Single Row Routing*

Raghunath Raghavan and Sartaj Sahni
University of Minnesota

Abstract

The single row routing problem is considered. It is shown that the use of backward moves can reduce street congestion when four or more tracks per street are available. We obtain an O((2k)!k*n*log k) algorithm to determine whether or not n nodes can be wired when only k tracks per street are available. An efficient algorithm is obtained for the case when wires are not permitted to cross streets. This case is shown to be related to a furnace assignment problem.

Keywords and Phrases: single row routing, complexity, algorithm.

*This research was supported in part by the National Science Foundation under grant MCS80-005856.

1. Introduction

The design of multilayer printed circuit boards (MPCB's) is of great importance in the design of complex electronic systems. MPCB design and layout involves the following steps:

(1) placement of the functional modules of the system on the MPCB, and

(2) conductor routing, subject to various physical constraints, on the MPCB to effect the necessary interconnections between the modules.

Clearly, these two steps are intimately related. However, the complexity of the total layout problem is so great that, traditionally, these two steps have been treated separately [BREU72].

This paper addresses some aspects of the latter step. Given module locations on the MPCB and lists of points to be made electrically common, one is concerned with the definition of the conductor paths on the MPCB that satisfy all the requirements and constraints.

Wire routing on MPCB's is a problem that arises at many levels of the interconnection hierarchy. For example, the modules might be IC's and the MPCB the circuit card; or the modules might be the circuit cards themselves and the MPCB the backplane. To simplify the discussion that follows, the term MPCB will be taken as referring to the interconnection medium at the most obvious level of the hierarchy.

In very large systems, where the number of interconnections might be in the tens of thousands, the MPCB tends to have a regular geometry. The points to be connected (module pins and feedthroughs, both of which appear as plated-through holes) are uniformly spaced on a rectangular grid. For such cases, Hing So [SO74] has proposed a systematic decomposition of the general multilayer routing problem into a number of independent single layer single row routing problems . He terms this approach unidirectional routing.

Prior to this, multilayer routing was done using multilayer versions of Lee's algorithm and other limited maze-running algorithms [BREU72]. Such approaches gave no estimate of the inherent routability of the problem. Hing So, on the other hand, developed sufficient conditions based on his single row routing decomposition that give a realistic estimate of routability.

Apart from having this predictive ability, single row routing produces wire layouts that are more amenable to automated fabrication, since all conductors on a given layer are predominantly either horizontal or vertical. This makes single row routing important for its own sake: as a legitimate approach to the solution of the general multilayer routing problem, rather than just being a good way to estimate routability.

In the single row routing problem, we are given a set V={1,2,...,n} of n nodes that are evenly spaced along a straight line; and a set L={N_1, N_2,..., N_n} of nets. Each net represents a set of nodes that are to be made electrically equivalent. The nets satisfy the following conditions:

(i) $N_i \ N_j = , \ i \neq j$

(ii) $\underset{i}{\cup} N_i = \{1,2,...,n\}$

The wires used to join together the vertices of a net are made up of horizontal and vertical segments. Figure 1.1 shows some of the possible ways to wire the net N_i = {2,5,8} when V = {1,2,...,8}. Note that it is permissible for vertical wire segments to cross from one side of the nodes to the other (as in Figures 1.1(b), (c) and (d)).

In the development of So [SO74], configurations such as the one in Figure 1.1(d) are not permitted. This configuration differs from the others in that the wire joining nodes 2 and 5 makes a backward move; it goes from 2 to beyond 1, and then back over 2. So considers only those wiring schemes in which such backward moves are not permitted. More formally, he restricts the wiring of nets such that if a vertical cut is made at any point along the axis defined by the nodes (see Figure 1.1(d)), this cut intersects at most 1 wire from each net.

Finally, horizontal wire segments are run in tracks. Two wires cannot share (overlap within) a segment of the track. Also, vertical wire segments are not permitted to cross over horizontal wire segments, and vice versa.

Figure 1.1: Some ways to wire a net.

A *realization* of a net set L is a wiring scheme that satisfies all the above requirements, including the cut requirement of So. A *realization with backward moves* is a wiring scheme that satisfies the above requirements, except possibly the cut requirement of So. The tracks above the line of nodes form the *upper street*, while those below form the *lower street*. We shall use C_u and C_l respectively to denote the maximum number of tracks required in the upper and lower streets. C_u and C_l are respectively the upper and lower street *congestions*. We shall use the term *wiring width* to denote the quantity C = max{C_u, C_l}. In the single row routing problem, we wish to obtain a wiring that has minimum wiring width.

Example 1.1: Consider the following single row routing problem instance:

$$n = 10, L = \{\{1,7\},\{2,8\},\{3,6\},\{4,9\},\{5,10\}\}$$

Figure 1.2 gives a realization of this instance that has $C_l = C_u = 3$. Thus the wiring width is C = max{C_u,C_l} = 3. □

A complete set of necessary and sufficient conditions for optimal single row routing has been developed by Kuh et al. [KUH79]. These conditions are valid only when backward moves are not permitted. Up to this time, however, no efficient algorithm to obtain an optimal wiring has been presented.

