Efficient Maze-Running and Line-Search Algorithms for VLSI Layout

Si-Qing Zheng, Joon Shik Lim, and *S. Sitharama Iyengar*

Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803-4020

ABSTRACT

In this paper, a new construct called connection graph, G_C, is proposed. An efficient geometric algorithm for constructing G_C is given. We present a framework for designing a class of time and space efficient maze-running and line-search rectilinear shortest path and rectilinear minimum spanning tree algorithms based on G_C. We give several example maze-running and line-search algorithms based on G_C to demonstrate the power of G_C in designing good sequential *VLSI* routing algorithms.

Keywords: VLSI routing, maze-running algorithm, line-search algorithm, rectilinear shortest path and rectilinear minimum spanning tree.

1. Introduction

Most existing VLSI computer-aided design systems are based on the uniform grid model. With the grid model, wires connecting signal nets are considered as subgraphs of the grid. The major constraints, such as the minimum wire width and the minimum separation between wires, imposed by the current VLSI technologies are ensured by an automatic process once the abstract layout is generated. The objectives of the VLSI layout problems include finding a circuit layout such that the total area used is small and the wires interconnecting signal nets are short.

One of the most classic, but still up-to-date, method for VLSI routing is called sequential routing. In this method, a Steiner tree is constructed for each net in a sequential order. Once a Steiner tree connecting a net is constructed, the routing space is updated so that the constrains for routing subsequent nets can be enforced. The sequential routing method has received the most attention in practice. It is widely used for global routing and detailed routing, as well as printed circuit board (PCB) design. There are two basic classes of sequential routing algorithms: maze-running algorithms and line-search algorithms. Most of these algorithms are aimed at finding an obstacle-avoiding path, preferably a shortest one, on the grid between two given grid points. Generalizations of these algorithms to the problem of finding a spanning or Stein tree connecting multiple grid points are usually straightforward. In this paper, we first consider the problem of finding rectilinear obstacle-avoiding shortest paths. Then we generalize our results to the minimum rectilinear obstacle-avoiding spanning tree problem.

Let R be an $n \times n$ *grid* that consists of a set of *grid nodes* $\{ (x, y) \mid x, y$ are integers such that $1 \le x, y \le n \}$, and *grid edges* that connects grid nodes that are unit distance apart. A horizontal (vertical) grid line segment is a path consists of horizontal (vertical) grid edges. Let $B = \{B_1, \ldots, B_m\}$ be a set of mutually disjoint rectilinear polygonal obstacles whose boundaries lie on the grid lines of R. Each B_i is represented by a set of grid line segments whose endpoints are the *corners* of B_i. Let *R-B* denote the partial grid of R (i.e. subgraph of R) that consists of grid nodes that are not contained in the interior of any B_i, and grid edges that are not incident to any grid nodes contained in the interior of any B_i. In the context of VLSI design, grid R represents a rectangular area for the circuit layout. Circuit components and previously laid out wires are characterized by rectilinear polygons B_i with boundaries lie on the grid lines. The grid nodes and edges covered by the interior of these polygons are considered not available for subsequent routing steps. Thus, what available for completing the routing is a partial grid of R, and the portion of the grid that are not usable are treated as obstacles. Given a *source node* s and a *target node* t in $R - B$, $R - B$ is the entire search space for all possible obstacle-avoiding paths from s to t. It is sometimes convenient to use another planar graph derived from grid $R - B$ to represent the layout space. Let H be an $(n+1) \times (n+1)$ grid consists of grid node set $\{ (x, y) \mid x = i - 0.5, y = j - 0.5, i$ and j are integers such that $1 \le i, j \le n + 1 \}$ and grid edges connecting two grid nodes that are unit apart. Each face formed by four grid nodes of H is called a *cell*. We define the *offset representation of* $R - B$ as the the portion of grid H with all cells in the interior of portions corresponding B_i's removed. Then, a path from a source node s to a target node t in $R - B$ corresponds to a sequence of cells in the offset representation of $R - B$, each contains a grid node of $R - B$ on the path from s to t. In figure 1, we show an instance of $R - B$, and in figure 2 we give the offset representation of $R - B$ of figure 1.

