Generalized terminal connectivity problem for multilayer layout scheme

Chia-Chun Tsai*, Sao-Jie Chen and Wu-Shiung Feng

Given a set of n horizontal (or vertical) wire segments run on different layers with variable widths (or heights), and a set of m terminals placed on different layers and with arbitrary rectangular shapes, a generalization of the terminal connectivity problem (TCP) is considered. This TCP can be applied to facilitate the VLSI or PCB multi-layer layout. First, it is proved that this TCP is NP-hard by showing that it is equivalent to a minimal steiner tree problem, which has been proved NP-complete. Then an efficient algorithm for the TCP is presented which runs in O(m + (1 + c)n) time (with some preprocessing work). Experimental results are given to verify the effectiveness of the algorithm.

electronic design automation, terminal connectivity, shortest connectivity path, hyper-complete graph, minimal steiner tree

In this paper, a new problem concerning terminal connectivity is proposed as follows. Let there be a set of n horizontal (or vertical) wire segments orthogonally run on different layers with variable widths (or heights), and a set of m terminals placed on different layers and with arbitrary rectangular shapes. Assume that two wire segments on adjacent layers can be connected by a special wire segment (called a via) and that two or more horizontal (or vertical) wire segments on the same layer cannot overlap each other. First, it is necessary to check whether the given m separate terminals are interconnected by these segments or not. Then, if they are connected, the shortest connectivity path must be found by eliminating any redundant segments (loops, pendant or isolated segments); if they are not connected, the disconnected occurrences must be reported or highlighted. This problem is a generalization of the terminal connectivity problem (TCP)[1] and it corresponds to a net connectivity problem (NCP). Both TCP and NCP problems deal with two or more terminals and with wires connecting terminals in a VLSI or PCB multilayer layout scheme[2-4].

Institute of Electrical Engineering National Taiwan University, Taipei, Taiwan 10764.
* Also with the department of electronic engineering, National Taipei Institute of Technology Taipei, Taiwan.
Paper received: 12 July 1989. Revised: 21 November 1989

In practice, tens of thousands of routing segments are required to form an extremely complex mesh in a VLSI chip or a PCB multi-layer layout. Since a routing segment can be placed horizontally or vertically on different layers, after some basic operations (such as moving, pushing, plowing[5], compaction[6], via minimization[7], and so on) are applied, the correctness of the net connectivity will be affected without any checking being performed. These kinds of design errors, which may occur in the VLSI or PCB layout design, can be classified into the following categories: geometrical design rule error, topological or logical error, and electrical performance error. In this paper, topological error (especially circuit connectivity) is the main concern. Most connectivity errors found in manual design examples are trivial errors[8,9], such as unreasonably connected nets (short circuits between subcircuits) or isolated nets (open circuits between subcircuits). However, manual checking or verification of the extracted (hierarchical) circuit connectivity is tedious, time-consuming and error-prone. Therefore, the TCP will certainly play an important role in this issue; moreover, we can achieve part of the electrical or design-rule checking when performing TCP operations for distinct nets.

An analogous problem of finding a Manhattan path was presented in Lipski[10,11] and Asano[12] as follows: given a set of horizontal and vertical line segments, and specifying two line segments, s and t, find a path between s and t. Both Lipski and Asano presented an efficient algorithm which runs in $O(n\log n)$ time and takes $O(n\log n)$ space using a segment tree structure, where n is the number of horizontal and vertical segments. But this problem was limited to only two terminals (or two segments, s and t) and to no more than two layers. However, the TCP algorithm can find the shortest connectivity path through m ($m \geqslant 2$) terminals and on k ($k \geqslant 1$) different layers, and it runs in linear time.

A given terminal connectivity problem can be treated on a graph, G, composed of vertices (terminals or intersections) and edges (wire segments). Since finding the shortest connectivity path in the graph G is equivalent to finding a minimal steiner tree, which has been proved NP-complete[13], it will be shown that the TCP is NP-hard and needs to be solved by using heuristic techniques. To attack this generalized TCP, the problem

 0010-4485/90/070423-11

will be mapped onto a special data structure – multi-layer corner-stitching[14] – and present an efficient algorithm which runs in $O(m + (1 + c)n)$ time, where m and n are the numbers of terminals and wire segments, respectively, and c is a small positive constant which depends on the intersections between the wire segments.

Figure 1 shows an example of the TCP, consisting of seven separate terminals and 18 wire segments with two layers. Clearly, the seven separate terminals are not connected together because the terminal t_7 is isolated from the wire segment v_5. In another example of the TCP shown in Figure 2(a), the six separate terminals are connected together through the 16 wire segments and at least one connected path exists through these six separate terminals. But note that at least three loops in Figure 2(a) are formed and some of the segments are redundant for the finding of the shortest connectivity path. Hence, it is possible to find the shortest connectivity path through the six separate terminals by eliminating those redundant segments, as shown in Figure 2(b).

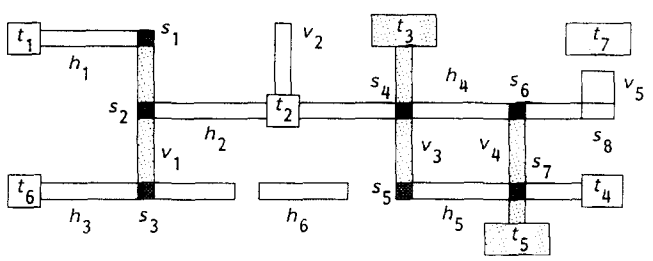


Figure 1. Example of a TCP with seven terminals and 18 wire segments, which is disconnected because of isolated terminal t_7

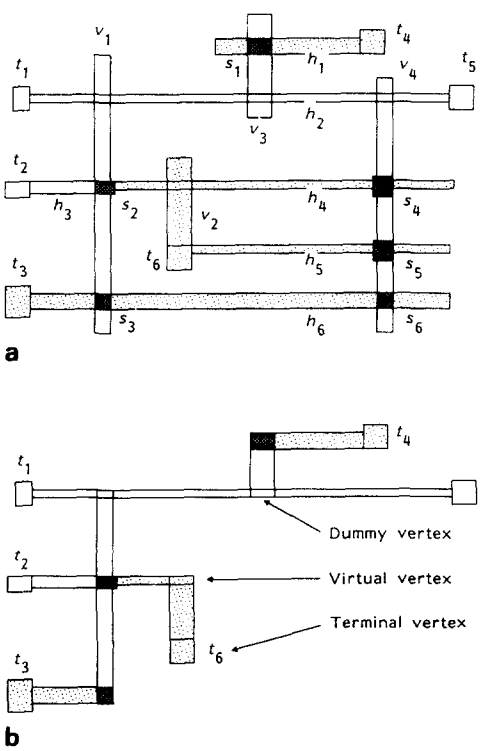


Figure 2. (a) Example of TCP with six teminals and 16 wire segments; (b) shortest connectivity path

In what follows, the TCP treated on a hyper-complete graph is proved NP-hard, and a heursitic technique is proposed for solving the TCP efficiently by mapping it into a multilayer corner-stitching data structure in which running time is linear. Some examples are presented to verify the effectiveness of the heuristic algorithm.

PROBLEM FORMULATION

Definition of terms

A given TCP consists of m terminals and n orthogonal (horizontal or vertical) segments on different layers. Intersections between the wire segments are called dummy terminals; these consist of bends in the same layer and vias between segments that occupy any two adjacent layers. In Figure 1, there are seven terminals and eight dummy terminals. A terminal or dummy terminal is also called a vertex or dummy vertex.