In this paper, we address a number of issues associated with optimum single row routing. We begin, in Section 2, by addressing the question of whether or not there exist single row routing instances for which the optimal realization with backward moves is better than the optimal realization when such moves are forbidden. This question is answered in the affirmative. It is shown that every realization with backward moves that has a wiring width of 1 or 2 can be transformed into a realization without backward moves that also has a wiring width of 1 or 2,

Figure 1.2: Realization for Example 1.1

respectively. Further, there exist instances for which the optimal wiring with backward moves has a width of 3 while that without such moves has a width of 4.

In Section 3, an O((2k)!*k*n*log k) algorithm is presented for wire layout when each street has a capacity of k tracks. In realistic situations, k will be small (2,3 or 4) and n large. Thus, this algorithm represents a significant improvement over the O(m!*n) (m is the number of nets) algorithm mentioned in [KUH79]. Subsequently, an O(m*n) algorithm has been developed for the case $k \leq 2$ by Tsukiyama et al. [TSUK80]. It is easy to see that the algorithm presented in Section 3 outperforms this algorithm, at the same time being less restrictive about the street capacities it can handle.

In Section 4, a restricted version of the optimum single row routing problem is studied. In this version, the wires are not allowed to cross between the upper and lower streets. Efficient algorithms are presented for wire layout.

Finally, in Section 5, the restricted version of the problem considered above is shown to be related to a problem in a different area. This related problem is that of optimally assigning jobs to furnaces when only two furnaces are available. The extension of this problem to the case of k furnaces corresponds to single row routing with k streets with no crossovers allowed. It is shown that this extension is NP-hard.

2. Routing with backward moves allowed

The formulation of the problem in [KUH79], where the net lists are represented as interval graphs, has greatly increased the understanding of the intricacies of the single row routing problem. However, this formulation restricts the number of possible wiring configurations. In particular every wiring configuration is such that there is no vertical cut that intersects two or more wires from the same net. Situations such as those depicted in Figure 2.1 are therefore prohibited.

It is not intuitively obvious that removing the cut requirement, i.e., allowing backward moves, will enable one to obtain realizations of smaller width. However, such is indeed the case.

In Figure 9 of [KUH79], an optimum realization of a particular net list (with no backward moves) is given. This is reproduced in Figure 2.2(a). This optimum realization has $C_u = 4$ and $C_l = 5$. So, $C = \max\{C_u, C_l\} = 5$. Figure 2.2(b) shows another realization, this one with backward moves allowed. For this realization, $C_u = C_l = 4$, which represents an improvement. Thus, allowing backward moves can reduce the optimal wiring width from 5 to 4.

In fact, it is possible to reduce the optimal street width from 4 to 3. Figure 2.3 shows a realization of a net list with $C_u = C_l = 3$. For this net list, there is no realization without backward moves that can do as well. We verified this latter statement experimentally. The O(m!*n) exhaustive search algorithm mentioned in [KUH79] was programmed in PASCAL and run on a

Figure 2.1

CDC 6400. It turned out that every realization generated had $\max\{C_u, C_l\} \geq 4$.

We have seen that allowing backward moves can be of help when we have only three tracks in the upper and lower streets. When only two tracks (or only one track) per street are available, nothing is to be gained by allowing backward moves. To see this, first consider the case of two tracks per street. Consider any realization that has backward moves. Thus, there is a vertical cut that intersects two or more wires from the same net.

Figure 2.4

If two of these are in the same street then the two wires may be combined into one (as in Figure 2.4). So, we may assume that if a vertical cut intersects wires from the same net, exactly two such wires (per net) are intersected and these are in different streets. The different possibilities are symmetric to the two given in Figure 2.5. This figure shows how both possibilities may be eliminated without increasing the wiring width. In some cases the segment (a-b) may be null. Using the transformations of Figures 2.4 and 2.5, every layout with backward moves that has a width of 2 can be transformed into a layout without backward moves that also has a width of 2.

It is a trivial matter to verify that when only one track per street is available, all realizable net sets can be realized without backward moves. Our discussion leads to the following theorem:

Theorem 2.1: Let (n,L) be a single row routing instance. Let C be the minimum width needed to realize (n,L) without backward moves. Let C_b be the minimum width needed when backward moves are permitted. $C = C_b$ for $1 \leq C \leq 3$ and $C \geq C_b$ when $C \geq 4$. Moreover, when $C \geq 4$, there exist (n,L) for which $C > C_b$. □

Figure 2.2

3. Routing with k tracks per street

Upto this time, an efficient algorithm to solve the optimum single-row routing problem (without backward moves) has not been found. The only algorithm presented so far that finds the optimum solution is an O(m!*n) algorithm mentioned in [KUH79]. Since m, the number of nets,

Figure 2.3

Figure 2.5

can be very large in realistic problems, this algorithm is of no practical value.

One might suspect that it should be possible to do better than O(m!*n). While m can be very large, the nature of the decomposition from the multilayer single row problem to the single layer single row problem ([SO74],[TING76],[TING78]) is such that the number of tracks needed in each layer is small. Thus, a large number of nets, if present, would be 'strung out' along the line. Nets that are far apart would not significantly influence each other's layout.

Two algorithms, POSSIBLE and RECONSTRUCT, are presented in this section.

Algorithm POSSIBLE determines whether a given net list can be routed using at most k tracks per street. This algorithm can be used to determine the minimum wiring width required, as described below.

The *cut number* of a node is the number of nets covering that node, i.e., the number of nets between whose extreme nodes the node in question lies.