The maze-running algorithms can be characterized as target directed grid expansion. The first such algorithm is Lee's algorithm [8], which is an application of the breadth-first shortest path search algorithm by Dijkstra [19] to the grid routing graph. In the worst case, Lee's algorithm takes $O(n^2)$ time. Several improved maze-running algorithms have been proposed [3, 4, 7, 8, 10-13, 15, 16]. Hadlock [4] applied the idea of using lower bound on distance to the target to direct the search proposed in [6] to the maze-running method. He gave a *minimum detour algorithm* [4]. He used a new labeling measure, called *detour number*, for each node. Let $M(s, t)$ denote the Manhattan distance (i.e. the distance in L_1 metric) between s and t. For a path P connecting s and t, the detour number $d(P)$ is the total number of units on P direct away from t. Then, the length of P is $M(s, t) + 2d(P)$. The minimum detour algorithm searches paths in the increasing order of detour numbers. It guarantees to find the shortest path using time between $O(n)$ and $O(n^2)$ for an $n \times n$ grid R. Soukup [16] proposed a fast algorithm that combines the depth-first-search with the breadth-first-search. This algorithm guarantees to find a path if it exists, but not necessarily an optimal path. Soukup's algorithm executes a depth-first-search from the source node toward the target node using "don't change direction" heuristic until an obstacle is hit or a target node is found. If an obstacle is hit, then a breadth-first-search is used for searching around the obstacle until a node directs to the target node is reached. Then, depth-first-search is continued. In figures 3, 4, and 5, we show the expanded nodes generated by Lee's algorithm, Hadlock's algorithm and Soukup's algorithm, respectively, using the offset grid representations. In these figures little circles and solid dots are the expanded nodes, and the dots represent a path from s to t.

Since the search space of all previous maze-running algorithms are represented as dense grid graphs, they are inherently inefficient in both time and space. The line-search algorithms have been proposed to achieve better performance. These algorithms use powerful computational geometry techniques to represent the search space by a set of line segments rather than unit grid edges. Consequently, they save space and quickly find a simple-shaped paths. The major drawback of the line-search algorithms is that they usually do not guarantee finding a shortest path. Early line-search algorithms are reported in [5] and [9]. The basic operations of algorithm by Mikami and Tabuchi [9] are as follows. First, straight lines are emanated from node s and node t in all possible directions. These search lines are called level-0 trial lines and stored in a temporary storage. Then, the path search is conducted by a iterating process. At the ith iteration, the following operations are preformed: pick up level-i trial lines one by one from the temporary storage. Along each such trial line, trace all grid nodes, and emanate new lines perpendicular to the trial line from these nodes. These newly generated line segments, which end either at the boundary of an obstacle B_i or the boundary of the grid R, are identified as level-$(i+1)$ trial lines. This process continues until a trial line from s meets a trial line from t. This algorithm finds a path from s to t if there exists one, but the path is not generated to be the shortest one. Figure 6 shows a running example of this algorithm. The line-search algorithm given in [5] is similar to the one in [9]. Another type of line-search algorithms achieve better expected performance by restricting the search on a graph that is much smaller than the given grid. For example, the line-search algorithm given in [18] is as follows. First, a special grid graph, called *track graph* G_T, is constructed from R and B. Then, a path from s to t on G_T is constructed by applying Dijkstra's algorithm. Since G_T is usually much smaller than the original grid, and G_T can be constructed efficiently, the time and space performances better than that of maze-running algorithms can be expected. However, the path found using G_T may not be the shortest one.

The major contributions of this paper are as follows:

(1) We introduce a new construct called connection graph, G_C, to reduce the size of search space.

(2) We show that there always exists a simplest minimum spanning tree connecting a set S of nodes in G_C with total edge length equal to the length of a minimum spanning tree of S in $R - B$.

(3) We give an efficient geometric algorithm for constructing the connection graph G_C.

(4) We present a framework for designing a class of time and space efficient maze-running and line-search shortest path and minimum spanning tree algorithms based on G_C.

(5) We give several example maze-running and line-search algorithms based on G_C to demonstrate the power of G_C in designing good sequential *VLSI* routing algorithms.

(6) Since the connection graph G_C is much sparser than grid $R - B$ in practice, and updating G_C after the wires connecting a net are introduced can be efficiently done by locally modifying G_C, our approach provide a powerful and versatile tool for designing efficient sequential *VLSI* routing algorithms.