When a vertex is contacted by a segment, then the vertex and the segment are said to be incident (branching or outgoing) with each other. A terminal (or bend) has at most four incident segments while a via has at most eight incident segments in the TCP. In Figure 1, for example, segments h_2, v_2 and h_4 are incident with terminal t_2. Two vertices are said to be adjacent if they share an incident segment. In Figure 1, s_4 and s_6 are adjacent, but s_4 and s_7 are not. A terminal without an incident segment is called an isolated terminal. For example, terminal t_7 in Figure 1 is an isolated terminal. Similarly, a segment that is isolated from the connected path is called an isolated segment, such as h_6 in Figure 1. A segment in contact with a vertex at a unique point is called a pendant segment, such as v_2 and v_5 in Figure. A segment that has a part of itself as a pendant segment is called a pendant segment, such as v_2 and v_5 in Figure 1. segment is called a loop segment if it belongs to one of the segments which form a loop. In a Manhattan routing, at least four segments are needed to form a loop. For example, in Figure 1, segments v_3, h_4, v_4, and h_5 are loop segments. A path containing some of the n segments and connecting between every pair of m terminals without a loop is called a connected path. A connected path with minimal (distance) cost is called a shortest connectivity path. The vertices and segments in the shortest connectivity path are called pathing vertices and pathing segments, respectively, as shown in Figure 2(b).

General graphs

A TCP can be treated on a graph $G = (V, E)$, which consists of a set of vertices (m terminals and r dummy terminals) $V = \{v_1, v_2, \ldots, v_m, v_{m+1}, \ldots, v_{m+r}\}$, and a set of n edges (horizontal and vertical wire segments) $E = \{e_1, e_2, \ldots, e_n\}$. An edge e_k is identified with an unordered pair (v_i, v_j) of vertices. The vertices v_i, v_j associated with edge e_k are called end vertices of e_k. An edge having the same vertex as both its end vertices is called a self-loop edge. Also note that the definition allows more than one edge to be associated with any given pair of vertices; such edges are referred to as parallel edges. In this paper, it is assumed that a graph,

G, having self-loops and/or parallel edges is called a general graph. (In some graph-theory literature[15], a graph is defined to be only a simple graph that has neither self-loops nor parallel edges.)

The number of edges incident on a vertex (or dummy vertex) v_i, with self-loops counted twice, is called the degree of the vertex (or dummy vertex) and is denoted $\deg(v_i)$. For the TCP, the degree of the terminal vertex is at most 4 and the degree of the dummy vertex is no less than 3 and at most 8. For example, in Figure 2(a), there are six terminal vertices and ten dummy vertices, and the degrees of vertices are: $\deg(t_6) = 2$, $\deg(s_3) = 4$, and so on. If the degree of a dummy vertex is reduced to 2, then the dummy vertex is called a virtual vertex. A hyperedge consists of an alternating sequence of virtual vertices and edges which are connected in series. Naturally, a hyperedge has at least one edge. For example, in Figure 2(b), there are three dummy vertices, three virtual vertices, and eight hyperedges.

The TCP can be conceptualized as a problem of finding the shortest connectivity path that connects the m terminal vertices in a graph G without causing any loops. First, it must be ascertained that the graph G is a connected graph before the shortest connectivity path can be found. The graph G is considered a disconnected graph if one of following conditions holds:

- G is a null graph with no edges and with m isolated terminal vertices
- G has at least one isolated terminal vertex
- G consists of two or more components each of which is a subgraph of G and has at least one terminal vertex.

Otherwise, the graph G is a connected graph. To determine whether a graph G is connected or not, it should traverse the entire graph beginning from any terminal vertex. Sometimes this can be easily done without passing through all the vertices and the edges in the graph. For example, if after passing through some of the edges and vertices in the graph G, a connected subgraph consisting of all terminal vertices is readily obtained, then the graph G can be immediately determined to be a connected graph. Similarly, if after passing through some vertices and edges in the graph G, an isolated vertex is detected, the graph G is said to be a disconnected graph.

Hyper-complete graphs

It was shown above that to solve the TCP on a connected graph G with m terminal vertices, the shortest connectivity path has to be found from all the connected paths (or steiner trees) in the graph, allowing self-loops, parallel edges, dummy vertices, terminal vertices, and edges on different layers. A graph in which there exists an edge between each corresponding (out going) branch in every pair of terminal vertices is called a hyper-complete graph. Figure 3 shows three different hyper-complete graphs. Clearly, the hyper-complete graph is also a connected graph. That is, if the graph

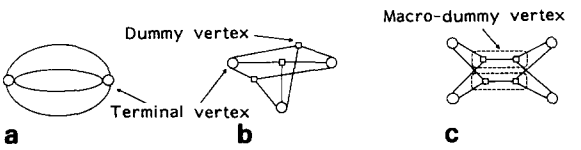


Figure 3. Three different hyper-complete graphs: (a) G_2 with $d_1 = 4$ and $d_2 = 0$; (b) G_3 with $d_1 = 3$ and $d_2 = 3$; (c) G_4 with $d_1 = 2$ and $d_2 = 3$

G is a hyper-complete graph, it will inherently contain many connected paths.

Without loss of generality, the TCP can be examined on a hyper-complete graph $G_m = (V, E)$, which consists of a set of m vertices and r dummy vertices, $V(G_m) = \{v_1, v_2, \ldots, v_m\} \cup \{v_{m+1}, v_{m+2}, \ldots, v_{m+r}\}$, and a set of n hyperedges, $E(G_m) = \{e_1, e_2, \ldots, e_n\}$. Let the degree of a terminal vertex, $\deg(v_t)$, be d_1 and the degree of a dummy vertex $\deg(v_d)$ be d_2, so that d_2 is at least 3 and no more than m. In the VLSI design, d_1 and d_2 are at most 4 and 8, respectively, where two adjacent layers are connected vertically by a via. Each of the d_1 branches in the m terminal vertices forms a hyper-connection. The hyper-complete graph G_m in turn has a set of d_1 hyper-connections. If a hyper-connection has more than one dummy vertex, it forms a macro-dummy vertex from these dummy vertices. Figure 3 shows three different hyper-complete graphs G_2, G_3 and G_4. For instance, G_2 has four hyper-connections and four hyperedges, G_3 has three hyper-connections, nine hyperedges and three dummy vertices, and G_4 has two hyper-connections, ten hyperedges and four dummy vertices (or two macro-dummy vertices).

Properties

After defining a hyper-complete graph G_m, some of its properties are introduced in the following lemmas.

Lemma 1

Given a hyper-complete graph $G_m = (V, E)$ with $m (m \geq 2)$ terminal vertices, let the degree of a terminal vertex $\deg(v_t)$ be $d_1 (d_1 \geq 1)$ and the degree of a dummy vertex $\deg(v_d)$ be $d_2 (d_2 \geq 3)$. Then, the total number of hyperedges $E(G_m)$, dummy vertices $D(G_m)$, and vertices $V(G_m)$ is

$$|E(G_m)| = \begin{cases} d_1 \left(m + \dfrac{m-3}{\lfloor d_2 - 2 \rfloor} \right) & \text{if } m = 2 \\ d_1 & \text{if } m \geq 3 \end{cases} \quad (1)$$

$$|D(G_m)| = \begin{cases} 0 & \text{if } m = 2 \\ d_1 \left(1 + \dfrac{m-3}{\lfloor d_2 - 2 \rfloor} \right) & \text{if } m \geq 3 \end{cases} \quad (2)$$

$$|V(G_m)| = m + |D(G_m)| \quad (3)$$

Proof

By the definition of a hyper-complete graph G_m with m $(m \geq 2)$ terminal vertices, G_2 consists of d_1 hyper-connections with no dummy vertex, i.e. these d_1 hyperedges directly connect one terminal vertex to another terminal vertex.

For $m \geq 3$, G_m also has a set of d_1 hyper-connections, and at least one dummy vertex exists in each hyper-

connection because the degree, d_2, of a dummy vertex is at least 3 but no more than m. First, assume that the number of terminal vertices, m, is equal to or larger than 3 but no more than d_2. Clearly, each hyper-connection in G_m has only one dummy vertex (the degree of which is just equal to m) and m hyperedges. In total, this G_m has d_1 dummy vertices and md_1 hyperedges, and the total number of vertices is $(m + d_1)$.

