A Multi-terminal Net Routing Algorithm for IC Design

Somchai Prasitjutrakul

Department of Computer Engineering, Chulalongkorn University
Phayatai Rd. Bangkok 10330, THAILAND

ABSTRACT

An algorithm for the global routing of a multiterminal net is presented. The algorithm based on the A*-search technique considers *all* the unconnected terminals *simultaneously* during the search for the minimum-cost path. The algorithm does not choose any specific terminal as the target to be connected, but rather allows the minimum-cost path to proceed in the minimum-cost direction and it eventually hits the appropriate target terminal. The cost of a path is the estimated total net length of the minimum Steiner tree connecting all the terminals. The experimental results show that a larger number of shorter global routes is achieved by the algorithm in comparison to the results obtained from previous approaches.

1. INTRODUCTION

In custom chip design, global routing is a process of determining the set of channels (which are the routing areas between modules) through which each signal net will pass. One problem in global routing is to determine the shortest route for connecting a given signal net. The shortest global route for a two-terminal net can be determined by using the Lee-Moore algorithm [1], the Dijkstra algorithm [2], or the A*-search algorithm [3]. For a multiterminal net, the global routing problem can be transformed to the problem of finding the minimum rectilinear Steiner tree on a routing graph, which is an NP-complete problem [4].

For the multiterminal-net global routing problem, most of the previous algorithms connect pre-selected vertices one at a time to the previously constructed connection tree (by using the shortest path algorithm) [3,5,6,7,8]. The main drawback of this approach is caused by the fact that the next target vertex to be connected is *pre-selected* without any knowledge of the structure of the routing graph. (The closest vertex to the connection tree of the already connected vertices is usually selected as the target vertex based on Prim's minimum spanning tree algorithm [9].) Moreover, the shortest path from the connection tree to the selected target vertex does not necessarily yield the minimum total length when all the terminals are connected. This is because the shortest path found may not be shared by other later paths for the unconnected terminals. This is because the unconnected terminals are not considered during the shortest path search process.

This paper presents an algorithm for finding a global route for a multiterminal net minimizing the total length of the global route. It can be used as part of a custom IC chip design system. The algorithm is a simple, modified version of the A*-search global routing algorithm [3]. Terminals are connected one at a time as usual, but the algorithm does not commit itself to connect any specific terminal. Rather, it considers *all* the unconnected terminals *simultaneously* during the search by estimating the lower-bound total length of the Steiner tree connecting all the unconnected terminals, and using this length to guide the path search process. The rest of the paper is organized as follows. In Section 2, we briefly describe the routing graph used in the routing process. The global routing algorithm is presented in detail in Section 3. Section 4 presents the experimental results for the algorithm. The conclusion is given in Section 5.

2. ROUTING GRAPH

In custom chip design methodology, a signal net is allowed to pass through channels, which are routing areas between the modules. A commonly used routing graph, which is constructed from the channels and their intersections [6,10], is used during the global routing. Each edge of the routing graph corresponds to a subchannel and each vertex represents the intersections between channels (see Figure 1). Before routing a signal net, a set of additional vertices (which represent the signal terminals) are added to the routing graph. The extra vertex (which will be called a terminal vertex) representing a signal terminal is added to the nearest routing edge (Figure 1). These extra vertices are deleted after a global route of the net is determined. By using this routing graph, determining a global route for a net becomes equivalent to finding a Steiner tree connecting all the terminal vertices on the routing graph.