2. Connection Graph

In this section, we introduce the connection graph G_C for the shortest path problem. More general form of G_C will be discussed later. Let $HL(R, B)$ and $VL(R, B)$ be the sets of horizontal and vertical line segments of the boundaries of R and obstacles in B, respectively. We define a horizontal (vertical) line segment $l = (u, v)$ in $R - B$ as a *maximal horizontal (vertical) line segment* of $G - B$ if l does not cross any B_j in B, and u and v are the only two points on l that are also on the boundaries of R or obstacles in B. Let $HL(R - B) = \{ l \mid l = (u, v)$ is a maximal horizontal line segment of $R - B$ such that at least one of its endpoints u and v is a corner of some B_i in $B\}$ and $VL(R - B) = \{ l \mid l = (u, v)$ is a maximal vertical line segment of $R - B$ such that at least one of its endpoints u and v is a corner of some B_i in $B\}$. Let $l_h(s)$ ($l_v(s)$) be the maximal horizontal (vertical) line segment of $R - B$ that contains s, the source node. We similarly define two line segment $l_h(t)$ and $l_v(t)$, which are the maximal line segments containing t, the target. The nodes of G_C

are the intersection points of the line segments in $HL(R \cup B) \cup VL(R \cup B) HL(R-B) \cup VL(R-B) \cup \{l_h(s), l_v(s), l_h(t), l_v(t)\}$, and the edges of G_C are the subsegments generated by these line intersections. The connection graph G_C for the example of figure 1 is given in figure 7.

The main purpose of introducing connection graphs is to reduce the search space in which a shortest path can be found. This should lead to shortest path algorithms that require less storage and time resources. The following theorem shows that the problem of finding a shortest path in $R-B$ can be reduced to the problem of finding a shortest path in G_C.

Theorem 1: If there exists a path from s to t in $R-B$, then the length of shortest path from s to t in G_C is equal to the length of the shortest path from s to t in $R-B$.

In the context of VLSI layout, in addition to minimizing the length of the path connecting two nodes, it is desirable to minimize the number of turning points on the path. We say that a shortest path P between s and t is a simplest shortest path if P contains minimum number of turning points among all shortest paths between s and t.

Theorem 2: If there exists a path from s to t in $R-B$, then a simplest shortest path in $R-B$ can be found in G_C.

We observe the following additional properties of G_C: (i) For practical VLSI layout problems, G_C is much sparser than $R-B$. (ii) In the context of VLSI layout, a path P between s and t in G_C corresponding to a wire connecting a net of two terminals, s and t, and once this wire is included into the routing solution, it will be considered as a obstacle for subsequent routing steps. Then, updating G_C to include P as an obstacle can be efficiently done by locally changing the structure G_C. Based on theorem 1, theorem 2 and these two properties, a class of rectilinear shortest path algorithms for VLSI routing can be designed using the connection graph G_C, instead of $R-B$.

3. Construction of Connection Graphs

We show how to efficiently construct the connection graph G_C, from given rectangular boundary R and a set B of mutually disjoint rectilinear polygonal obstacles in R. The construction of G_C uses the *plane-sweep* technique from computational geometry [22]. G_C can be constructed by first construct $HL(R, B) \cup HL(R-B)$ and $VL(R, B) \cup VL(R-B)$. Then, all intersection points of line segments in $HL(R, B) \cup HL(R-B)$ and $VL(R, B) \cup VL(R-B)$ are generated. Finally, line segments $l_h(s)$, $l_v(s)$, $l_h(t)$ and $l_v(t)$ and their intersections with the segments in $HL(R, B) \cup HL(R-B) \cup VL(R, B) \cup VL(R-B)$ are generated. We assume that G_C is represented by the adjacency lists. Since the methods for constructing $HL(R, B) \cup HL(R-B)$ and $VL(R, B) \cup VL(R-B)$ are similar, we only describe the procedure for constructing $HV(R, B) \cup HV(R-B)$.

The set $HV(R, B)$ is given as part of input. We only need to generate $HV(R-B)$ to complete the construction of $VL(R, B) \cup VL(R-B)$. To facilitate our discussions, we introduce a couple of new notions. We call a vertical boundary line segment l of an obstacle B_i a *left (right) segment* of B_i if the interior of B_i is to the right (left) of l. We call a corner point w formed by two orthogonal boundary segment $l_1 = (u, w)$ and $l_2 = (w, v)$ of B_i a *convex corner* if there exists a line segment $l' = (a, b)$ such that a is on l_1 and b is on l_2, $a \neq w$, $b \neq w$, and all point on l' except a and b are in the interior of B_i. If such a line segment does not exist, w is called a *concave corner* of B_i. The following procedure generates all segments in $VL(R-B)$. In this procedure, we use $x(l)$ to denote the x-coordinate of vertical segment l. We use $low(l)$ and $high(l)$ to represent the lower and upper endpoints of l, respectively, and use $x(u)$ and $y(u)$ to represent the x- and y-coordinates of point u, respectively.