Let q_M = the maximum cut number in the net list
q_o = the wiring width required by the optimum realization.
q_M can be easily determined a priori. q_o is the quantity to be determined.
From [SO74], $q_o \leq q_M - 1$.
From [KUH79], $q_o \geq q_t = \lceil q_M/2 \rceil$.
Now, to determine q_o, one could proceed as follows:

1. $q_o \leftarrow \lceil q_M/2 \rceil$
2. *while* $q_o \leq q_M$-1 *do*
3. *if* net list can be routed with q_o tracks/street
   then exit
   endif
4. $q_o \leftarrow q_o + 1$
5. *endwhile*
6. *return* (q_o)

An alternate approach might be to perform a binary search, as below:

1. low $\leftarrow \lceil q_M/2 \rceil$; high $\leftarrow q_M - 1$
2. *while* low $\neq$ high *do*
3. $q_o \leftarrow$ (low+high)/2
4. *if* net list can be routed with q_o tracks/street
   then high $\leftarrow q_o$
   else low $\leftarrow q_o + 1$
   endif
5. *endwhile*
6. *return*(low)

In the worst case, the latter approach makes fewer iterations of the *while* loop than the former. However, because of the complexity of line 4, it is practical to execute this line only for small q_o (say at most 4 or 5). Hence, it is quite possible for the binary search approach to fail (because of large computing requirements) when the sequential approach succeeds. For example, suppose q_M = 10, and q_o is in fact 5. The former method takes only one pass through the loop, while with the latter, q_o = 7 on the first iteration, and at least one more pass is required. Furthermore, it may be computationally infeasible to do line 4 with q_o = 7.

Algorithm RECONSTRUCT is not formally presented as an algorithm. Instead, modifications to algorithm POSSIBLE are outlined that will allow the determination of the exact conductor paths.

The basic idea behind both algorithms, POSSIBLE and RECONSTRUCT, is as follows. Since there are only 2k tracks totally, there are only (2k)! ways in which the wires of the various nets can be ordered in arriving at a node. Therefore, by considering the (2k)! possibilities at each of the n nodes of the net list, all possible wire layouts are covered.

The algorithms consider the possibilities in a systematic way, keeping track of the relationships between the possibilities in the process. Both algorithms have a worst case time complexity of O((2k)!*k*n*log k). Even though this contains a (2k)! term, since k is expected to be small (2,3 or 4), a legitimate case can be made for the practicality of these algorithms.

Each possible ordering of wires that can be encountered by a node can be maintained simply as an ordered list of net indices. There is no explicit division of the nets between the upper

and lower streets. Such an explicit division is neither desirable nor necessary. As will be explained subsequently, street overflow conditions can be detected easily without knowing this division. Only the relative order (top to bottom) is relevant; this same order is reflected in the realization.

Figure 3.1

An advantage of this approach is that functionally equivalent situations, such as the ones in Figure 3.1(a) and (b), are not treated separately. Such functionally equivalent situations arise when there is more than one feasible way to assign an ordered list of nets ((N_2,N_3,N_1,N_5) in Figure 3.1) to the 2k available tracks. Avoiding such a replication of information substantially reduces the number of possibilities that need to be considered.

Before presenting the algorithms, it is necessary to introduce some terminology. The nodes of the net list are classified into one of the following three categories:

1) type B node:
   a node at which a net begins; the left extreme node of a net
2) type E node:
   a node at which a net ends; the right extreme node of a net
3) type M node:
   a "middle", or non-extreme, node of a net; conductor segments both begin and end at such nodes.

Let N_{x_i} = the index of the net to which node i belongs.
Let p_i = an ordering of wires encountered by node i. Each wire is represented by the index of the net it belongs to. In Figure 3.2, p_6 = (4,1,3).
Let P_i = $\{p_i\}$
= the set of all the feasible orderings of wires that can be encountered by node i.

Figure 3.2

The set P_i will be maintained as a trie [HORO76], as illustrated by the following example.

Example 3.1
$P_i = \{p_{i_1}, p_{i_2}, p_{i_3}, p_{i_4}, p_{i_5}\}$

$p_{i_1} = (4,1,2,6)$
$p_{i_2} = (4,1,6,2)$
$p_{i_3} = (1,2,4,6)$
$p_{i_4} = (1,4,2,6)$
$p_{i_5} = (6,4,1,2)$

Figure 3.3 shows the trie representation of P_i. □

From the example, we can deduce the following facts, which will hold for all trie representations of P_i's:

(a) number of fields/node ≤ 2k.
When looking for a particular ordering in the trie, a binary search is required at each node encountered to determine the appropriate field from which to branch.

(b) number of levels in the trie ≤ 2k.

(c) number of leaf nodes ≤ (2k)!.
Clearly each leaf node represents an ordering, and there can be no more than (2k)! distinct orderings.

(d) all the leaf nodes are at the same level of the trie

(e) the time required to look for an ordering in the trie is 2k log (2k) in the worst case. To see this, note that
(i) one node is visited at each level
(ii) time required at each node ≤ log (2k)
(iii) number of levels ≤ 2k.

(f) the time required to insert a new ordering into the trie is O(k log k) in the worst case. To see this, let the new ordering agree with an existing ordering in the first r elements. Then, the time required to make the insertion is less than or equal to r log 2k + 2k - r

Figure 3.3

= O(k log k) in the worst case, as $0 \leq r \leq 2k$. An algorithm to do the insertion is not difficult to arrive at.