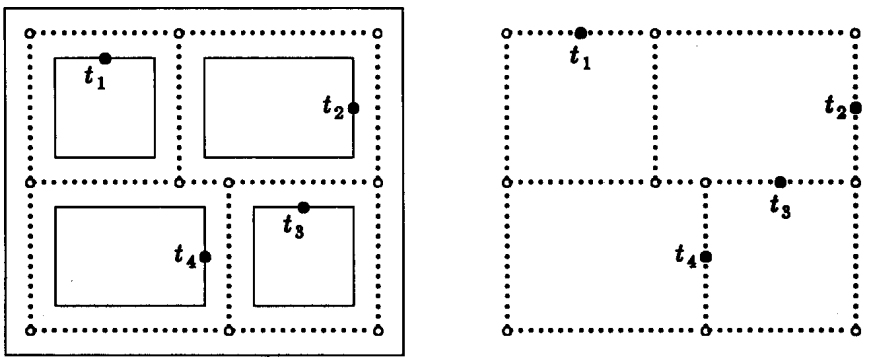

Figure 1. Routing graph with additional terminal vertices of a signal.

3. THE ALGORITHM

Given a routing graph $G = (V,E)$ where V is a set of vertices and E is a set of edges, along with a set of terminal vertices $T = \{t_1,t_2,...,t_n\}$, $T \subset V$, the global routing algorithm determines a Steiner tree on graph G having all the terminal vertices, which minimizes the total edge length of the tree. The algorithm begins by selecting a single terminal vertex to be considered as a source-vertex group and then searches for the minimum-cost path. Each time the minimum-cost path reaches a new terminal vertex, that new terminal vertex is added to the source-vertex group which contains all the vertices of the existing minimum-cost path and the algorithm proceeds to find the new minimum-cost path. This process continues until all the terminal vertices are included in the source-vertex group.

The cost of a path is taken to be the estimated lower bound length of the minimum Steiner tree connecting all the unconnected terminal vertices. Given :

1. a connection tree, V_{src}, which is the source-vertex group,
2. the minimum-cost path $P(v)$ having total length of $L_p(v)$, starting from a vertex in V_{src} and ending at a vertex v (we call v the front-end vertex of path $P(v)$), and
3. a set of unconnected terminal vertices T_u,

the lower bound length of interconnections, L^*, needed to connect from V_{src} to all the vertices in T_u using path $P(v)$, can be determined as follows.

$$L^*(v) = L_p(v) + \frac{2}{3} \sum_{\forall t_i, t_i \in T_u} L_{min}(t_i)$$

where $L_{min}(t_i)$ is the length of the edge connecting terminal vertex t_i of the minimum spanning tree obtained from the following procedure (see Figure 2).

GetMSTEdgeLength(V_{src}, v, T_u) {
    $V'_{src} = V_{src} \cup P(v)$; $T'_u = T_u$
    do {
        (t_i, w) = **GetTwoClosestVertices(** T'_u , V'_{src})
            where $t_i \in T'_u$ and $w \in V'_{src}$
        $L_{min}(t_i)$ = **RectDist(** t_i , w)
        $V'_{src} = V'_{src} \cup \{ t_i \}$
        $T'_u = T'_u - \{ t_i \}$
    } while ($T'_u \neq \varnothing$)
    return(L_{min}())
}

Hwang [11] showed that the ratio of the length of a minimum rectilinear Steiner tree to a minimum rectilinear spanning tree is no greater than 2/3. Therefore, we first determine the minimum rectilinear spanning tree used for connecting $V_{src} \cup P(v)$ and T_u. Then the lower bound length of interconnections needs to connect from V_{src} to T_u using the path $P(v)$ is the length of the path $P(v)$, which is $L_p(v)$, plus 2/3 of the total length of all of the edges of the minimum rectilinear spanning tree. (Note that for the function

RectDist(*x*,*y*), if *x* and *y* are adjacent, it equals to the length of the edge connecting *x* and *y*, otherwise, it is a lower bound length of the path from *x* to *y*.)