procedure *VERTICAL_SGMT*
  Sort all vertical boundary segments of obstacles in B in lexicographical order by their lower endpoints into a queue Q;
  $x' := x(l)$, where l is the first segment in Q;
  while Q is not empty **do**
    $l := dequeue(Q)$;
    if $x' \neq x(l)$ **then** $x' := x(l)$;
    case
      :l is a left boundary segment and $low(l)$ is a convex corner:
        find the largest element y' in T that is smaller than $y(low(l))$
        let $u = (x, y')$ and $v = (x, y(low(l))$;
        $VL(R-B) := VL(R-B) \cup \{(u, v)\}$;
        $T := T \cup \{y(low(l))\}$;
      :l is a left boundary segment and $low(l)$ is a concave corner:
        $T := T - \{y(low(l))\}$;
      :l is a right boundary segment and $low(l)$ is a convex corner:
        find the largest element y' in T that is smaller than $y(low(l))$
        let $u = (x, y')$ and $v = (x, y(low(l))$;
        $VL(R-B) := VL(R-B) \cup \{(u, v)\}$;
        $T := T - \{y(low(l))\}$;
      :l is a right boundary segment and $low(l)$ is a concave corner:
        $T := T \cup \{y(low(l))\}$;
    endcase
    case
      :l is a left boundary segment and $high(l)$ is a convex corner:
        find the smallest element y' in T that is larger than $y(high(l))$

let $u = (x, y')$ and $v = (x, y(high(l))$;
$VL(R-B) := VL(R-B) \cup \{(u, v)\}$;
$T := T - \{y(high(l))\}$;
:l is a left boundary segment and $high(l)$ is a concave corner:
$T := T \cup \{y(high(l))\}$;
:l is a right boundary segment and $high(l)$ is a convex corner:
find the smallest element y' in T that is larger than $y(high(l))$
let $u = (x, y')$ and $v = (x, y(high(l))$;
$VL(R-B) := VL(R-B) \cup \{(u, v)\}$;
$T := T - \{y(high(l))\}$;
:l is a right boundary segment and $high(l)$ is a concave corner:
$T := T \cup \{y(high(l))\}$;
endcase
endwhile
end *VERTICAL_SGMT*

Theorem 3: Connection graph G_C can be constructed in $O(n_c \log n_c + e)$ time, where n_c is the total number of corner points of B and e is the total number of edges in G_C.

4. Maze-Running and Line-Search Algorithms Based on Connection Graphs

Using the connection graph G_C, we can obtain a class of efficient modified shortest path algorithms. These algorithms may use maze-running techniques, or line-search techniques, or the combination of these techniques. We discuss a few possibilities.

Let $(R-B) \cap G_C$ denote the partial uniform grid defined on G_C. Clearly, $(R-B) \cap G_C$ can be constructed from G_C by breaking each edge of G_C into grid edges of unit length. The time required for this construction is $O(l_c)$ and the space for $(R-B) \cap G_C$ is $O(l_c)$, where l_c is the total edge length of G_C. In figure 8 we show $(R-B) \cap G_C$ for the example of figure 1 by marking its cells with x's. Then, all existing maze-running algorithms can be applied to the partial grid $(R-B) \cap G_C$. Since $(R-B) \cap G_C$ is always consists of less cells (in the offset grid representation) than $G-B$, these modified maze-running algorithms are more time and space efficient than their original ones. In figures 9, 10 and 11, we show the improved performance of modified Lee's algorithm, Hadlock's algorithm and Soukup's algorithm. The meaning of little circles and solid dots is the same as in figures 3, 4 and 5. Compared with figure 3, 4 and 5, the number of expanded nodes by each of these modified algorithms is much smaller than that of the original algorithm. Similar improvements can be observed in the modified versions of other existing maze-running algorithms. It should be mentioned that all previously introduced coding methods for reducing the storage requirement are valid on the grid graph $(R-B) \cap G_C$.

Theorem 4: An obstacle-avoiding shortest path from s to t can be computed by modified maze-running algorithms in no more than $O(l_c)$ time and space from the connection graph G_C, where l_c is the total edge length of G_C.

Modified maze-running algorithms may still require excessive storage and time in the worst case. The connection graph G_C can be considered as a "supergrid", which consists of much less number of grid nodes and edges than $R-B$ (Note: in any grid graph the number of nodes and the number of edges are about the same since the degree of each node is no more than 4.) Based on G_C, we can obtain a set of modified line-search algorithms from the existing ones. We give two examples. By applying the line-search algorithm of Mikami and Tabuchi to the connection graph G_C, we obtain a modified line-search algorithm which only generate trial lines that are in G_C. Since G_C is much sparser than $G-B$, finding a path from s to t in G_C requires much less time and space. Note that the original algorithm in [9] cannot be directly applied to the problem instances in which the obstacles are not defined on a uniform grid. Using the connection graph G_C, this restriction is removed. As the original algorithm by Mikami and Tabuchi, this modified algorithm does not guarantee a shortest path. The performance of this modified algorithm for the example of figure 6 is shown in figure 12.