(g) time required to output all the orderings represented by the trie is O((2k)!*k). To see this, note that there can be no more than (2k)!*2k fields in the whole trie. In a systematic traversal, each field is visited O(1) times.

We are now in a position to present the algorithms.

3.1 Algorithm POSSIBLE

Recall that P_i is the set of all the feasible orderings of wires that can be encountered at node i. Starting with P_1 = , the algorithm POSSIBLE systematically generates $P_2, P_3, \ldots, P_{n+1}$. Each P_{i+1} is determined from P_i and the type (B,E or M) of node i. This is explained below on a case-by-case basis:

Case 1: Node i is *type B*; i.e., net N_{x_i} begins at node i. Each ordering $p \in P_i$ generates a number of orderings $p'_l \in P_{i+1}$, since there are a number of places where N_{x_i} can enter the ordering p.

Let $P = \begin{matrix} N_{j_1} \\ N_{j_2} \\ N_{j_3} \\ . \\ . \\ N_{j_r} \end{matrix}$ be the ordering. Clearly, $r \leq k$.

If r = 2k, it is clear that adding N_{x_i} to p will definitely cause an overflow, resulting in

infeasibility. Since none of the orderings p ∈ P_i can generate orderings p' ∈ P_{i+1} without causing infeasibility, P_{i+1} = . (Recall that $p_a \in P_j$ and $p_b \in P_j$:> $|p_a| = |p_b|$ from fact (d) above, where $|p_x|$ = number of wires in p_x).

Figure 3.4

Figure 3.4 illustrates the various ways in which N_{x_i} can be added to the ordering p when r = |p| < 2k. Figure 3.4(a) illustrates the case when r ≤ k, and Figure 3.4(b) the case when k < r < 2k. In Figure 3.4(b), AA is the highest point in the list where N_{x_i} can be inserted. If N_{x_i} attempts to enter the list above this point, the number of conductors that will have to be in the lower street to allow this causes an overflow in the lower street. An analogous argument can be used to explain why N_{x_i} cannot enter the list below BB. The regions above AA and below BB will be termed "forbidden zones". From Figure 3.4(a), we see that the number of orderings p' ∈ P_{i+1} generated from p is r+1 ≤ k+1. From Figure 3.4(b), we see that the number of orderings p' ∈ P_{i+1} generated from p is k-(r-k)+1 = 2k-r+1 ≤ k.

Case 2: Node i is *type E*; i.e., net N_{x_i} ends at node i.
Each ordering p ∈ P_i can generate at most one ordering p' ∈ P_{i+1} as explained below.

First of all, note that if N_{x_i} wants to end at node i, all the nets below N_{x_i} in the ordering p have to be in the lower street at node i, and all the nets above it have to be in the upper street. Therefore, depending on the size of p and the location of N_{x_i} in the ordering, some orderings p will not allow N_{x_i} to end without causing an overflow. Every ordering p that does not allow N_{x_i} to terminate has the following characteristics:

(i) |p| > k
(ii) N_{x_i} is in a "forbidden zone" (see Figure 3.4(b)) of p.

From each ordering p that does allow N_{x_i} to end, the corresponding order p' $\in P_{i+1}$ is derived by simply deleting N_{x_i} from the order. Note that this might result in two different orderings p_a , $p_b \in P_i$ giving rise to the same ordering p' $\in P_{i+1}$. The duplicate orderings are automatically eliminated since each P_j is being maintained as a trie, in which duplicates are not permitted.

Case 3: Node i is *type M*;
Here, P_{i+1} is obtained from P_i by simply removing all those orderings that do not allow N_{x_i} to connect to node i without causing an overflow in one of the streets.

These different cases can now be amalgamated into the algorithm POSSIBLE (Figure 3.5).

Analysis Of Algorithm POSSIBLE

Let m = number of nets in net list
n = number of nodes
k = number of tracks/street.

Time Complexity

. time taken for each pass through case :B:
= O((2k)!*k*log k) in the worst case.

At first sight, it might appear that O((2k)! k^2 log k) time is required for this. However, by interleaving ordering-generation and insertion (lines 7 and 8), the claimed time complexity can be achieved.

. number of times case :B: is entered = m

. time taken for each pass through case :E:
= O((2k)!*k*log k) in the worst case.

. number of times case :E: is entered = m

. time taken for each pass through case :M:
= O((2k)!*k*log k) in the worst case.

. number of times case :M: is entered = n-2m

Overall time complexity = O((2k)!*k*n*log k)

Space Complexity

It is clear that once P_{i+l} has been generated from P_i, P_i is not needed any longer, and the space used by P_i can be reclaimed for use by P_{i+1} at the end of the iteration.
The space required for each trie = O((2k)!*k) in the worst case. This is the space complexity of the algorithm.

```
0.  procedure POSSIBLE
1.  P_1 ← ; i ← 1
2.  while i ≤ n do
3.    P_{i+1} ←
4.    case type (node i) of
5.          :B:
6.           for each ordering p ∈ P_i do
7.             derive all orderings p' obtained
                    by inserting N_{x_i} into p
8.             insert each such ordering p'
                    into the trie for P_{i+1}
9.           endfor
10.         :M:
11.          for each ordering p ∈ P_i do
12.            if p allows N_{x_i} to connect to i
                 without causing an overflow in
                 one of the streets
13.            then insert p into the trie for P_{i+1}
14.            endif
15.          endfor
16.         :E:
17.          for each ordering p ∈ P_i do
18.            if p allows N_{x_i} to end at node i
                 without overflow
19.            then p' ← p with N_{x_i} deleted
20.                 insert p' into trie P_{i+1}
21.            endif
22.          endfor
23.   endcase
24.   if P_{i+1} = then return ('infeasible')
25.   i ← i + 1
26. endwhile
27. return ('feasible')
28. end POSSIBLE
```

Figure 3.5

3.2 Algorithm RECONSTRUCT

The only differences between the two algorithms center around the necessary bookkeeping. In Algorithm POSSIBLE, the bookkeeping is extremely simple: the space used by P_i is re-used by P_{i+1}.