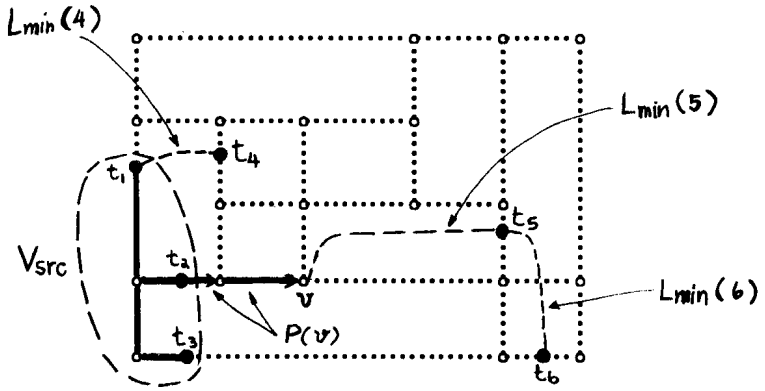

Figure 2. Total net length estimation.

Rather than recomputing $L_{min}(t_i)$ by reconstructing the minimum rectilinear spanning tree each time L^* is needed, $L_{min}(t_i)$ can be incrementally updated in order to reduce the computation time. Let $L'_{min}(v,t_i)$ be the $L_{min}(t_i)$ of a path ending at vertex v. $L'_{min}(v,t_i)$ and $L_p(v)$ can be determined from $L'_{min}(w,t_i)$ and $L_p(w)$ where vertex w is the parent vertex of v on the path as shown below. (**RectDist**(*w*,*v*) is the rectilinear distance between *w* and *v*).

$$L^*(v) = L_p(v) + \frac{2}{3} \sum_{\forall t_i, t_i \in T_u} L'_{min}(v, t_i)$$

where $L_p(v) = L_p(w) +$ **RectDist**(w, v)

$L'_{min}(v, t_i) =$ Min{ **RectDist**(v, t_i) , $L'_{min}(w, t_i)$ }

Notice that when there is a path $P(v)$ whose front-end vertex v does not change any L'_{min} of parent vertex of v (in other words, vertex v dose not change the structure of the previous minimum spanning tree), $L^*(v) = L^*(w) +$ **RectDist**(w,v). The additional value of **RectDist**(w,v) provides no information about the unconnected terminal vertices which can be used to guide the searching process. This can causes the search to wander to some unnecessary vertices and increases the search time. To avoid this situation, we add a value $\Delta_{min}(v)$ (shown below) which depends on how far away the vertex v is from the unconnected terminal vertices when the vertex v does not change the previous minimum spanning tree (the new L^* is denoted as L^{**}).

$$L^{**}(v) = \begin{cases} L_p(v) + \frac{2}{3}\left[\Delta_{min}(v) + \sum_{\forall t_i, t_i \in T_u} L'_{min}(v, t_i) \right] & \text{if } |T_u| > 1 \\ L_p(v) + L'_{min}(v, t_i) + \Delta_{min}(v) & \text{otherwise} \end{cases}$$

where $\Delta_{min}(v) =$ Min { **RectDist**$(v, t_i) - L'_{min}(v, t_i)$ } $\quad \forall\, t_i, t_i \in T_u$

Note that when the vertex v decreases some L'_{min} of parent vertex of v, (i.e., v becomes the closest vertex to some unconnected terminal vertex), the value of $\Delta_{min}(v)$ is

zero. And when there is only one unconnected terminal vertex, the searching process becomes equivalent to the A*-search [3] by using $\Delta_{min}(v)$ and not multiplying the 2/3 factor (i.e., $L_p(v) + L'_{min}(v, t_i) + \Delta_{min}(v) = L_p(v) +$ **RectDist**(v, t_i))

The global routing algorithm presented here uses a simple, modified version of the A*-search technique by using $L^{**}(v)$ as a cost during the search process. The minimum-cost path which is expanded from one vertex to the other is implemented by using a priority list, *OPEN*, and a list, *CLOSED* [3]. The *CLOSED* list contains vertices considered to be part of the minimum-cost path found so far. The *OPEN* list contains vertices being expanded from the vertices in the *CLOSED* list. Elements in the *OPEN* list are ordered in such a way that the first element of the list has the smallest cost.