Using the connection graph G_C, most of existing maze-running algorithm can be transformed into line-search algorithms. The performance of the line-search versions of Lee's algorithm, Hadlock's algorithm and Soukup's algorithm for the example of figure 1 are shown in figures 13, 14 and 15, respectively. In these figures, little circles and solid dots are the expanded nodes, and the dots represent a path from s to t. In these line-search algorithms, edges of G_C, each of them may consist of many unit grid edges of $G-B$, are consider one at a time. Since the number of edges (and nodes) is much smaller than the number of grid edges of $G-B$, these line-search algorithms are much more time and space efficient than their maze-running versions on $(G-B) \cap G_C$. Note that the line-search version of Lee's algorithm here is exactly Dijkstra's shortest path algorithm applied to G_C. Although in general the size of the track graph G_T introduced in [18] is smaller than the connection G_C, applying the the Dijkstra's or Hadlock's algorithm to G_T does not always guarantee a shortest path. In contrast, by theorem 1, using Dijkstra's algorithm and Hadlock's algorithm on G_C a shortest path is always guaranteed. This leads to the following claim.

Theorem 5: An obstacle-avoiding shortest path from s to t can be computed from G_C in no more than $O(e + m \log m)$ time and $O(e)$ space, where e is the total number of edges (and nodes) in the connection graph G_C, and m is the total number of nodes of G_C expanded when a shortest path is found.

It is important to note that the line-search versions of Lee's algorithm, Hadlock's algorithm and Soukup's algorithm can be applied to problem instances in which the boundaries of obstacles in B are not defined on a uniform grid.

5. Generalizations

A direct generalization of the shortest path algorithms presented in this paper is the design of efficient maze-running algorithms and line-search algorithms for constructing obstacle-avoiding minimum length rectilinear Steiner trees and spanning trees. The Steiner tree problem corresponds to the problem of introducing wires to connect a multi-terminal net on a *VLSI* chip or a printed-circuit board. The minimum rectilinear Steiner tree problem is NP-complete [20]. It is known that the ratio between a the length of a rectilinear minimum spanning tree (*MST*) and the length of a rectilinear minimum Steiner tree is no more than 3/2 [21]. In practice, a rectilinear minimum spanning tree is first constructed, and then modified to obtain a Steiner tree. Given a rectangle boundary R, a set B of mutually disjoint rectilinear polygonal obstacles in R and a set $S = \{t_1, \ldots, t_p\}$ of points in $R - B$, the objective of the *MST* problem is to construct a rectilinear *MST* T of minimum total length that connects all points in S, and any line segments in T does not cross any boundary segment of R and B. The connection graph G_C for this problem is defined as follows. For each $t_i \in S$ construct two line segments, horizontal segment $l_h(t_i)$ and vertical segment $l_v(t_i)$, passing through t_i such that their two endpoints are the only points on them that are the boundary points of R and/or obstacles in B. Then, the intersection points of $HL(R, B) \cup VL(R, B) \cup HL(R-B) \cup VL(R-B) \cup \{l_h(t_i), l_v(t_i) \mid i = 1, 2, ..., p\}$ are the nodes of G_C and the line segments with endpoints from these intersection points are the edges of G_C. If we treat each $t_i \in S$ as a corner point, we can obtain an efficient algorithm for constructing G_C using the plane-sweep technique.

Theorem 6: Given R, B and S, the connection graph G_C can be constructed in $O((n_c + s)\log(n_c + s) + e)$ time, where n_c is the total number of corner points of B, s is the number of points in S and e is the total number of edges in G_C.

The following properties of generalized G_C are important for designing efficient maze-running and line-search algorithms for constructing obstacle-avoiding *MST*'s.

Theorem 7: If there spanning tree S in $R - B$, then the length of the *MST* of S in G_C is equal to the length of the *MST* of S in $R - B$.

In a spanning tree T in the grid graph, we call a node with degree greater than 3 a *junction*. In the context of VLSI layout, it is desirable to minimize the the number of junctions and the number of turning points in a spanning tree. We say that an *MST* T of S is a simplest *MST* if the sum of the number of junctions and the number of turning points is the minimum among all *MST*'s of S.

