assignment A_1 is a feasible solution. Assume, for the sake of contradiction, that A_1 is not optimal. Let $H = \{ \eta(I_1),...,\eta(I_M) \}$ be the optimal solution that matches A_1 for as long as possible. Then

$$cost(H) = \sum_{i=1}^{i=M} \eta(I_i) < \sum_{i=1}^{i=M} \eta_1(I_i) = cost(A_1)$$

and there is a $t \geq 1$ such that

$$\eta(I_j) = \eta_1(I_j) \;\; \text{, for } j < t$$

and

$$\eta(I_t) < \eta_1(I_t) .$$

$\eta(I_t) < \eta_1(I_t)$ follows since if $\eta(I_t) > \eta_1(I_t)$, then for $t < M$ we could apply Lemma 6 to H and A_1 to obtain a feasible solution H' with $cost(H') = cost(H)$ and $\eta'(I_j) = \eta_1(I_j)$ for $j \leq t$, i.e. an optimal solution which matches A_1 longer than H, and, for $t=M$ we would have $cost(A_1) < cost(H)$.

Consider the column assignment for critical interval I_t (step (2) of One-pass Algorithm):

$$\eta_1(I_t) = \max_{w \in V_t} \{\delta_t(w)\}$$

Let $v \in V_t$ be a window that achieved the maximum, i.e. $\eta_1(I_t) = \delta_t(v)$. Let the intervals intersected by v be $I_x, \ldots, I_t$, for some $x \leq t$. Then, by reasoning analogous to that in the proof of Lemma 7,

$$\delta(v) = \delta_x(v) = \sum_{j=x}^{t} \eta_1(I_j) = \eta_1(v)$$

Then,

$$\begin{aligned} \eta(v) &= \sum_{j=x}^{t} \eta(I_j) \\ &= \sum_{j=x}^{t} \eta_1(I_j) + (\eta(I_t) - \eta_1(I_t)) \\ &< \eta_1(v) = \delta(v) \quad , since\ \eta(I_t) - \eta_1(I_t) < 0. \end{aligned}$$

But this means that v's demand is not satisfied by H, which contradicts that H is a feasible solution. Our assumption that A_1 is not optimal has led to a contradiction; thus, we conclude that A_1 is an optimal solution.

□

Section 5: Notes on the *smooth-flux* metric

There are a couple of observations that help reduce the time complexity to lower the smooth-flux to a given target value. One simple point, which applies to any other appropriate metric, is that we only have to consider a window if its metric value is greater than the target value. If the value is less than or equal to the target value, we may just leave that window alone.

An important point in lowering the smooth-flux is that we may disregard any window w that contains an empty space in either its leftmost or its rightmost column. To see why this is so, consider the largest subwindow w' of w such that:

- w' contains all the terminals contained in w
- w' contains terminals in both its leftmost and rightmost columns (see Figure 6).

From w to w', the values for U, R, and S remain unchanged, but the value for e has decreased. As we did in arguing that condition (M1) holds (Section 2), we conclude that if we decrease e, f increases. In other words, the smooth-flux value for w' is greater than or equal to the value for w. And since any empty column we place within w' will also lie within w, we may decrease the smooth-flux of w by decreasing the smooth-flux of w'.

Now, we shall compute the time complexity of calculating the starting value of smooth-flux for a problem instance. Let N be the maximum of the number of top row terminals and the number of bottom row terminals. As a preliminary step we order each row of terminals by position; this takes $O(N \log N)$ time. Since we only need to consider the windows that begin and end with terminals (but we need to consider both top windows and bottom windows), there are at most $2 * \binom{N}{2} = N(N-1)$ relevant windows. We scan through the relevant windows by continuously fixing the left endpoint (at some terminal's position) and varying the right endpoint (at another terminal's position). While doing this, determine the values n_c, S, U, and R [$e = n_c - (S+U+R)$] for each window. The update time for each window is constant, since the smooth-flux for a window with only one terminal is immediate, and all other windows

