Finding a Maximum Planar Subset of a Set of Nets in a Channel

KENNETH J. SUPOWIT

***Abstract*—An algorithm is presented that, given N two-pin nets in a channel, finds, in $O(N^2)$ time, the largest subset that can be routed all on one layer.**

I. Introduction

CONSIDER THE ROUTING of two-pin nets in a channel using two layers, one of which is strongly preferred over the other. Such a situation is quite common in many currently used technologies, e.g., traces on one layer may be 250 times as resistive or four times as wide as those on the other layer (e.g., the former case arises when one layer is polysilicon and the other metal, the latter when using two layers of metal in certain technologies). In this case, a good strategy is to river-route (see [1], [2]) the largest possible subset using the preferred layer, and then to route the remaining nets using both layers. The current paper gives an algorithm to find this largest river-routable subset.

The problem is precisely equivalent to finding a maximum cardinality independent set in a circle graph. Consider a finite set C of N chords of a circle. Associate with C an undirected graph defined as follows: each vertex of the graph corresponds to a chord of C, and there is an edge between each pair of vertices whose corresponding chords intersect. Such a graph is called a *circle graph*. Fig. 1 illustrates a set of chords and its asociated circle graph (for now, ignore the numbers on the chords' endpoints). Circle graphs were introduced in [3].

An *independent set* of a graph is a subset of its vertices of which no two are joined by an edge. The problem of finding a maximum independent set for arbitrary graphs is NP-hard [4]. However, for the special case of circle graphs, Gavril [5] gives a $\Theta(N^3)$ time algorithm for finding this maximum independent set. The current paper presents an $O(N^2)$ time algorithm for this problem. Our algorithm is also conceptually simpler and more straightforward to program than that of [5].

Related, but distinct, problems that have received attention include testing whether a given set of nets is planar [6], and finding a *maximal* independent set in a graph (i.e., an independent set of vertices that is not a subset of another independent set) [7].

Manuscript received June 26, 1984; revised July 26, 1986.

The author was with Hewlett-Packard Laboratories, Palo Alto, CA 94304. He is now with the Department of Computer Science, Princeton University, Princeton, NJ 08544.

IEEE Log Number 8611293.

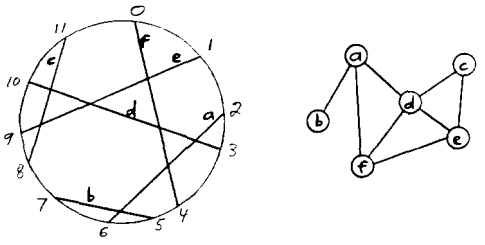

Fig. 1. A set of arcs and its circle graph.

II. The Algorithm

Given is a set C of N chords of a circle. We assume without loss of generality that no two chords of C share an endpoint, since we can slightly move one chord without changing the circle graph, as is observed in [5]. Number the endpoints of these chords from 0 to $2N - 1$, clockwise around the circle (as in Fig. 1). We use ab to denote the chord having points numbered a and b as endpoints.

Let

$$V = \{v_{a,b}: a < b \text{ and } ab \in C\}$$

and let

$$E = \{(v_{a,b}, v_{a',b'}) \in V^2: a < a' < b < b' \text{ or } a' < a < b' < b\}.$$

In other words, each vertex $v_{ab} \in V$ corresponds to a chord $ab \in C$, and there is an edge in E for each pair of vertices whose corresponding chords intersect. Thus, $G = (V, E)$ is the circle graph associated with C. If $0 \le i, j \le 2N - 1$, then let G_{ij} denote the subgraph of G induced by the set of vertices

$$\{v_{ab} \in V: i \le a, b \le j\}.$$

Let $\mathrm{MIS}(i, j)$ denote a maximum independent set of G_{ij}. Note that if $i \ge j$, then G_{ij} is the empty graph and, hence, $\mathrm{MIS}(i, j) = \varnothing$.

The algorithm is an application of dynamic programming. In particular, $\mathrm{MIS}(i, j)$ is computed for each pair i, j; $\mathrm{MIS}(i, j_1)$ is computed before $\mathrm{MIS}(i, j_2)$ if $j_1 < j_2$. To compute $\mathrm{MIS}(i, j)$, we let k be the unique number such that $kj \in C$ or $jk \in C$. If k is not in the range $[i, j - 1]$, then $G_{ij} = G_{i,j-1}$, and, hence, we set $\mathrm{MIS}(i, j) = \mathrm{MIS}(i, j - 1)$. If k is in the range $[i, j - 1]$, then there are two cases to consider: if $v_{kj} \in \mathrm{MIS}(i, j)$, then, by

definition of an independent set, $\mathrm{MIS}(i, j)$ contains no vertices v_{ab} such that $a \in [i, k-1]$ and $b \in [k+1, j]$; hence, we set

$$\mathrm{MIS}(i, j) = \mathrm{MIS}(i, k-1) \cup \{v_{kj}\} \cup \mathrm{MIS}(k+1, j-1).$$

On the other hand, if $v_{kj} \notin \mathrm{MIS}(i, j)$, then we set $\mathrm{MIS}(i, j) = \mathrm{MIS}(i, j-1)$, since G_{ij} differs from $G_{i,j-1}$ only in that G_{ij} has the extra vertex v_{kj} and its associated edges. Thus, we set $\mathrm{MIS}(i, j)$ to be the larger of the two sets $\mathrm{MIS}(i, j-1)$ and $\mathrm{MIS}(i, k-1) \cup \{v_{kj}\} \cup \mathrm{MIS}(k+1, j-1)$.