Second, consider that the number of terminal vertices m is larger than d_2. It is easy to see that each hyper-connection in this G_m must have at least one dummy vertex to form a macro-dummy vertex. However, it is important to count how many dummy vertices are added in a hyper-connection as the number of terminal vertices, m, increases. It will be seen that a hyper-connection in this G_m has an extra dummy vertex (and hyperedge) as long as the number of terminal vertices, m, is increased by a number ranging from 1 to *disp* ($disp = d_2 - 2$). Obviously, the total number of dummy vertices and hyperedges in G_m will be increased by a multiple of d_1 dummy vertices and hyperedges, respectively. Hence, we have the following inductive formulation.

$m = 2$	$\|E(G_m)\| = d_1$	$\|D(G_m)\| = 0$
$3 \leqslant m \leqslant d_2$	$\|E(G_m)\| = d_1 m$	$\|D(G_m)\| = d_1$
$d_2 + 1 \leqslant m \leqslant d_2 + 1(disp)$	$\|E(G_m)\| = d_1(m + 1)$	$\|D(G_m)\| = 2d_1$
$d_2 + 1(disp) + 1 \leqslant m \leqslant d_2 + 2(disp)$	$\|E(G_m)\| = d_1(m + 2)$	$\|D(G_m)\| = 3d_1$
...	...	...
...	...	...
...	...	...
$d_2 + (\lambda - 1)(disp) + 1 \leqslant m \leqslant d_2 + \lambda(disp)$	$\|E(G_m)\| = d_1(m + \lambda)$	$\|D(G_m)\| = d_1(1 + \lambda)$

where λ is $\lfloor (m - 3)/(disp) \rfloor$. The total number of vertices $V(G_m)$ is the sum of the terminals and the dummy vertices, i.e. $[m + d_1(1 + \lambda)]$. □

Figure 3(c) shows an example of a hyper-complete graph, G_4, with $m = 4$, $d_1 = 2$, and $d_2 = 3$. The number of hyperedges, dummy vertices and vertices can be calculated as follows. Since $disp = d_2 - 2 = 1$ and $\lambda = \lfloor (m - 3)/(disp) \rfloor = 1$, then

$$|E(G_4)| = d_1(m + \lambda) = 10$$

$$|D(G_4)| = d_1(1 + \lambda) = 4$$

$$|V(G_4)| = m + |D(G_4)| = 8$$

Lemma 2

Given a hyper-complete graph $G_m = (V, E)$ with m ($m \geqslant 2$) terminal vertices, the number of hyperedges, $C(G_m)$, in a connected path is:

$$|C(G_m)| = (m - 1) + \sum_{i=1}^{r} \pi_i$$

$$0 \leqslant r \leqslant \min(m - 1, d_1) \qquad (4)$$

where r is the number of required macro-dummy vertices; d_1 is the degree of a terminal vertex; and π_i is a positive integer, β, which represents the number of dummy vertices that form a macro-dummy vertex, if such a macro-dummy vertex, u_i, exists (otherwise, π_i is 0).

Proof

As shown in Lemma 1, a hyper-complete graph G_m with m ($m \geqslant 2$) terminal vertices has at least one connected path through these vertices, and at least one dummy vertex exists if the number of terminal vertices, m, is larger than 2.

If no dummy vertex is included in a connected path, i.e. $r = 0$, the number of edges in a connected path is exactly $m - 1$ hyperedges. (For example, G_2 has one hyperedge directly connecting the two terminal vertices without any dummy vertex.) Otherwise, a connected path in G_m ($m \geqslant 3$) has at least one dummy vertex, $r \geqslant 1$. In other words, a connected path is then equivalent to a steiner tree; the number of required extra r dummy vertices in the G_m is analogous to the number of required r steiner points in the steiner tree. Therefore, the total number of hyperedges in a connected path is the sum of $m - 1$ and r hyperedges.

The degree of any one of the r vertices is at least 2 if a dummy vertex is degraded to a virtual vertex. With the r dummy vertices, at most $r - 1$ terminal vertices and at least $2(r - 1)$ hyperedges can be connected to these dummy vertices without causing a loop. But these r dummy (virtual) vertices have at least $2r$ branches. By the definition of a connected path, no loop is included among the m terminal vertices. Figure 4 shows an aspect of the representation of a connected path. Hence, an inequality is obtained as follows.

$$2r - 2(r - 1) \leqslant m - (r - 1)$$

$$2 \leqslant m - r + 1$$

$$r \leqslant m - 1$$

A hyper-connection in a G_m must have at least one dummy vertex which forms a macro-dummy vertex, i.e. $\beta \geqslant 1$. If β is just 1, the macro-dummy vertex is equivalent to a dummy vertex and the degree d_2 of the dummy vertex is equal to the number of terminal vertices m, as discussed in Lemma 1. From Lemma 1, if d_2 ($d_2 \geqslant 3$) is smaller than the number of terminal

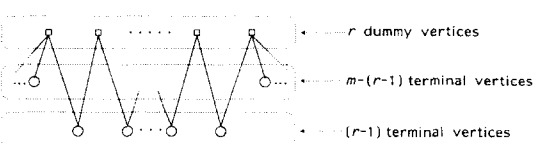


Figure 4. Appearance of representation of connected path

vertices m, the macro-dummy vertex in G_m has at least two dummy vertices. In other words, a hyper-connection in G_m has just one macro-dummy vertex connecting these dummy vertices. And the total number of hyperedges $E(G_m)$ will be increased by one while an extra dummy vertex is added to the hyper-connection. Hence, r is equal to the number of the macro-dummy vertex; π_i is β if the macro-dummy vertex u_i exists, other it is 0.

However, considering a G_m with degree d_1 less than m, the number of terminal vertices, this time the G_m has at most d_1 (not $m-1$) macro-dummy vertices. Therefore, the maximum value of number r is equal to the min $(m-1, d_1)$. □

By way of illustration, Figure 5(a) shows an example of the hyper-complete graph G_3, with $m=3$, $d_2=3$, and $\pi_i=\beta=1$, and Figure 5(b) shows all possible distinct connected paths among the three terminal vertices. These 21 connected paths are divided into two kinds of connected paths. One connected path contains only one dummy vertex ($r=1$) in addition to the m terminal vertices, and the total number of hyperedges is $C(G_3)=(m-1)+\pi_1=(3-1)+1=3$. The other connected path contains two virtual vertices ($r=2$) and four hyperedges: $C(G_3)=(m-1)+\pi_1+\pi_2=(3-1)+1+1=4$.

As shown in Figure 3(c), a hyper-connection in the G_4 ($m=4$, $d_1=2$, $d_2=3$) has two dummy vertices, i.e. $\pi_i=\beta=2$ and $1\leqslant r\leqslant \min(m-1, d_1)=2$. This G_4 has two types of connected paths, $r=1$ or $r=2$. One of the connected paths contains a macro-dummy vertex, the total number of hyperedges is $C(G_4)=(m-1)+\pi_1=(4-1)+2=5$. The other connected path contains two macro-dummy vertices and seven hyperedges: $C(G_4)=(m-1)+\pi_1+\pi_2=(4-1)+2+2=7$.

To develop the next lemma, let the number of r-permutations from n objects, denoted $P(n, r)$, be defined as:

$$P(n, r),\ n(n-1)(n-2)\ldots(n-r+1) \tag{5}$$

If $r=n=P(n, r)=n!$, if $r>n$, $P(n, r)=0$, and if $r=0$, $P(n, r)=1$. This definition can be applied to any natural numbers n and r. For example, $P(2, 0)=1$ and $P(2, 3)=0$.