Given a connection tree, V_{src}, and a set of unconnected terminal vertices, T_u, the global routing algorithm first puts each of the vertices of V_{src}, along with its cost, into the *OPEN* list. Then it proceeds by taking the first vertex of the *OPEN* list, *w*, from the list. Next the L^{**} of each of the child (adjacent) vertices of *w* (except the parent vertex of *w*) is evaluated and put into the *OPEN* list, and vertex *w* is put into the *CLOSED* list. Whenever a terminal vertex is taken from the front of the *OPEN* list, it means that the minimum-cost path has encountered that terminal vertex. The minimum-cost path can then be determined by tracing the vertex in the *CLOSED* list. All the vertices of the minimum-cost path, including the new terminal vertex, are added into the source group V_{src}. Then the path search process resumes with the new V_{src} and the remaining T_u.

In order to calculate L^{**} as described above and used as a cost during path searching, a vertex *v* in either the *OPEN* or *CLOSED* list is associated with the values of $L^{**}(v)$, $L_p(v)$, $L'_{min}(v,t_i)$, and *Parent*(*v*), which is the vertex from where *v* is expanded.

The global routing algorithm for finding a rectilinear Steiner tree connecting all terminal vertices in *T* is shown below.

```
GlobalRouting( T )
    t_src = RandomlyChooseATerminalVertex( T )
    V_src = { t_src }; T_u = T - { t_src }
    L_min() = GetMSTEdgeLength( V_src , t_src , T_u )
    do {
        Empty the OPEN and CLOSED lists
        for each v ∈V_src {
            ( -, L_min(), L** ) = ComputeVertexAttributes( v, 0, L_min(), T_u )
            SetVertexAttributes( v, NULL, 0, L_min(), L**, T_u )
            Put v into the OPEN list
        }
        w = GetOPENList()
        while ( w ∉ T_u ) {
            for each vertex v adjacent to w AND v ≠ Parent(w) AND v ∉V_src {
                ( L_p, L_min(), L** ) = ComputeVertexAttributes( v, L_p(w), L'_min(w), T_u )
                if ( L** < L**(v) ) {
```

```
                if ( v ∈ OPEN  OR v ∈ CLOSED ) {
                    SetVertexAttributes( v, w, Lp, Lmin(), L**, Tu )
                    if ( v ∈ OPEN )  remove v from the OPEN list
                    else             remove v from the CLOSED list
                    Put v into the OPEN list
                }
            } else {
                if ( v ∉ OPEN  AND v ∉ CLOSED ) {
                    SetVertexAttributes( v, w, Lp, Lmin(), L**, Tu )
                    Put v into the OPEN list
                }
            }
        }
        Put w into the CLOSED list
        w = GetOPENList()
    }
    Put w into the CLOSED list
    Vsrc = Vsrc ∪ { vertices traced back from w in the CLOSED list }
    Tu = Tu - {w}
    Lmin() = L'min(w)
  } while ( Tu ≠ ∅ )
}

ComputeVertexAttributes( v, Lp(w), L''min(), Tu )
  Lp = Lp(w)+ RectDist( Parent(v) , v )      /* w is the parent vertex of v */
  for each ti ∈ Tu
    Lmin(ti) = Min {  RectDist(v, ti), L''min(ti) }
  Δmin(v) = Min { RectDist(v, ti) - Lmin(ti) }   ∀ ti, ti ∈ Tu
```

$$L^{**} = \begin{cases} L_p + \frac{2}{3}\left[\Delta_{min}(v) + \sum_{\forall t_i, t_i \in T_u} L_{min}(t_i)\right] & if \ |T_u| > 1 \\ L_p + L_{min}(t_i) + \Delta_{min}(v) & otherwise \end{cases}$$

```
  return( (Lp , Lmin() , L** ) )
}
SetVertexAttributes( v, parent, Lp, Lmin(), L**, Tu ) {
  Parent(v) = parent
  Lp(v) = Lp ;   L**(v) = L**
  for each ti ∈ Tu
    L'min(v, ti) = Lmin(ti)
}
```