More formally, the algorithm is as follows:

```
FOR j ← 0 TO 2N − 1 DO
  BEGIN
    [[comment: compute MIS(i, j) for each i < j]]
      k ← the number such that kj ∈ C or jk ∈ C;
      FOR i ← 0 TO j − 1 DO
        IF i ≤ k ≤ j − 1
            and |MIS(i, k − 1)| + 1 + |MIS(k + 1, j − 1)| > |MIS(i, j − 1)|
          THEN MIS(i, j) ← MIS(i, k − 1) ∪ {v_kj} ∪ MIS(k + 1, j − 1)
          ELSE MIS(i, j) ← MIS(i, j − 1)
  END
```

Finally, since $G = G_{0,2N-1}$, we have that $\mathrm{MIS}(0, 2N-1)$ is a maximum independent set of G.

Note that the list of elements of $\mathrm{MIS}(i, j)$ need not be explicitly stored when it is computed in the body of the interior loop; rather $\mathrm{MIS}(i, j)$ may be represented by pointers to $\mathrm{MIS}(i, k-1)$ and to $\mathrm{MIS}(k+1, j-1)$ if the condition is true, and by a pointer to $\mathrm{MIS}(i, j-1)$ if the condition is false. Then the last step of the algorithm is to explicitly enumerate the elements of $\mathrm{MIS}(0, 2N-1)$, which can be done in $O(N)$ time by traversing back from $\mathrm{MIS}(0, 2N-1)$ along these pointers (which form a tree of $O(N)$ nodes each having at most two sons). We must store the cardinality of $\mathrm{MIS}(i, j)$ when it is computed in the body of the interior loop; this is an $O(1)$ operation since we have explicitly stored the sizes of $\mathrm{MIS}(i, k-1)$, of $\mathrm{MIS}(k+1, j-1)$, and of $\mathrm{MIS}(i, j-1)$. Therefore, the body of the interior loop can be implemented in $O(1)$ time, and, hence, the entire algorithm runs in $O(N^2)$ time.

III. Open Problems

An interesting generalization of this problem arises when there are $k > 2$ layers available, with some being more preferable for routing than others. More formally, define the *weighted coloring* problem as follows: given a circle graph $G = (V, E)$ and a cost vector $\boldsymbol{c} = (c_1, c_2, \cdots, c_k)$, partition V into independent sets $V_1, V_2, \cdots, V_k$ so as to minimize $\Sigma_i c_i |V_i|$. Thus, the special case in which $c_1 = 0$, $c_i = 1\ \forall i \geq 2$ is precisely the maximum independent set problem. A useful cost vector for routing might be to have c_i be the width of the traces on routing layer i.

References

[1] D. Dolev, K. Karplus, A. Siegel, A. Strong, and J. D. Ullman, "Optimal wiring between rectangles," in *Proc. 13th ACM Symp. Theory Computing*, May 1981, pp. 312–317.

[2] M. Tompa, "An optimal solution to a wire-routing problem," *J. Comput. Syst. Sci.*, vol. 23, pp. 127–150, Oct. 1981

[3] S. Even and A. Itai, "Queues, stacks, and graphs," in *Theory of Machines and Computations*, A. Kohavi and A. Paz, Eds. New York: Academic, 1971, pp. 71–86.

[4] R. M. Karp, "Reducibility among combinational problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. New York: Plenum Press, 1972, pp. 85–104.

[5] F. Gavril, "Algorithms for a maximum clique and a maximum independent set of a circle graph," *Networks*, vol. 3, pp. 261–273, 1973.

[6] M. Marek-Sadowska and T. T.-K. Tarng, "Single-layer routing for VLSI: Analysis and algorithms," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, no. 4, pp. 246–259, Oct. 1983.

[7] T. Chiba, I. Nishioka, and I. Shirakawa, "An algorithm for maximal planarization of graphs," in *Proc. ISCAS*, 1979, pp. 649–652.

*

Kenneth J. Supowit received the A.B. degree in linguistics from Cornell University, Ithaca, NY, in 1978, and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 1981, in the area of computational geometry.

From 1981 to 1984, he was a member of the Design Automation Department at Hewlett-Packard Laboratories in Palo Alto, CA. He is currently an Assistant Professor of Computer Science at Princeton University, Princeton, NJ. His research interests include design automation analysis of algorithms.