Now let the number of r-combinations from n objects, denoted $C(n, r)$, be defined as:

$$C(n, r)=\frac{P(n, r)}{r!}=\frac{n(n-1)(n-2)\ldots(n-r+1)}{r!} \tag{6}$$

If $r=n$, $C(n, r)=1$, if $r>n$, $C(n, r)=0$, and if $r=0$, $C(n, r)=1$. This definition is true for any natural numbers n and r. For example, $C(2, 0)=1$ and $C(2, 3)=0$.

Lemma 3

Given a connected path with r macro-dummy vertices ($r\geqslant 0$) in a hyper-complete graph $G_m=(V, E)$ with m terminal vertices, if $0\leqslant r\leqslant 1$, the number of connected paths is:

$$CP_r=P(d_1, 1) \tag{7}$$

where d_1 is the degree of the terminal vertex. Otherwise, if $r\geqslant 2$, there are $r-1$ types of connected paths and the total number of connected paths to each type is

$$CP_r=\sum_{j=1}^{r-1} P(m, j)\eta(m, r, j)P(d_1, r) \tag{8}$$

where $\eta(m, r, j)$ is a type function.

Proof

In a hyper-complete graph G_m, with m terminal vertices and d_1 macro-dummy vertices, its connected path contains r macro-dummy vertices, $0\leqslant r\leqslant \min(m-1, d_1)$, as discussed in Lemma 2.

If $r=0$, the connected path does not have a macro-dummy vertex, i.e. the path connects directly to its two terminal vertices; this is the case for G_2. Hence, the total number of connected paths in G_2 depends on the degree d_1 of the terminal vertex, i.e. $CP_0=P(d_1, 1)$.

If $r=1$, the connected path has only one macro-dummy vertex. However, a macro-dummy vertex connects just m terminal vertices. Hence, the total number of connected paths depends on the degree, d_1, of the terminal vertex, i.e. $CP_1=P(d_1, 1)$.

Next, consider a connected path with r macro-dummy vertices in G_m, $r\geqslant 2$. No more than r terminal vertices can form a path connecting these macro-dummy vertices, otherwise it would generate one or more loops; thus, a connected path has 1, 2, ... j, ..., or $r-1$ terminal vertices to connect the r macro-dummy vertices, i.e. there are $r-1$ types of connected paths. Each type has j-combinations from m terminal vertices multiplied by $j!$, that is, $C(m, j)j!=P(m, j)$. And the remaining terminal vertices, $m-j$, are distributed into r macro-dummy vertices but the degree of each macro-dummy vertex is at least 2. The distribution is not uniform to each type. There are a number of different combinations which vary within each type; therefore, a type function $\eta(m, r, j)$ needs to be

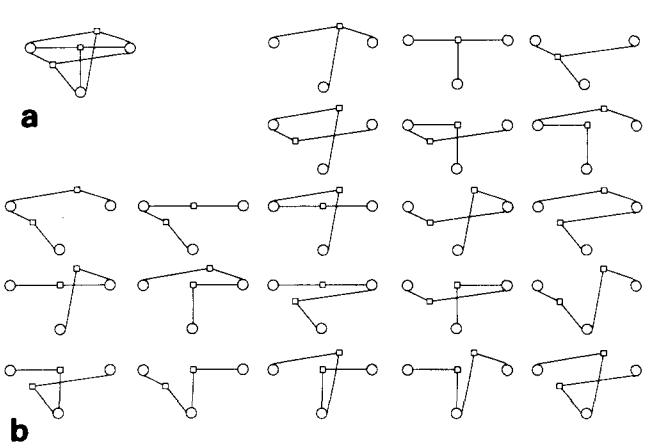


Figure 5. (a) Hyper-complete graph G_3 with three terminal vertices, $d_1=3$ and $d_2=3$; (b) all distinct connected paths of G_3

introduced. In addition, the r macro-dummy vertices have also r-permutations from the degree, d_1, of a terminal vertex, i.e. $P(d_1, r)$. Therefore, we have $r-1$ types of connected paths, and each type has $P(m, j)\,\eta(m, r, j)P(d_1, r)$ connected paths. □

The type function $\eta(m, r, j)$, just introduced, varies with the type of each connected path in G_m. Given a connected path with r macro-dummy vertices, $r \geqslant 2$, and j terminal vertices, $1 \leqslant j \leqslant r-1$, which are predetermined (see Figure 4), there exist at most $y_1 (y_1 = r + j - 1)$ hyperedges causing no loop and y_2 ($y_2 = 2r - y_1 = r - j + 1$) hyperedges, making the degree of each macro-dummy vertex equal to 2. Since some of the $(m-j)$ terminal vertices are matched with these y_2 hyperedges, and letting the rest of terminal vertices be z in number, we then obtain $z = (m-j) - y_2 = m - r - 1 \geqslant 0$. The type function $\eta(m, r, j)$ depends heavily on the value z.

If $z = 0$, a connected path contains just $m-1$ (since $z = m - r - 1 = 0$, $r = m - 1$) macro-dummy vertices. In other words, the m (i.e. $m - z = m - 0 = m$) terminal vertices are distributed into r macro-dummy vertices and the degree of each macro-dummy vertex is just 2. This is a unique match and the type function $\eta(m, r, j)$ is always equal to 1. For example, Figure 6 shows all types of connected paths with five macro-dummy vertices in G_6, each with type function 1. And in Figure 6(c), given $m = 6$, $r = 5$, and $j = 3$, then $y_1 = r + j - 1 = 5 + 3 - 1 = 7$, $y_2 = r - j + 1 = 5 - 3 + 1 = 3$, and $m - j = 6 - 3 = y_2$ or $z = m - r - 1 = 6 - 5 - 1 = 0$.

If $z > 0$, a connected path contains less than $m-1$ (since $z = m - r - 1 > 0$, $r < m - 1$) macro-dummy vertices. In other words, the $(m-z)$ terminal vertices are connected by r macro-dummy vertices such that the degree of each macro-dummy vertex is just 2, and the rest of z terminal vertices can be freely distributed into those r macro-dummy vertices. Experimentally, the distribution is not exactly uniform to each type and the type function $\eta(m, r, j)$ depends on different type combinations, but a general form is given as follows.

$$\eta(m, r, j) = \sum_{i=z+1,\, i=i-1}^{\lfloor (m-j)/2 \rfloor} \frac{(m-j)!}{i!\,i_1!\,i_2!\,\ldots\,i_{r-1}!} \qquad (9)$$

where $i, i_1, i_2, \ldots, i_{r-1}$ must satisfy the following condition: $i + i_1 + i_2 + \ldots + i_{r-1} = m - j$.

For example, Figure 7 shows two types of connected paths with two macro-dummy vertices in G_5, $m = 5$, $r = 2$, and $j = 1$. Then $z = m - r - 1 = 5 - 2 - 1 = 2$, and the type function is equal to the sum of $4!/3!1!$ and $4!/2!2!$.

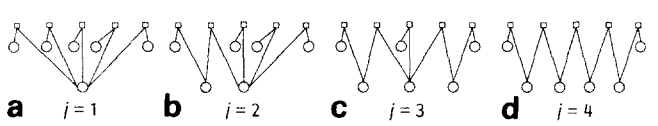


Figure 6. All types of connected paths with five macro-dummy vertices in G_6. Their type function is always 1

Figure 7. Two types of connected paths with two macro-dummy vertices in G_5

Figure 8 shows all types of connected paths with three macro-dummy vertices in G_5, with $m = 5$, $r = 3$, and $1 \leqslant j \leqslant 2$. If $j = 1$, then $z = m - r - 1 = 5 - 3 - 1 = 1$, the type function $\eta(m, r, j) = \eta(5, 3, 1) = 4!/2!1!1!$, and the number of connected paths is $CP_1 = P(5, 1)\,\eta(5, 3, 1)\,P(4, 3) = 1440$. If $j = 2$, then $z = m - r - 1 = 5 - 3 - 1 = 1$, type function $\eta(m, r, j) = \eta(5, 3, 2) = (3!/2!1!) + (3!/2!1!)$ (note, this is not a general form) and the number of connected paths is $CP_2 = P(5, 2)\,\eta(5, 3, 2)\,P(4, 3) = 2880$. Hence, the total number of connected paths is $CP_1 + CP_2 = 4320$.