In Algorithm RECONSTRUCT, the complete wiring layout has to be determined. In order to do this, it is necessary to maintain all the P_i's. Furthermore, it is necessary to keep track of how the various orderings were generated, i.e., maintain an association between each ordering and all the orderings generated by it. This is best done by maintaining a directed acyclic graph of orderings, as shown in Figure 3.6. Each node represents an ordering. Directed edges are from the creator ordering to the created ordering(s). As shown in the figure, the graph can be partitioned into the various P_is.

To complete the bookkeeping, the following node deletion rule is used. If a node in any partition other than P_{n+1} fails to generate successors, that node is "killed", and the liveness of its

Figure 3.6

predecessor is re-examined. Thus, if the wiring is not feasible with k tracks/street, the graph will eventually destroy itself.

If this does not happen, any path from P_1 to any node in P_{n+1} will give n orderings from which the actual wiring pattern can be determined very easily.

Analysis Of Algorithm RECONSTRUCT

Quite clearly, the space complexity of the algorithm is O((2k)!*k*n). Therefore, it is necessary that k be reasonably small for this algorithm to be useful.

4. Routing With No Crossovers

In this section, a restricted form of the single row routing problem is considered. Given a net list, the objective is to determine a wire layout that minimizes $\max\{c_u,c_l\}$, subject to the additional constraint that no wire may cross from one street to another between two nodes. While in [KUH79] it was shown that all net lists can be wired when crossovers are permitted, this is not the case when crossovers are not permitted.

The restriction that wires not crossover at points between nodes allows one to decompose nets that contain more than two nodes into a set of nets, each containing exactly two nodes. This decomposition is shown in Figure 4.1. Since no wires can be run between 2^1 and 2, or 3^1 and 3, these node pairs can be coalesced after the wire layout for the net list $\{\{1,2^1\},\{2,3^1\},\{3,4\}\}$ has been obtained. This coalescing yields a wiring for the net {1,2,3,4}.

We begin this section by developing an algorithm, FEASIBLE, to determine whether or not a set of nets can be wired with no crossovers. This algorithm assumes that nets have already been decomposed such that each net contains exactly two nodes (Figure 4.1). In Section 4.2, we develop (informally) a dynamic programming algorithm to obtain wirings of minimum width.

Figure 4.1

4.1 Determining feasibility

With the restriction that cross-overs between nodes are not allowed, not all net lists can be wired. Figure 4.2 illustrates this fact.

Figure 4.2

Therefore, before attempting to wire up a net list, it is necessary to determine whether or not it is possible to perform the wiring.

Before presenting algorithm FEASIBLE, some terminology is necessary:

An interlocked set is a maximal subset of the net list, such that the assignment of any net in the set to a street forces the assignment of all the other nets in this set to appropriate streets. Figure 4.3 illustrates the concept. The nets of Figure 4.3(a) do not constitute an interlocked set, while those of Figure 4.3 (b) and (c) do.

The domain of a net is the set of nodes that lie between its end points. The domain of the net {1,4} (Figure 4.3(c)) is {2,3}.

The length of a net is the size of its domain.

Figure 4.3

With these three definitions, it becomes possible to present the feasibility algorithm. The algorithm attempts to partition the net list into its constituent interlocked sets. Quite clearly, the net list can be wired feasibly if and only if such a partition exists.

The algorithm is presented (Figure 4.4) in semi-formal form. A few more data structures are necessary for implementation in a conventional programming language.

Each iteration of the major loop (lines 8-31) outputs one interlocked set. Each interlocked set is built up by the repeat loop.

Analysis Of The Algorithm

Since the interlocked sets are disjoint, the statement on line 17 can be executed at most m times. Thus, the block of lines 16-28 is executed at most m times.

Each time through this block, the *for* loop (lines 18-28) is iterated a number of times equal to the length of the net under consideration at line 17. Each iteration of the *for* loop takes constant time.

Since the worst case length of each net is O(m), the worst case time complexity of the algorithm = $O(m^2)$

4.2 Determining The Optimum Street Width

Once the feasibility of doing the wiring has been established, it is necessary to determine the wiring layout. In what follows, an algorithm is presented that finds the optimum value of max $\{C_u, C_l\}$. The layout itself is not explicitly determined. However, it can be easily reconstructed with the information provided by the algorithm.

At this point, the basic entities one is dealing with are the interlocked sets produced by algorithm FEASIBLE. Before going any further, it is necessary to establish a structural relationship between the interlocked sets.

Definition : *The domain* of an interlocked set is the set of nodes between the extreme nodes in the set. *A gap* in an interlocked set is a contiguous set of nodes in the domain that do not belong to any of the nets in the interlocked set.