Theorem 8: If there exists a spanning tree of S in $R - B$, then a simplest minimum spanning tree of S in $R - B$ can be found in G_C.

By these three theorems, a class of maze-running and line-search algorithms for the rectilinear *MST* problem can be designed using the connection graph G_C, instead of $R - B$. The performance improvements in these *MST* algorithms should be similar to the shortest path algorithms we demonstrated in the previous section. As a special example, based on the techniques proposed in [18], we have the following claim:

Theorem 9: A minimum spanning tree of S in $R - B$ can be found in G_C in no more than $O(e \log e)$ time, where e is the number of edges in G_C.

The connection graph G_C can be quite dense. One open problem is to identify and characterize a graph whose size is much smaller than G_C, yet good enough to guarantee the existence of shortest paths and minimum spanning trees. If such a connection graph can be constructed from R and B efficiently, more effective maze-running and line-search algorithms are possible.

6. References

[1] S.B. Akers, "A Modification of Lee's Path Connection Algorithm," *IEEE Transactions on Electronic Computers*, EC-16(2), pp. 97-98, 1967.

[2] K.L. Clarkson, S. Kapoor, and P.M. Vaidya, "Rectilinear Shortest Paths through Polygonal Obstacles in O(n(log n)2) Time," In *Proceedings of the Third Annual Conference on Computational Geometry*, pp. 251-257, ACM, 1987.

[3] J.M. Geyer, "Connection Routing Algorithms for Printed Circuit Boards," *IEEE Transactions on Circuit Theory*, CT-18(1), pp. 95-100, 1971.

[4] F.O. Hadlock, "A Shortest Path Algorithm for Grid Graphs," *Networks*, 7, pp. 323-334, 1977.

[5] D.W. Hightower, "A Solution to Line Routing Problems on the Continuous Plane," In *Proceedings of the Sixth Design Automation Workshop*, pp.1-24, IEEE, 1969.

[6] P. Hart, N. Nilsson and B.Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems, Science and Cybernetics*, SCC-4(2), pp. 100-107, 1968.

[7] J.H. Hoel, "Some Variation of Lee's Algorithm," *IEEE Transactions on Computers*, C-25(1), pp. 19-24, 1976.

[8] C. Y. Lee, "An Algorithm for Path Connections and Its Applications," *IRE Transactions on Electronic Computers*, EC-10(3), pp. 346-365, 1961.

[9] K. Mikami, K. Tabuchi, "A Computer Program for Optimal Routing of Printed Circuit Connectors," *IFIPS Proceedings*, H-47, pp. 1475-1478, 1968.

[10] E. F. Moore, "The Shortest Path Through a Maze," *Annals of the Harvard Computation Laboratory*, Vol. 30, Pt. II, pp. 285-292,

1959.

[11] T. Ohtsuki, "maze-running and Line-search Algorithms," In T. Ohtsuki, editor, *Advances in CAD for VLSI*, Volume 4: Layout Design and Verification, pp.99-131, North-Holland, New York, 1986.

[12] I. Pohl, "Heuristic Search viewed as path finding in a Graph," *Artificial Intelligence*, Vol. 1, pp.193-204, 1970.

[13] I. Pohl. "Bi-Directional Search," Machine Intelligence, Volume 6, pp. 127-140, 1971.

[14] P.J. Rezend, D.T. Lee, and Y.-F. Wu, "Rectilinear Shortest Paths with Rectangular Barriers," In *Proceedings of the Second Annual Conference on Computational Geometry*, pp. 204-213, ACM, 1985.

[15] F. Rubin, "The Lee Path Connection Algorithm," *IEEE Transactions on Computers*, C-23(9), pp. 907-914, 1974.

[16] J. Soukup, "Fast Maze Router," *Proceedings 15th Design Automation Conference*, pp.100-102, 1978.

[17] P. Widmayer, "Network Design Issues in VLSI," Manuscript, Department of Computer Science, University of Freiburg, Freiburg, West Germany, 1989.

[18] Y.-F. Wu, P. Widmayer, M.D.F. Schlag, C.K. Wong, "Rectilinear Shortest Paths and Minimum Spanning Trees in the Presence of Rectilinear Obstacles," *IEEE Transactions on Computers*, C-36(3), pp.321-331, 1987.

[19] E.W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", *Numerische Mathematik*, Vol. 1, pp. 269-271, 1959.