The value of $L'_{min}(t_i)$, $\Delta_{min}(v)$, and L^{**} are all based on function **RectDist** which is used to estimate the lower bound connection length between two given vertices. Because the connection path can only be routed along the routing edges, the value of **RectDist** may sometimes mislead the search process. For example in Figures 3, length of the shortest path between vertices v and t_i is much longer than the rectilinear distance between both vertices. To avoid this situation as much as possible, a one-step look ahead in the routing graph is used to estimate the shortest connection path between two given

vertices. Given a vertex v (whose parent vertex is $Parent(v)$) and a terminal vertex t_i, the estimated shortest length between v and t_i, **RectDist1**(v, t_i), can be computed as follows.

$$\textbf{RectDist1}(v, t_i) = \text{Min} \{ \textbf{RectDist}(v, u) + \textbf{RectDist}(u, t_i) \} \qquad \forall\, u,\, u \in A(v)$$

where $A(v)$ is a set of adjacent vertices of v excluding $Parent(w)$. **RectDist1**(v, t_i) is used for the computations of $L_{min}(t_i)$ and $\Delta_{min}(v)$ in the procedure **ComputeVertex-Attributes**.

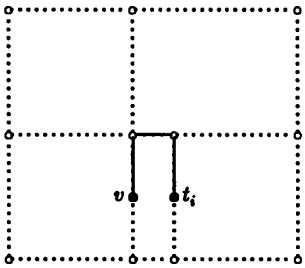

Figure 3. The shortest path from v to t_i is longer than the **RectDist**(v, t_i)

4. EXPERIMENTAL RESULTS

To demonstrate the superiority of the global router presented here over the previous work, three different routing graphs were used for the experiment. For each routing graph, eight sets of one thousand randomly generated multiterminal nets were routed with the global routing algorithm presented here and compared with the global routing results using the Clow's algorithm [3] as shown in Table 1. The results are also compared with the results obtained by using the Hsu's algorithm [7] (in Table 2) which is an improved-version of Clow's work obtained by modeling the unconnected terminal vertices as magnets which attract the path search process. For example (Table 2), 491 routes of 1000 9-terminal nets in Graph #2 produced from our algorithm are shorter than the routes obtained from Hsu's, and 109 routes produced from ours are longer. Figure 4 shows a distribution histogram of the differences between net lengths obtained from our algorithm and those obtained from Hsu's for the 1000 9-terminal nets (where a minus sign means that shorter net lengths from our algorithm were obtained). The experimental results show that our algorithm produces a larger number of shorter routes than the routes obtained from the other algorithms. Figures 5 shows the global route examples for 2 signal nets obtained from the three algorithms. It is obvious from the figure that our algorithm performs better than the others because most of the connection paths are shared by one another instead of having the shortest path from one terminal to the other. For example, for the global route on the right side of Figure 5, after terminals 1, 5, and 2 were connected, terminal 3 was selected to be the next target in both (5a.2) and (5b.2) and a shortest path connection was made. In (5c.2) however, a different path was chosen which is longer than that in (5a.2) and (5b.2), but the overall length is shorter because both terminals 3 and 4 were considered during the routing process.

#terminals	Graph #1 (44 vertices 64 edges)		Graph #2 (87 vertices 129 edges)		Graph #3 (41 vertices 60 edges)	
	%better	%worse	%better	%worse	%better	%worse
3	11.6	3.5	15.8	3.6	10.3	2.3
4	20.1	4.7	28.5	5.1	20.8	3.8
5	25.5	5.9	32.2	7.6	26.2	6.0
6	31.5	7.5	38.0	9.2	30.0	8.7
7	35.3	8.3	45.4	8.9	33.4	8.5
8	36.8	8.8	48.8	11.1	38.5	8.3
9	40.2	9.1	52.7	9.9	40.5	8.6
10	41.4	10.5	53.8	11.4	43.2	8.6

Table 1. Comparison of global routes between our algorithm and the Clow's algorithm.