Since the interlocked sets are independent of each other (in the sense that each interlocked set can be laid out independent of the other sets), there are only two ways in which two interlocked sets can be related to each other:

```
0.  algorithm FEASIBLE
1.  set S       /*interlocked set*/
2.  queue a
3.  array asigned [1:n] of Boolean /*m=number of nets */
4.  array street [1:m] of {UP,DOWN}

5.  initialize a to be empty
6.  for i ← 1 to m do assigned [i] ← false
7.  loop
8.    S ←
9.    if all nets assigned then exit   /*loop*/
l0.   j ← index of first unassigned net
11.   assigned [j] ← true
12.   street [j] ← UP   /*arbitrarily*/
13.   enqueue (j,a)
14.   repeat
15.       k ← dequeue(a)
16.       S ← S ∪ {N_k}
17.       for each net N_p with one end in N_k's domain
            and the other outside do
18.         if assigned [p]
19.         then if street[p] = street[k]
20.                then
21.                  return ("infeasible")
22.                endif
23.         else street[p] ← opposite (street[k])
24.              assigned [p] ← true
25.              enqueue (p,a)
26.         endif
27.       endfor
28.   until queue_empty(a)
29.   output(S)
30. forever
12. end FEASIBLE
```

Figure 4.4

(i) their domains are disjoint, or
(ii) one set is entirely contained in a gap of the other.

A moment's reflection will show that there is no other relationship possible between two interlocked sets.

The relationships between the various interlocked sets can be captured by establishing a partial order, based on containment, between the sets. The interlocked set S_2 is *contained* in the interlocked set S_1 iff it is entirely contained in a gap of S_1.

By introducing a dummy net that contains all the other nets, a partial order tree can be defined. Each node in the tree represents an interlocked set. The children nodes of each node represent interlocked sets that are contained within the interlocked set it represents. If S_1 and S_2 are two interlocked sets such that S_2 is contained in S_1, then S_1 is an ancestor of S_2.

Let max(i) and min(i), respectively, be the largest and smallest indexed nodes in the interlocked set S_i. One may verify that for the partial order tree defined above, the following statments

are true:

(1) If max(i) > max(j) and min(j) - 1 ∈ S_i, then S_i is the parent of S_j.

(2) If max(i) = min(j) - 1, then S_i is a sibling of S_j.

(3) If S_i is a sibling of S_j and S_j is a sibling of S_k, then S_i is a sibling of S_k.

By making use of these three facts, we can arrive at an efficient algorithm (Figure 4.5) to construct the partial order tree.

```
 0. algorithm PARTIAL_ORDER
   //m=number of nets; all nets are of size =2//
   //r=number of interlocked sets output by algorithm
                              FEASIBLE//
   //n=number of nodes//
 1. create the dummy interlocked set S0={0,n+1}
      that contains all the other sets
 2. SIB(i)←PARENT(i)←0, 1≤i≤r
 3. for j←1 to r do
 4.    i←index of the interlocked set that contains node
          min(j)-1
 5.    if max(i) > max(j) then PARENT(j)←i
 6.                  else SIB(i)←j
 7.    endif
 8. endfor
       //set remaining parent links//
 9. for j←1 to r do
10.    k←j
11.    p←PARENT(j)
12.    while SIB(k) ≠ 0 do
13.        q←k; k←SIB(k); SIB(q)←0
14.        PARENT(k)←p
15.    repeat
16. endfor
17. end PARTIAL_ORDER
```

Figure 4.5

In line 1 of algorithm PARTIAL_ORDER, a dummy interlocked set S_0 = {0,n+1} is created. This has the property that every interlocked set S_i is contained in it. In the **for** loop of lines 3 to 8, each interlocked set (except S_0) either determines its parent set or one of its siblings (or both). Observe that when this loop is exited, the first node on every chain linked by SIB is such that its PARENT has also been determined. In the loop of lines 9 to 16, we follow down chains of siblings and set the PARENT fields. The dummy set S_0 represents the root of the partial order tree. The time complexity of the algorithm is easily seen to be O(r).

Once the partial order tree has been constructed, the street capacity required for the optimum configuration may be determined using dynamic programming, as explained below.

The basic operation used is the *node merge*, in which two interlocked sets are merged to form a single sub-assembly. This operation is illustrated below using an example.

Example 4.1: Figure 4.6(a) shows two nodes A and B of the partial order tree. A is the parent of B (note that A may have other children). The layout for the interlocked sets associated with each

Figure 4.6

node is also given. Corresponding to each layout, there is another possible layout. This is obtained by flipping the given layout about the axis defined by the nodes. The layout of node B can be inserted into that of node A in one of two ways. Figures 4.6(b) and (c) give the two possibilities. □

One way to obtain an optimal layout is to systematically merge nodes into their parents, keeping track of all possible outcomes. First, all leaf nodes are merged into their parents; next the new leaf nodes are merged into their parents, and so on. Since the root represents a dummy

interlocked set, the children of the root need not be merged into the root. Instead, these may simply be concatenated.

Before presenting an example to illustrate the merging process, some notation is introduced:

Let m = number of nets
r = number of interlocked sets
= number of nodes in partial order tree - 1. Clearly, $r \leq m$.
Let g_i = number of gaps in nodes i, $1 \leq i \leq r$. Clearly, for the leaf nodes, $g_i=0$.