A problem meanwhile arises. How many connected paths are present in a hyper-complete graph G_m with m terminal vertices? By Lemma 2, a connected path consists of $m-1$ hyperedges while a G_m has no macro-dummy (or dummy) vertex, like G_2. The number of connected paths $CP(G_2)$ in the G_2 is equal to the number of 1-permutations from the degree of a terminal vertex, d_1. Hence, $CP(G_2) = CP_0 = P(d_1, 1) = d_1$, as per Lemma 3. For example, in Figure 3(a), $CP(G_2) = P(4, 1) = 4$.

For the G_m with $m \geqslant 3$, by Lemma 3, we have $r-1$ types of connected paths and each type has a lot of different connected paths, where $1 \leqslant r \leqslant \min(m-1, d_1)$, as discussed in Lemma 1. For example, in Figure 3(b), G_3 has two types of connected paths, containing one or two dummy vertices. Figure 9 shows the different representations of the two types of connected paths. For the first type of connected path, $r = 1$, the number of distinct connected paths is $CP_1 = P(d_1, r) = P(3, 1) = 3$. For the second type of connected path, $r = 2$, since $z = m - r - 1 = 0$, the type function is equal to 1 and the number of distinct connected paths is $CP_2 = P(m, 1)\,P(d_1, r) = P(3, 1)\,P(3, 2) = 18$. Therefore, G_3 has 21 different connected paths, as shown in Figure 5(b).

Theorem 1

The total number of connected paths in a hyper-complete graph, $G_m = (V, E)$, with m terminal vertices, is

$$CP(G_m) = P(d_1, 1)\big|_{r=0,1} + \sum_{r=2}^{\min(m-1,\, d_1)} \left\{ \sum_{j=1}^{r-1} P(m, j)\eta(m, r, j)P(d_1, r) \right\} \qquad (10)$$

where d_1, r and $\eta(m, r, j)$ are previously defined.

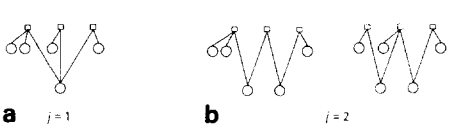


Figure 8. All types of connected paths with three macro-dummy vertices in G_5

Figure 9. Two different connected paths of Figure 3(b)

Proof

From Lemma 2, a connected path has at most $\min(m-1, d_1)$ macro-dummy vertices in a hyper-complete graph G_m with m terminal vertices with degree d_1.

By the definition of a hyper-complete graph G_m, the number of terminal vertices must be at least 2, $m \geqslant 2$. Also, by Lemma 3, if a G_m has just two terminal vertices, $m = 2$, then its connected path will contain no macro-dummy vertex, $r = 0$, and the total number of connected paths will be $CP(G_2) = CP_0 = P(d_1, 1)$. If a G_m has at least three terminal vertices, $m \geqslant 3$, its connected path will contain a different number of r macro-dummy vertices, $1 \leqslant r \leqslant \min(m-1, d_1)$, and the total number of connected paths will be $CP(G_m) = CP_1 + CP_2 + \ldots + CP_{\min(m-1, d_1)}$.

NP-hardness of TCP

Next, we want to find the shortest connectivity path in the hyper-complete graph G_m. This path must contain each terminal vertex exactly once and its length (or distance cost) will be less than other connected paths if they exist. Theoretically, the problem of finding the shortest connectivity path can always be solved by enumerating all the connected paths $CP(G_m)$ (see Theorem 1), by calculating the distance cost of each path traversal, and then by picking the shortest one. However, for a large value of m and d_1, the work involved is too great even for a digital computer. The problem is to prescribe a manageable algorithm for finding the shortest route. No efficient algorithm for problems of arbitrary size has yet been found, although many attempts have been made. Hence, a heuristic method must be applied to solve it, which gives a route very close to the shortest one, but does not guarantee the route to be the shortest.

The problem of finding the shortest connectivity path in the hyper-complete graph G_m is analogous to the finding of a minimal steiner tree, which has been proved NP-complete[13].

Theorem 2

The problem of finding the shortest connectivity path in a hyper-complete graph G_m with m terminal vertices is NP-hard. Therefore, the TCP is also NP-hard. (The proof of this theorem is given in Appendix)

In the next section, we propose a heuristic algorithm to solve the TCP to find the shortest connectivity path in linear time with some proprocessing work.

TERMINAL CONNECTIVITY PROBLEM ALGORITHM

TCP algorithm overview

In order to solve the TCP it is necessary to begin by determining the connectivity of the terminals and then to find the shortest connectivity path if the TCP is proved to be connected. In order to achieve this and to implement the algorithm efficiently, a powerful structure – corner-stitching – is applied, for which a lot of efficient supporting algorithms have been proposed[14].

First, it is necessary to map the m terminals and n wire segments into the multi-layer (corner-stitching) planes according to the given layer information. Then, a breadth-first expansion is begun from one of the terminals in the multi-layer planes. If any one of the m terminals remains unvisited, there is no connected path through the m terminals in the TCP. Otherwise, the TCP has one or more connected paths through the m terminals, one of which is the shortest path. To simplify the process of finding the shortest connectivity path, a maximal loop is obtained by eliminating some redundant segments and by skipping some unique segments if they exist before the shortest connectivity path is found. Therefore, a part of the shortest connectivity path can be obtained from the maximal loop. By combining this part of the shortest connectivity path and those just skipped and unique segments, the whole shortest path connecting the m terminals is finally generated.

Figure 10 presents an overview of the TCP. First, given m terminals and n wire segments with their layering information, construct the layout in a multi-layer plane model. Then, traverse the terminals and wire segments in the multi-layer planes and determine whether or not the m separate terminals are connected to each other by those wire segments. If so, first remove those redundant wire segments (such as isolated and pendant segments) to find the maximal loop, if any; then obtain the shortest connectivity path of the TCP from the maximal loop. Otherwise, highlight the disconnected occurrences or try to repair them by adding or stretching a segment without violating electrical or

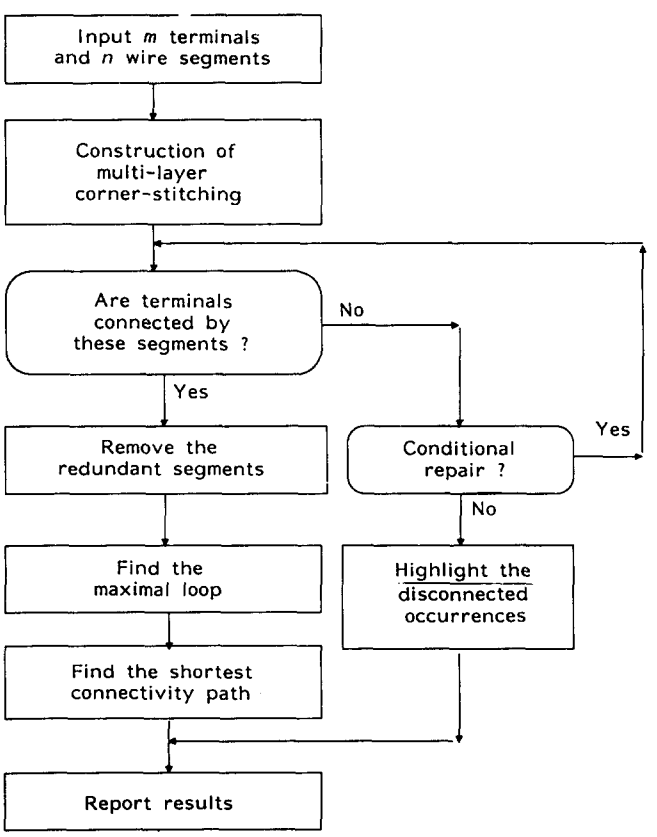


Figure 10. Overview of terminal connectivity problem algorithm

design rules. For example, in Figure 1, the vertical wire segment v_5 can be stretched up to reach the isolated terminal t_7 such that this TCP become connected. If the disconnected occurrence(s) can be repaired, then the connectivity checking is continued. Otherwise, all of the disconnected information must be reported.