[20] M.R. Garey and D.S. Johnson, "The Rectilinear Steiner Tree Problem is NP-Complete", *SIAM J. Comput.*, Vol. 15, pp. 317-340, 1986.

[21] F.K. Hwang, "On Steiner Minimal Trees with Rectilinear Distance", *SIAM J. Appl. Math.*, Vol. 30, pp. 104-114, 1976.

[22] F. P. Preparata and M.I. Shamos, *Computational Geometry*, Springer-Verlag, 1985.

7. Figures

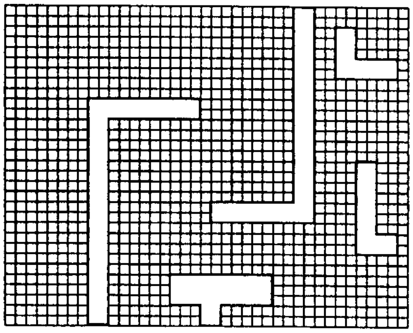


Figure 1: Grid representation of $R - B$,

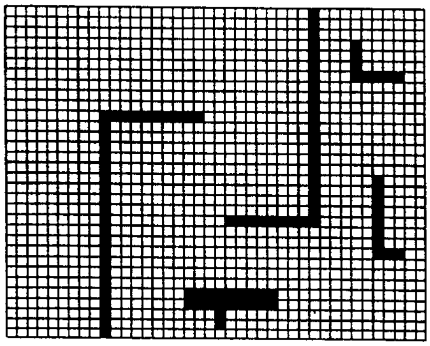
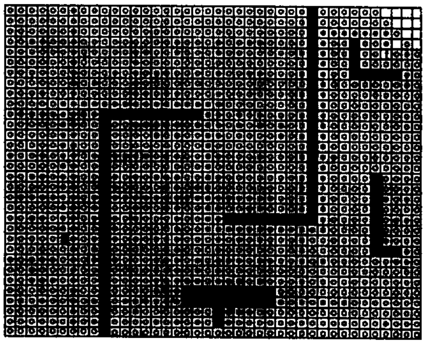
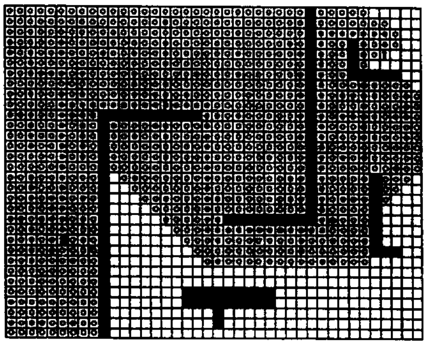


Figure 2: Offset grid representation of $R - B$



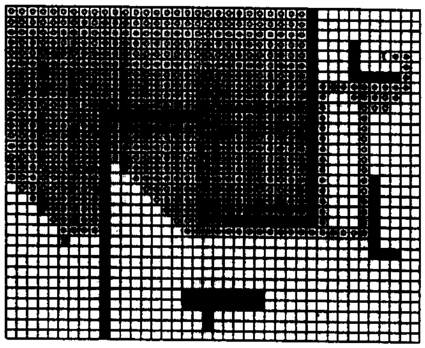
s: Source Node, t: Target Node, o: Extended Nodes

Figure 3: Expanded Nodes by Lee Algorithm



s: Source Node, t: Target Node, o: Extended Nodes

Figure 4: Expanded Nodes by Hadlock Algorithm



s: Source Node, t: Target Node, o: Extended Nodes

Figure 5: Expanded Nodes by Soukup Algorithm

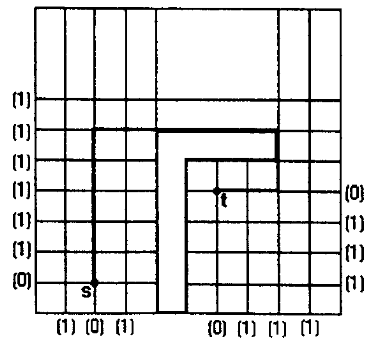


Figure 6: Line Search Algorithm of Mikami and Tabuchi

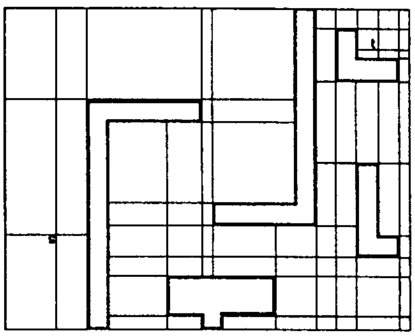


Figure 7: The connection graph G_C

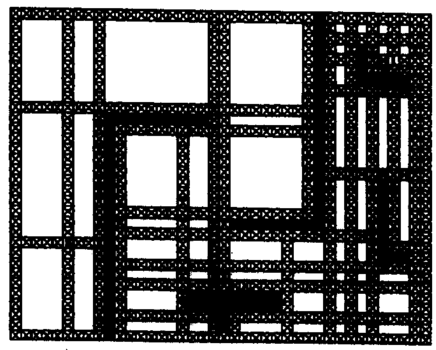


Figure 8: $(R - B) \cap G_c$. Grid Nodes on are marked by X's.