With respect to a fixed orientation, each node is characterized by g_i+1 tuples:
. the tuple (u_{i0},l_{i0}) represents the interlocked set's maximum track requirements in the upper and lower streets in the non-gap portions of the set;
. the tuple (u_{ij},l_{ij}), $1 \leq j \leq g_i$, represents track requirements in the upper and lower streets at the j^{th} gap.

Figure 4.7

Example 4.2: For the interlocked set of Figure 4.7, $g_i=3$ and
$(u_{i0},l_{i0}) = (3,2)$
$(u_{i1},l_{i1}) = (0,1)$
$(u_{i2},l_{i2}) = (1,0)$
$(u_{i3},l_{i3}) = (1,1)$ □

The next example illustrates the use of the merging operation in determining the track requirements of the optimum configuration.

Example 4.3: Figure 4.8 represents a partial order tree after nodes 1 and 2 have been merged into 3. The upper and lower street track requirements of node 3 are:

(u_{30},l_{30})
$(u_{31}+u_{10},l_{31}+l_{10})$
$(u_{31}+l_{10},l_{31}+u_{10})$
$(u_{32}+u_{20},l_{32}+l_{20})$
$(u_{32}+l_{20},l_{32}+u_{20})$

Simultaneously, node 4 has been merged into node 5 to generate the following set of track requirements at node 5

Figure 4.8

(u_{50},l_{50})
(u_{51},l_{51})
$(u_{52}+u_{40},l_{52}+l_{40})$
$(u_{52}+l_{40},l_{52}+u_{40})$

Figure 4.9 depicts the situation after this first set of merge operations. Finally, nodes A' and B' are merged, giving rise to the following list of all possible track requirements:

(u_{50},l_{50})
$(u_{51}+u_{30},l_{51}+l_{30})$
$(u_{51}+l_{30},l_{51}+u_{30})$
$(u_{51}+[u_{31}+u_{10}],l_{51}+[l_{31}+l_{10}])$
$(u_{51}+[l_{31}+l_{10}],l_{51}+[u_{31}+u_{10}])$
$(u_{51}+[u_{31}+l_{10}],l_{51}+[l_{31}+u_{10}])$
$(u_{51}+[l_{31}+u_{10}],l_{51}+[u_{31}+l_{10}])$
$(u_{51}+[u_{32}+u_{20}],l_{51}+[l_{32}+l_{20}])$
$(u_{51}+[l_{32}+l_{20}],l_{51}+[u_{32}+u_{20}])$
$(u_{51}+[u_{32}+l_{20}],l_{51}+[l_{32}+u_{20}])$
$(u_{51}+[l_{32}+u_{20}],l_{51}+[u_{32}+l_{20}])$
$(u_{52}+u_{40}, l_{52}+l_{40})$
$(u_{52}+l_{40}, l_{52}+u_{40})$

By looking through the list, one can determine the optimum track requirements at each gap

Figure 4.9

of the root set, and thereupon the overall optimum track requirement. □

From the way the merge process works, it is quite clear that all the possible layouts of the net list are generated. In order to determine the optimum layout, one has to look through all the generated solutions. This is quite expensive for large r as the number of possible layouts is exponential in r. A consideration of the merge operation shows that each tuple at a given node generates 2 tuples when that node is merged into its parent. Figure 4.10 depicts a worst-case scenario for the structure of the partial order tree. Thus, 1 tuple at node 1 becomes 2 tuples at node 2, 4 at node 3, 8 at node 4, etc., and 2^{r-1} at the root. Note that r=O(m).

The exponential growth of the total enumeration algorithm just described can be circumvented using dynamic programming. Let Y be any layout for a set of nodes. Let $c_L(Y)$ and $c_U(Y)$, respectively, denote the lower and upper street widths of Y. If Y and Z are two layouts for the same node set, then we shall say that Y *dominates* Z if and only if $c_L(Y) \leq c_L(Z)$ and $c_U(Y) \leq c_U(Z)$. If Y dominates Z, then Z is dominated by Y. It should be apparent that no layout obtained by merging Z into a parent layout can be better than the best obtainable by merging Y into a parent layout. So, dominated layouts may be eliminated from consideration.

If the number of nets is m, then $0 \leq c_L(Y) \leq m$ and $0 \leq c_U(Y) \leq m$. If a node of the partial order tree (together with its descendents) represents p different nets, then from the dominating rule, it follows that this node can have at most p+1 layouts that do not get dominated by other layouts. As a result, it is possible to merge the nodes of the partial order tree together in $O(m^2)$ time (note that $p \leq m$ and the number of nodes is $\leq m$). From the layouts obtained for the root, the optimal layout may be picked by determining the one with least $\max\{c_L(\ ), c_U(\ )\}$.

5. A Related Problem

A furnace consists of two elements:
(i) the heating element, which is in a compartment with a door, and
(ii) a rack of hooks, which is used to place objects into the furnace (Figure 5.1).

When an object wishes to enter the furnace, it is placed on the next available hook in the rack (the one nearest the door), and the hook advanced into the furnace as far as possible.

When an object is to be removed from the furnace, it has to be on the hook inside that is closest to the door; if there is any other object between it and the door, it cannot be removed from the furnace at this time (as such a removal will also require the removal of objects that are not

Figure 4.10

Figure 5.1

ready to be removed).