TCP heuristic algorithm

The heuristic algorithm of the TCP is divided into six steps. Each step is discussed in detail, with the help of an example, shown in Figure 11, and its time complexity is also calculated.

Step 0: Constructing multi-layer planes

The construction of the multi-layer planes is just a mapping of the m terminals and n wire segments on the original flat layout on to different planes. Overlapping tiles are not allowed in the same plane. A segment should be split if it is intersected by other segments. To construct the multi-layer planes efficiently, first map the m terminals and horizontal segments on to the multi-layer planes without considering overlapping, then map the remaining segments (vertical segments). If the vertical segments intersect the horizontal segments in the same layer plane, splitting operations will be required. Thus, the multi-layer planes of m terminals and n wire segments can be constructed. Figure 11(a) depicts the construction of a two-layer corner-stitching for the inputs of six terminals and 16 wire segments.

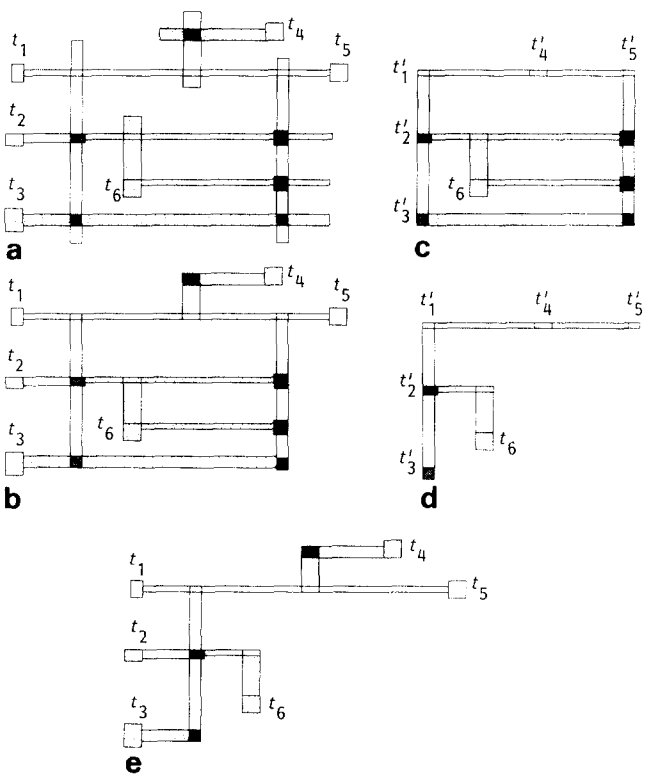


Figure 11. TCP process with six terminals and 16 wire segments: (a) construction of multi-layer corner-stitching; (b) after removal of all pendant segments; (c) maximal loop consisting of six dummy terminals; (d) shortest path obtaining from the maximal loop; (e) results of the shortest connectivity path

Since the solution of the TCP is based on the construction of multi-layer planes, this step is said to be a preprocess of the TCP. By using a corner-stitching data structure, all preprocessing operations are almost localized and with linear run time. In particular, the expected time of a corner-stitching tile insertion or deletion is constant. The same configuration requires a memory space about three times as great as the degree of time complexity. In total, this construction takes $O(m + (1 + c)n)$ time complexity and requires three times as much memory space, where c is a positive constant that depends on the intersections between n wire segments.

Step 1: Determining terminal connectivity

After the construction of the multi-layer planes, the next step is to recognize whether or not at least one connected path between m separate terminals exists in the multi-layer planes. For this purpose, a breadth-first searching method is applied by starting from any terminal and passing the incident segments to reach its adjacent vertices. This process is continued until all the m terminals are visited. If an isolated terminal, a pendant vertex or segment, is found, the occurrence (which is disconnected from the connected paths) should, if possible, be repaired by adding or stretching a segment without violating any design rule, if no connected path is found, report or highlight the corresponding disconnected occurrences should be reported or highlighted and the TCP stopped. Otherwise, there exists at least one path connecting the m terminals. For example, the path shown in Figure 11(a) is connected. To simplify the process of finding the shortest connected path, it is necessary to remove any redundant segments, such as isolated or pendant segments. A pendant_list is used to record these redundant segments during the expansion.

Note that in the above method, all vertices and segments are marked exactly once, except for those isolated segments which are irrelevant to the determination of terminal connectivity. Therefore, the algorithm has $O(m + (1 + c)n)$ time complexity.

Step 2: Eliminating redundant segments

Having established that the m separate terminals are connected together through some segments, this implies that at least one shortest connectivity path exists through the m terminals and that some segments are possibly redundant. Isolated, pendant, and loop segments are examples of redundant segments which are useless for connecting the m terminals and need to be removed. To illustrate, Figure 11(b) shows the appearance after the removal of all the pendant wire segments from Figure 11(a).

Since all the pendant segments have been recorded in the pendant_list in Step 1, one can easily eliminate every alternating sequence of vertices and segments, whose starting segment is in the pendant_list. Further, the isolated segments will vanish naturally when the multi-layer planes are converted back to the original layout with m terminals and n' segments ($n \leqslant n$). Which loop segments need to be removed in finding

the shortest connectivity path will be discussed in Step 4.

From the above discussion, some alternating sequences of vertices (vias or bends) and segments are eliminated in the multi-layer planes. Let n_p denote the number of these vertices and segments. Hence, the algorithm will take $O(n_p)$ time. In general, n_p is smaller than n, the total number of segments.

Step 3: Searching for maximal loop

After the elimination of the pendant segments, it is time to find the shortest connectivity path among the m terminals. Sometimes there are many unique paths or multiple loops between every pair of terminals in the multi-layer planes. But it must be established whether any loop exists in the multi-layer planes before the shortest connectivity path through the m terminals can immediately be obtained from the multi-layer planes. Otherwise, there exist some loop segments among the m terminals. Accordingly, first it is necessary to pick out the m dummy vertices forming the maximal loop, and then to find the shortest connectivity path within the maximal loop. A dummy terminal can be detected by searching from each terminal to a vertex which has more than one incident segment, excluding the segment that is just going in. At the same time, a distance cost from every terminal to its corresponding dummy terminal is to be recorded in this dummy terminal. These dummy terminals and their cost, which are used to find the shortest connectivity path in the maximal loop, will be discussed in next step. Figure 11(c) shows the maximal loop composed of six dummy terminals of Figure 11(b).

In this step, there are some alternating sequences of vertices and segments from every terminal to the corresponding dummy terminals. These sequences are unique and from part of the shortest connectivity path among the m terminals. Let n_m be the number of these pathing vertices and segments. Thus, the algorithm takes $O(n_m)$ time. In general, n_m is smaller than the total segments n.

Step 4: Finding shortest connectivity paths

Step 3 generates part of the shortest connectivity path through the m terminals (named path_1) and a set of m dummy terminals (recorded in the vertex_list) which form the maximal loop. If the vertex_list is empty, the path_1 found in Step 3 is just the shortest connectivity path that connects the m separate terminals. Otherwise, it is necessary to find the shortest connectivity path (named path_2) within the maximal loop formed by the m dummy terminals by eliminating some loop segments. In this case, the two parts of the shortest connectivity paths (i.e. path_1 and path_2) must be combined to obtain the whole shortest connectivity path through the m terminals. Figure 11(d) shows the shortest path obtained from the maximal loop in Figure 11(c).