#terminals	Graph #1 (44 vertices 64 edges)		Graph #2 (87 vertices 129 edges)		Graph #3 (41 vertices 60 edges)	
	%better	%worse	%better	%worse	%better	%worse
3	10.3	3.5	13.3	3.7	8.8	2.5
4	18.9	5.2	23.5	5.8	19.0	4.1
5	23.7	6.4	28.2	8.1	24.3	6.6
6	30.0	7.6	34.7	10.4	28.5	8.8
7	33.4	9.0	41.8	10.2	31.6	8.8
8	35.7	9.6	45.7	12.8	36.8	8.9
9	39.4	9.2	49.1	10.9	39.5	8.8
10	39.9	10.6	51.2	11.8	42.1	8.8

Table 2. Comparison of the global routes between our algorithm and the Hsu's algorithm.

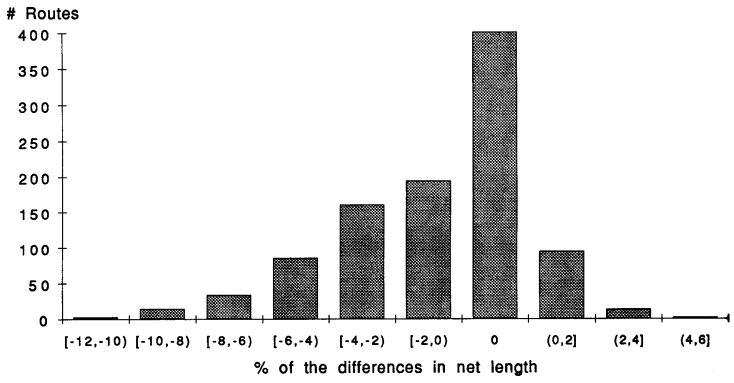

Figure 4. Distribution of the differences in net length of 1000 routes obtained from our algorithm and obtained from Hsu's algorithm.