Furnace design involves deciding how many hooks to install on the rack. In other words, we need to determine the required furnace capacity.

An object *heating schedule* is a tuple (*ts*,*te*), where

ts = the exact time at which the object has to enter the furnace,
te = the exact time at which the object has to leave the furnace.

Two object heating schedules (ts_1, te_1) and (ts_2, te_2) are *non-conflicting* if and only if ts_1 , te_1 , ts_2 and te_2 are distinct.

5.1 The Optimal Two Furnace Design Problem

Given m object heating schedules that are pairwise non-conflicting, we are required to assign the m objects to the two furnaces such that the maximum number of hooks required on a rack is minimized.

It is not too difficult to see that this problem is identical to the optimum single row routing problem when the wires are not allowed to switch between the upper and lower streets. It is observed that:

. the object heating schedules correspond to the nets
. the two racks correspond to the two streets.

Assigning an object to a particular furnace corresponds to assigning a net to a particular street. Minimizing the maximum rack size amounts to minimizing the maximum street usage.

The algorithms developed in Section 4 can clearly be used to solve this problem.

5.2 The Optimal K-Furnace Design Problem

This is a natural extension of the two-furnace design problem. In this problem, one is concerned with m object heating schedules and k furnaces.

The dynamic programming algorithm developed in Section 4 can be extended to solve this problem. An $O(m^k)$ algorithm results. This is practical only for small k.

The question of whether or not the k-furnace problem can be solved by an algorithm that is polynomial in both m and k is related to the famous P=NP problem [GARE79]. Determining whether or not m objects can be scheduled on k furnaces using at most p hooks per furnace is NP-complete. It is easy to see that the problem is in NP. So, we need only be concerned with a proof of the fact that the problem is NP-hard. We shall prove this by showing that the existence of a polynomial time solution to the k-furnace design problem (or the equivalent wiring problem in k streets) implies the existence of an efficient solution to the 3-Partition problem, a known strongly NP-Hard problem [GARE79].

Proof Consider an arbitrary instance of 3-Partition:

Given, $A = \{a_1, a_2, ..., a_{3m}\}$;
$s : A \rightarrow Z^+$: a "size" function ;
$B \in Z^+$;
Further, it is given that

$$\sum_{i=1}^{3m} s(a_i) = mB$$

and $\frac{B}{4} < s(a_i) < \frac{B}{2}$, $1 \leq i \leq 3m$.

The question is whether A can be partitioned into m subsets, the sizes of the elements of each of which sum to B.

Given an arbitrary instance of 3-Partition an equivalent instance of the k-street single row routing problem is obtained in the following way.

For each a_i, $1 \leq i \leq 3m$, the equivalent ensemble is given in Figure 5.2. Each wire of the *enforcer subassembly* has to be assigned to a different street. This then forces all the wires of the size subassembly to be assigned to the remaining street. Hence, the name.

Figure 5.2

The 3m ensembles are then put together as shown in Figure 5.3. Figure 5.3(a) depicts the pictorial short-hand to be used for an ensemble. Figure 5.3(b) shows how the ensembles are put together. This net list is now wired optimally in m streets.

It is not hard to see that the optimum street capacity is 3m-3+B if and only if there exists a 3-Partition. Each street gets the size subassemblies of three ensembles that consititute a partition, and 1 wire from the enforcers of each of the other 3m-3 ensembles.

This completes the proof of NP-hardness. □

6. Conclusions

We have studied several aspects of the single row routing problem. It was shown that backward moves can result in layouts with smaller widths. For the case when no backward moves are permitted, we developed an algorithm that is practical for small k. By adding the no crossover restriction, the problem can be solved in $O(m^2)$ time, where m is the number of nets in the reduced problem (i.e., each net has exactly two nodes). This restricted problem was seen to be equivalent to a furnace design problem. Finally, we showed that the furnace design problem with many furnaces is NP-hard.

Figure 5.3

REFERENCES

[BREU72] M. A. Breuer [ed.], *Design Automation of Digital Systems*, Prentice-Hall 1972.

[GARE79] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman, 1979.

[HORO76] E. Horowitz and S. Sahni, *Fundamentals of Data Structures*, Computer Science Press, 1976.

[KUH79] E. S. Kuh, T. Kashiwabara and T. Fujisawa, "On Optimum Single Row Routing", *IEEE Transactions on Circuits and Systems*, Vol. CAS 26, No. 6, June 1979, pp. 361-368.

[SO74] H.C. So , "Some Theoretical results on the Routing of Multilayer Printed-Wiring Boards", *IEEE Symposium on Circuits and Systems 1974*, pp. 296-303.

[TING76] B. S. Ting, E. S. Kuh and I. Shirakawa , "The Multilayer Routing Problem: Algorithms and Necessary and Sufficient Conditions for the Single-Row Single-Layer Case", *IEEE Transactions on Circuits and Systems*, Vol. CAS 23, No. 12, December l976, pp. 768-778.

[TING78] B. S. Ting and E. S. Kuh, "An Approach to the Routing of Multilayer Printed Circuit Boards", *IEEE Symposium on Circuits and Systems 1978*, pp. 902-911.

[TSUK80] S. Tsukiyama, E. S. Kuh and I. Shirakawa, "An Algorithm for Single-Row Routing with Prescribed Street Congestions", *IEEE Transactions on Circuits and Systems*, Vol. CAS 27, No. 9, September 1980, pp. 765-771.