In this step, each vertex and segment in the maximal loop is visited at the stages of expanding and backtracking. Let n_w be the total number of vertices and segments within the maximal loop and x a positive constant that is the number of times a vertex or segment is visited. Therefore, the algorithm has $O(xn_w)$ time complexity.

Step 5: Reporting the TCP results

Finally, what remain in the multi-layer planes are the shortest connectivity path with its connecting m terminals and the other negligible isolated segments. First, to refine the shortest connectivity path in the multi-layer planes, it is necessary to eliminate any redundant bends or vias (constructed in Step 0) as shown in Figure 11(e). Then, the shortest connectivity path is converted back to the original layout with m terminals and n' ($n' \leqslant n$) segments.

In summary, the TCP algorithm has $O(m + (1 + c)n)$ linear time complexity, as established in the above six steps. Obviously, this linear time algorithm is achieved with the assistance of the powerful data structure – corner-stitching.

EXPERIMENTAL RESULTS

The TCP heuristic algorithm was implemented on a SUN III/160 workstation using standard C and the Berkeley 4.2 UNIX operating system. Table 1 shows the experimental results, as well as some significant data, such as the number of layers (layer), the number of terminals (term), the number of original segments (seg), the length of original segments (len), the number of transferred segments (tseg), and the length of the shortest connectivity path (tlen).

A circuit with n solid tiles represented with a data structure – corner-stitching – never requires more than $3n + 1$ space tiles even in the worst case. However, in actual circuit layouts, the number of space tiles required is close to n. Hence, the TCP algorithm takes $O(3(m + (1 + c)n))$ memory space in the worst case, and its time complexity is $O(m + (1 + c)n)$.

Figure 12 shows the curve of the sum of terminals and wire segments to be related to the running time. The running time is linearly related to the sum of given terminals and wire segments. Note that the running time is the sum of the preprocessing time (pre_time) and the time taken to find the shortest connectivity path (tcp_time).

Figure 13(a) shows an example of tcp5 with 19 terminals and 104 wire segments, and Figure 13(b) gives the shortest connectivity path.

CONCLUSION

We have shown that the new problem of terminal connectivity (TCP) in the VLSI or PCB multi-layer layout

Table 1. Experimental data and results

	layer	term	seg	len	tseg	tlen
tcp1	1	2	95	5236	6	852
tcp2	1	5	18	800	9	288
tcp3	2	6	30	493	23	356
tcp4	2	6	41	1274	17	568
tcp5	2	19	104	2978	57	1508
tcp6	3	38	199	5260	125	3180
tcp7	3	60	382	8716	220	5270
tcp8	4	74	625	12628	268	4960

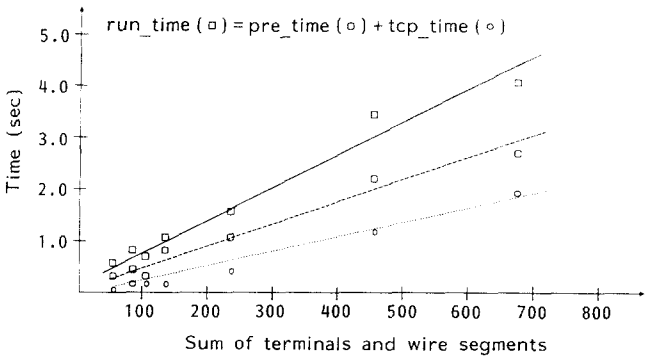
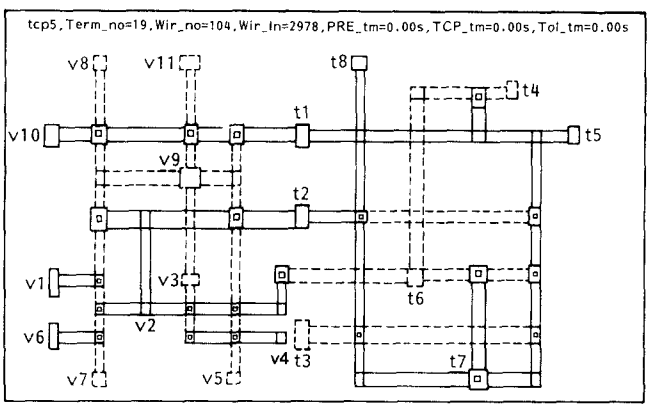
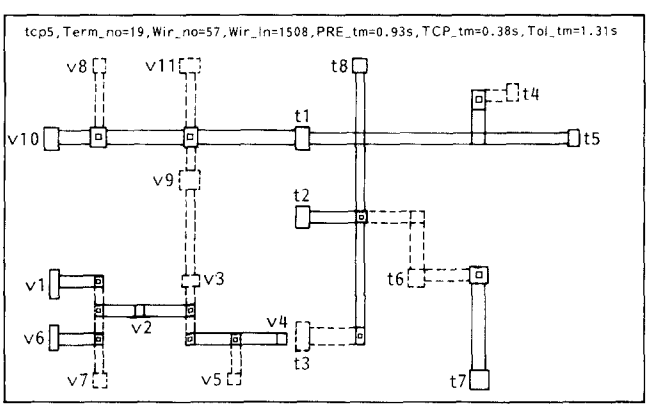


Figure 12. Graph of running time against sum of terminals and wire segments



a



b

Figure 13. (a) Example of tcp5 with 19 terminals and 104 wire segments; (b) shortest connectivity path

scheme is NP-hard, because finding the shortest connectivity path in the TCP can be viewed as a minimal steiner tree problem (which has been proved NP-complete). We have proposed a heuristic algorithm based on multi-layer corner-stitching to solve the TCP efficiently in $O(m + (1 + c)n)$ linear time. This algorithm may also be applied for the checking of electrical or design rules.

ACKNOWLEDGEMENT

This work was supported by the National Science Council, Taipei, Taiwan, under Grant NSC 78-0404-E002-46 and Grant NSC 79-0404-E002-33.

REFERENCES

1 **Tsai, C C, Feng, W S and Chen, S J *et al.*** 'Generalized terminal connectivity problem for multi-layer layout scheme' *Joint Tech. Conf. on Circuits System Computers and Communications* (June 1989) pp 173–178

2 **Brown, A D** 'Automated placement and routing' *Comput.-Aided Des.* Vol 20 No 1 (1988) pp 39–44

3 **Mueller, H and Mlynski, D A** 'Automatic multilayer routing for surface mounted technology' *International Symp. on Circuits and System* (May 1988) pp 2189–2192

4 **Soukup, J** 'Circuit layout' *Proc. IEEE* Vol 69 No 10 (1981) pp 1281–1304

5 **Scott, W S and Ousterhout, J K** 'Plowing: Interactive stretching and compaction in MAGIC' *Proc. 21st Des. Automation Conf.* (June 1984) pp 166–172

6 **Chow, Y E** 'A subjective review of compaction' *Proc. 22nd Des. Automation Conf.* (June 1985) pp 396–404

7 **Xiong, X M and Kuh, E S** 'The constrained via minimization problem for PCB and VLSI design' *Proc. 25th Des. Automation Conf.* (June 1988) pp 573–578

8 **Losleben, P and Thompson, K** 'Topological analysis for VLSI circuits' *Proc. 16th Des. Automation Conf.* (June 1979) pp 461–473

9 **Takashima, M *et al.*** 'Programs for verifying circuit connectivity of MOS/LSI mask artwork' *Proc. 19th Des. Automation Conf.* (June 1982) pp 544–550

10 **Lipski, W** 'Finding a Manhattan path and related problems' *Networks* Vol 13 (1983) pp 399–409

11 **Lipski, W** 'An $O(n \log n)$ Manhattan path algorithm' *Inform. Process. Lett.* Vol 19 (August 1984) pp 99–102

12 **Asano, T** 'Generalized Manhattan path algorithm with applications' *IEEE Trans. on CAD* Vol 7 No 7 (1988) pp 797–804

13 **Garey, M R and Johanson, D S** *Computers and intractability: A guide to the theory of NP-completeness* W. H. Freeman, USA (1979)

14 **Ousterhout, J K** 'Corner stitching: A data-structuring technique for VLSI layout tools' *IEEE Trans. on CAD* Vol 3 No 1 (1984) pp 87–100

15 **Deo, N** *Graph theory with applications to engineering and computer science* Prentice Hall, USA (1974)

APPENDIX 1: PROOFS

Proof of Theorem 2

Here, an attempt is made to establish the NP-hardness of the process of finding the shortest connectivity path in hyper-complete graphs (SCPHCG) G_m, which implies the NP-hardness of the terminal connectivity problem (TCP). This is shown by reducing the known NP-complete 'steiner tree in graphs' (STG) problem to the problem of finding the shortest connectivity path.