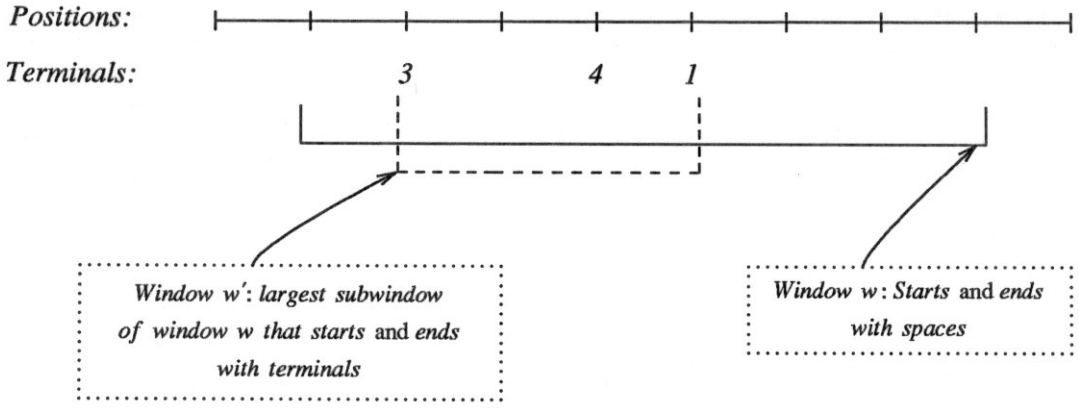


Figure 6. Important subwindow.

only have to increment or decrement each value by at most 1 from the values for the previous window. With these values, we can compute the smooth-flux for each window. Totally, this takes $O(N^2)$ time.

Finally, we note that the number of critical windows is bounded by the number of possible left endpoints for critical windows. This is simply $N - 1$. Thus, using equation (3) from Section 4.1, we have:

Corollary to Theorem 2:

The total time required to reduce the smooth-flux of a channel to a target value T is $O(N^3)$.

Section 6: Extensions and future work

Noting that we know of only one example of an appropriate, non-trivial metric for our algorithm, it would be interesting to find other useful metrics that belong to the same class. Also, there are a number of possible extensions and modifications for our problem of reducing a given cost metric. Here are two very closely related problems:

(1) The extra columns may be constrained to be placed within a set of *allowable* line segments. This situation might arise if the terminals are divided into *components* that are rigid. In such a case, we may expand or contract the amount of space only *between* components.

(2) We may be given a fixed number of extra columns, with the task of placing these columns in order to reduce the cost metric as much as possible.

Problem (1) above can be solved by our One-pass Algorithm, with the modification that instead of considering only the critical intervals, we consider placing columns only in the maximal intervals among atomic intervals intersecting some *allowable* segment, where atomic intervals are partially ordered by the inclusion order of their induced cliques.

Problem (2) can be solved by applying our One-pass Algorithm and using a binary search on the target value for the cost metric. Since the value for smooth-flux, like the value for flux, is bounded by $O(\sqrt{n})$, where n is the total number of nets, we only need to consider at most $O(\log n)$ target values. The

total running time of this algorithm is $O(N^3 \log n)$. It is not clear that this is the optimal method for solving (2), though. More research might yield a better algorithm. It may also be possible to decrease the running time of the One-pass Algorithm by finding a better data structure for the $\rho(I)$'s.

The One-pass algorithm adds the minimum number of columns when the critical extent of each window is a single interval. If the critical extent of a window is not a single interval, but rather a set of intervals, the technique of postponing the addition of columns as long as possible will still find a feasible solution, but it does not give an optimal solution. This is because Property 3 of Lemma 3 does not hold; a window can skip over intervals. It still suffices to consider only maximal atomic intervals, but now a window may have several left and right boundary points defining atomic intervals. Even if all window demands equal 1, this extension of SWD is NP-complete, by transformation from Minimum (Set) Cover [GaJo]. Since we know of no channel width metric that can yield a critical extent containing several intervals, the problem is principally of theoretical interest.

A very interesting extension of our problem that is of practical significance is the following open problem:

Arbitrary Channel Lengthening

Instead of adding *columns*, we may add arbitrary *spaces* on the top or bottom row, as long as the total number of columns does not exceed a given limit. This situation might represent a channel where the terminal ordering is fixed, but the terminals are otherwise flexible.

This is similar in flavor to the problems examined in [GCW] and [JLP]. This extension adds substantial complexity when *smooth-flux* is considered because trivial nets can be created.