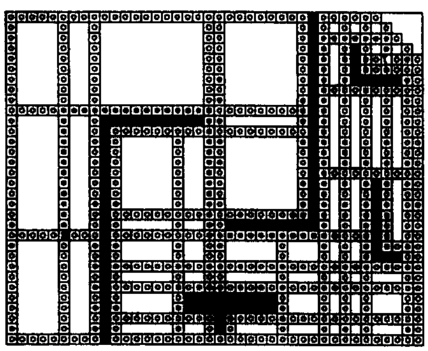


Figure 9: Expanded Nodes by the modified Lee Agorithm

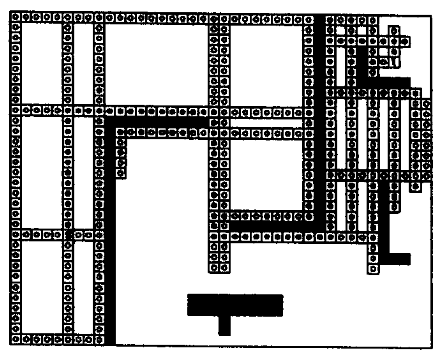


Figure 10: Expanded Nodes by the modified Hadlock Agorithm

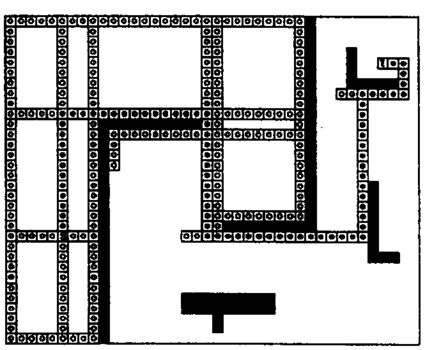


Figure 11: Expanded Nodes by the modified Soukup Agorithm

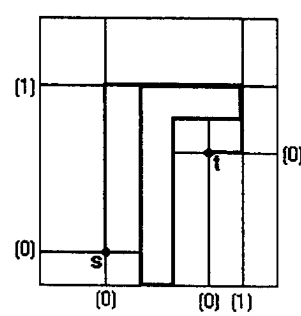


Figure 12: Modified Version of the Algorithm by Mikami and Tabuchi

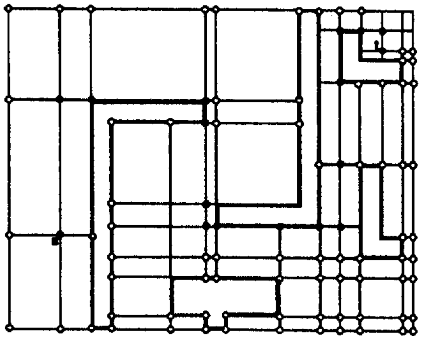


Figure 13: The Performance of the Line-Search Version of Lee Algorithm

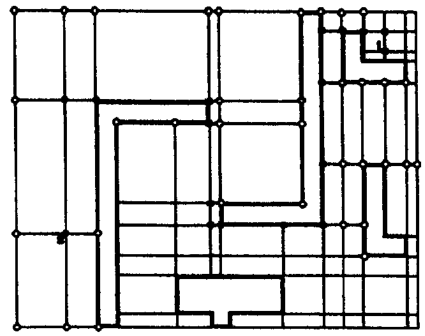


Figure 14: The Performance of the Line-Search Version of Hadlock Algorithm

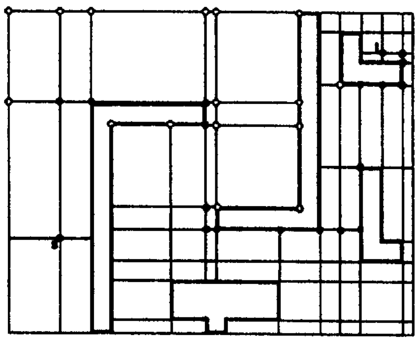


Figure 15: The Performance of the Line-Search Version of Soukup Algorithm