Steiner Tree in Graphs[13]

Instance: A graph $G = (V', E')$, a weight $w(e') \in \mathbb{Z}$ ($\mathbb{Z}$ denotes the set of positive integers) for each $e' \in E'$, a subset $R' \subseteq V'$, and a positive integer bound B'.

Question: Is there a subtree of G that includes all vertices of R' such that the sum of the weights of the edges in the subtree is no more than B'?

Shortest Connectivity Path in Hyper-Complete Graphs

Instance: A hyper-complete graph $G_m = (V, E)$ with m terminal vertices, a weight $w(e) \in \mathbb{Z}$ for each $e \in E$, a subset $R \subseteq V$, and a positive integer bound B.

Question: Is there a subtree of G_m that includes all vertices of R but must contain m terminal vertices, such that the sum of the weights of the edges in the subtree is no more than B?

It is easy to see that finding the shortest connectivity path in a hyper-complete graph G_m is NP, since a nondeterministic algorithm is needed only to guess a subset of vertices and hyperedges with distance cost no more than B and to check that it is legal.

STG will be transformed in to SCPHCG. Let an arbitrary instance of STG be given by the graph $G = (V', E')$ and the positive integer B'. A hyper-complete graph $G_m = (V, E)$ must be constructed such that G_m has a shortest connectivity path if and only if G has a steiner tree with a sum of edge weights less than or equal to B'.

Once more the construction can be viewed in terms of matching in a bipartite graph by assuming that the vertex set V' in STG can be decomposed into two disjoint subsets V_1' and V_2' such that each of the edges in the STG graph G joins a vertex in V_1' with a vertex in V_2'. Let the subset V_1' be the number of m terminal vertices, $V_1' = \{v_1, v_2, \ldots, v_m\}$, and the subset V_2' be the sum of steiner and dummy-steiner vertices, $V_2' = \{v_1', v_2', \ldots, v_s'\} \cup \{v_1'', v_2'', \ldots, v_t''\}$. In general, for a steiner tree, the degree of a terminal vertex is at least 1; the degree of a steiner vertex is at least 3 and the degree of a dummy-steiner vertex is always 2. For convenience, we use a relaxed-bipartite graph to represent an instance of steiner tree, that is, an edge may exist between a steiner vertex and dummy-steiner vertex in the subset V_2'. First, the graph G_m has m terminal vertices from the subset V_1', which will be used to select r steiner or dummy-steiner vertices from the subset V_2'. Second, for each edge in E', G_m contains a number of hyperedges that will be used to ensure that any loop does not exist among the $m + r$ vertices. One instance of matching in the relaxed-bipartite graph is shown in Figure 14(a). It has four terminal vertices, $V_1' = \{v_1, v_2, v_3, v_4\}$, one steiner vertex and two dummy-steiner vertices, $V_2' = \{v_1'\} \cup \{v_1'', v_2''\}$, and six edges, $E' = \{e_1, e_2, \ldots, e_6\}$, as shown in Figure 14(b).

By the definition of the hyper-complete graph G_m, Figure 14(b) can be transformed into an instance of G_4 with four terminal vertices, $V_1 = \{v_1, v_2, v_3, v_4\}$, one dummy vertex v_d (merging of v_1' and v_2''), one virtual vertex v_1'', one macro-dummy vertex $V_2 = \{v_d\} \cup \{v_1''\}$, and five hyperedges, $E = \{e_1, e_2, \ldots, e_{5,6}\}$, as shown in Figure 14(c).

In the completed construction, the steiner tree will be replaced by the matching of relaxed-bipartite graph

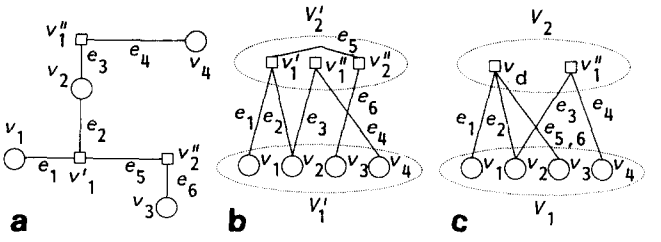


Figure 14. Instance of minimal steiner tree is used in transforming to finding of shortest connectivity path in hyper-complete graph: (a) minimal steiner tree with four terminals and six edges; (b) relaxed-bipartite graph of (a); (c) bipartite graph of shortest connectivity path is transformed from (b)

$G = (V', E')$, where $V_1' = \{v_1, v_2, \ldots, v_m\}$, $V_2' = \{v_1', v_2', \ldots, v_s'\} \cup \{v_1'', v_2'', \ldots, v_t''\}$, the subset of vertices $R' \subseteq V'$ is $R' = \{V_1' \cup V_2'\}$, the subset of edges $F' \subseteq E'$ is $F' = \{e_1', e_2', \ldots, e_n'\}$, and the total weight of the steiner tree $\sum_{i=1}^{f'} w(e_i')$ is less than the positive integer bound B'.

The construction of a hyper-complete graph $G_m = (V, E)$ from the graph G is described below. The interconnected vertices in the subset vertices V_2' are merged into one dummy vertex and their edges are also merged into one hyperedge; that is, $V_1 = \{v_1, v_2, \ldots, v_m\}$, $V_2 = \{u_1', u_2', \ldots, u_x'\} \cup \{u_1'', u_2'', \ldots, u_y''\}$, the subset of vertices $R \subseteq V$ is $R = \{V_1 \cup V_2\}$, the subset of hyperedges $F \subseteq E$ is $F = \{e_1, e_2, \ldots, e_p\}$, and the total weight of the connected path (see Theorem 1), $\sum_{j=1}^{f} w(e_j)$, is less than the positive integer bound B. It is easy to see that G_m can be constructed from G in polynomial time.

We claimed above that G_m has a shortest connectivity path if and only if G has a steiner tree with a sum of edge weights less than or equal to B. Suppose a connected path with m terminal vertices, r macro-dummy vertices, and $m - 1 + r$ hyperedges (see Lemma 2) is the shortest connectivity path of G_m. The r-macro-dummy vertices can be partitioned into two subsets of vertices, i.e. the subset of dummy vertices r_1 and the subset of virtual vertices r_2. These dummy and virtual vertices are equivalent to the steiner and dummy-steiner vertices in a steiner tree, respectively. Thus, those $m - 1 + r$ hyperedges are also decomposed into a number of edges after the r macro-dummy vertices are partitioned. Therefore, the shortest connectivity path for G_m is completely equivalent to the minimal steiner tree for G.

Conversely, given a minimal steiner tree in G, its steiner vertices, dummy-steiner vertices, and edges can be replaced by the dummy vertices, the virtual vertices, and the hyperedges in G_m, respectively. The degree of a dummy vertex is at least 3 but no more than m, the degree of a virtual vertex is just 2, and the hyperedge consists of a sequence of dummy or virtual vertices and edges. It is easy to see that a minimal steiner tree in G is also equivalent to a shortest connectivity path in G_m. □