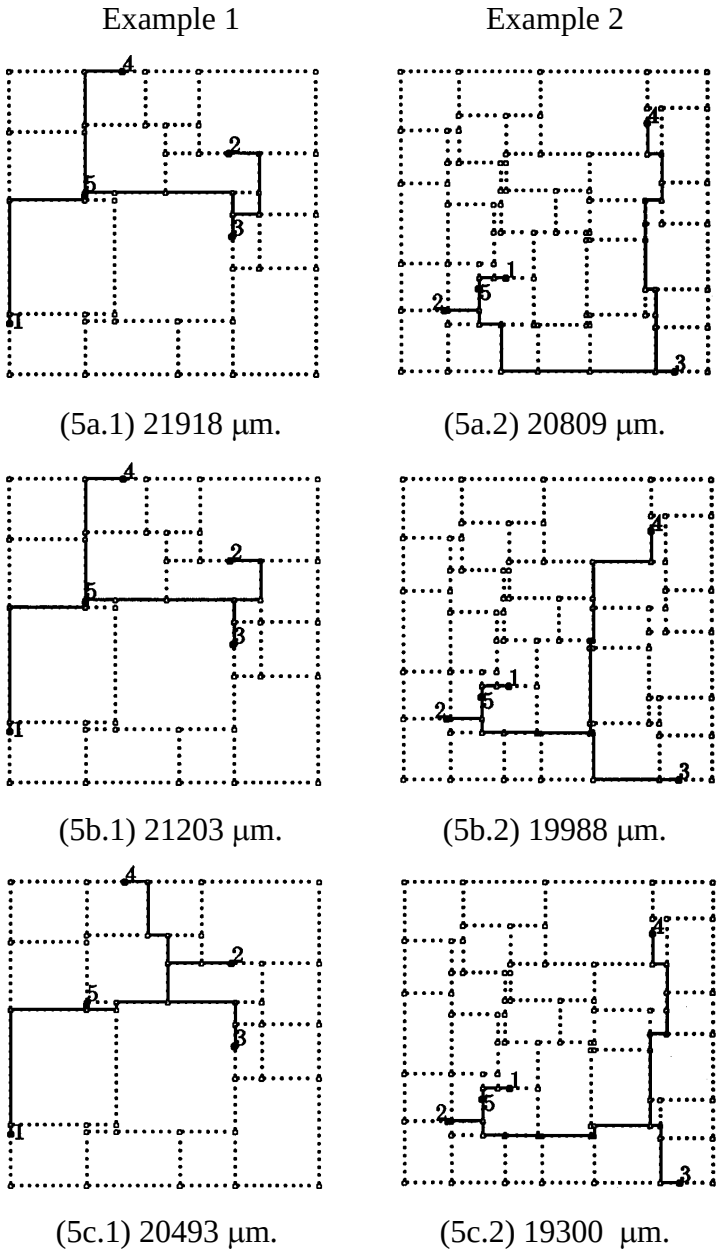

Figure 5. Two global route examples from (a) Clow's, (b) Hsu's, and (c) our algorithm.

5. CONCLUSIONS

This paper presents a new algorithm for finding a global route for a multiterminal net which minimizes the total net length, and is useful as part of a custom chip design methodology. The algorithm is a simple, modified version of the A*-search global routing algorithm [3]. It considers all the unconnected terminals simultaneously during the search for the minimum-cost path. The algorithm does not pre-select any specific terminal as the target to be connected, but rather allows the minimum-cost path to proceed in the minimum-cost direction and eventually hits the appropriate target. The connection path searching process use the estimated total net length of the minimum Steiner tree connecting all the terminals as the cost of a path. The length is determined by incrementally updating a previously obtained minimum rectilinear spanning tree and converting the total length of the tree to the lower-bound length of the minimum

rectilinear Steiner tree as presented by Hwang [11]. The experimental results show that a larger number of shorter global routes is achieved by the algorithm in comparison to the results obtained from previous approaches.

6. REFERENCES

1. Lee, C.Y., "An Algorithm for Path Connection and Its Applications", *IRE Trans. on Electronic Computers*, Vol. 2, No. 4, 346-365, 1961.
2. Dijkstra, E.N., "A Note on Two Problems in Connexion with Graphs" *Numer. Math*, Vol. 1, 269-271, 1959.
3. Clow, G.W., "A Global Routing Algorithm for General Cells", *ACM/IEEE Proc. 21st Design Automation Conf*, 45-51, 1984
4. Garey, M.R. and Johnson, G.S., "The Rectilinear Steiner Tree Problem is NP-complete", *SIAM Journal of Applied Mathematics*, Vol. 32, 826-834, 1977.
5. Chopra, S., "An Efficient Method for Custom Integrated Circuit Global Routing" ,*Proc. Custom Integrated Circuit Conf*, 11.3.1-11.3.6, 1988.
6. Fukui, M., Yamamoto, A., and et. al., "A Block Interconnection Algorithm for Hierarchical Layout System", *IEEE Trans on Computer-Aided Design*, Vol. CAD-6, No. 3, 383-391, 1988.
7. Hsu, Y.C., Pan, Y., and Kubitz, W.J., "A Path Selection Global Router" *ACM/IEEE Proc. 24th Design Automation Conf.*, 641-644, 1987.
8. Kimura, S., Kuba, N., Chiba, T., and Nishioka, I., "An Automatic Routing Scheme for General Cell LSI", *IEEE Trans on Computer-Aided Design*, Vol. CAD-2, No. 4, 285-292, 1983.
9. Prim, R.C., "Shortest Connection Networks and Some Generalizations", *Bell System Tech.*, Vol. 36, 1389-1401, 1957.
10. Dai, W., Asano, T., and Khu, E.S., "Routing Region Definition and Ordering Scheme for Building-Block Layout", *IEEE Trans. on Computer-Aided Design*, Vol. CAD-6. No. 3, 189-197, 1985
11. Hwang, F.K., "On Steiner Minimal Trees with Rectilinear Distance", *SIAM Journal of Applied Mathematics*, Vol. 30, No. 1, 104-114, 1976.