Acknowledgments

We would like to thank Martin Golumbic for pointing out the Weighted Clique Cover formulation and Jin-Yi Cai for help in clarifying the presentation.

The work of Fook-Luen Heng, William Lin and Andrea LaPaugh was supported in part by NSF grant number MIP86119335 and DARPA/ONR contract N00014-88-K-0459.

References

[AtHa] Atallah, M. J. and S. E. Hambrusch, "On Bipartite Matchings of Minimum Density," *Journal of Algorithms*, vol. 8, pp. 480-502, 1987.

[BBL] Baker, B., S. N. Bhatt, and F. T. Leighton, "An Approximation Algorithm for Manhattan Routing," *Advances in Computing Research 2 (VLSI Theory)*, pp. 205-229, ed. F. P. Preparata, JAI Press, 1984.

[BrBr] Brady, M. and D. J. Brown, "Optimal Multilayer Channel Routing with Overlap," *4th MIT Conf. on Advanced Research in VLSI*, pp. 281-298, MIT Press, 1986.

[BrRi] Brown, D. J. and R. L. Rivest, "New Lower Bounds for Channel Routing," *Proc. 1981 CMU Conf. on VLSI*, pp. 178-185, 1981.

[Bur] Burstein, M., "Channel Routing," from *Layout Design and Verification,* T. Ohtsuki, ed., volume 4 of *Advances in CAD for VLSI* series, pp. 133-167, North-Holland, 1986.

[GaHa] Gao, S. and S. Hambrusch, "Two-Layer Channel Routing with Vertical Unit-Length Overlap," *Algorithmica*, vol. 1, no. 2, pp. 223-232, Springer-Verlag, 1986.

[GaJo] Garey, Michael R., and David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W. H. Freeman and Company, San Francisco, 1979.

[Gol] Golumbic, M. C., *Algorithmic Graph Theory and Perfect Graphs,* Academic Press, New York, 1980.

[GCW] Gopal, I. S., D. Coppersmith, and C. K. Wong, "Optimal Wiring of Movable Terminals," *IEEE Trans. on Computers*, vol. C-32, no. 9, pp. 845-858, Sept. 1983.

[Ham] Hambrusch, S., "Channel Routing Algorithms for Overlap Models," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-4(1), pp. 23-30, Jan. 1985.

[JLP] Johnson, D. S., A. S. LaPaugh and R. Y. Pinter, "Minimizing Channel Density by Lateral Shifting," unpublished manuscript, 1988.

[KoDr] Kobayashi, H. and C. E. Drozd, "Efficient Algorithms for Routing Interchangeable Terminals," *IEEE Transactions on Computer-Aided Design* , vol. CAD-4, No. 3, pp. 204-207, IEEE, July 1985.

[LaPi] LaPaugh, Andrea S., and Ron Y. Pinter, "On Minimizing Channel Density by Lateral Shifting," *Proceedings of the International Conf. on Computer-Aided Design,* September, 1983.

[MPS] Mehlhorn, K., F. Preparata, and M. Sarrafzadeh, "Channel Routing in Knock-Knee Mode: Simplified Algorithms and Proofs," *Algorithmica*, vol. 1, no. 2, pp. 213-221, Springer-Verlag, 1986.

[PrLi] Preparata, F. P. and W. Lipski, "Optimal Three-Layer Channel Routing," *IEEE Trans. on Computers*, vol. C-33, pp. 427-437, 1984.

[RBM] Rivest, R. L., A. Baratz, and G. Miller, "Provably Good Channel Routing Algorithms," *1981 CMU Conf. on VLSI Systems and Computations*, pp. 158-159, Oct. 1981.

[RiFi] Rivest, R. L. and C. M. Fiduccia, "A 'Greedy' Channel Router," *Proc. 19th Design Automation Conference*, pp. 418-423, IEEE, 1982.

[Szy] Szymanski, T. G., "Dogleg Channel Routing is NP-Complete," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-4(1), pp. 31-41, Jan. 1985.

[YoKu] Yoshimura, T. and E. Kuh, "Efficient Algorithms for Channel Routing," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD-1(1), pp. 25-35, Jan. 1982.