VLSI DESIGN
1999, Vol. 9, No. 1, pp. 91–104
Reprints available directly from the publisher
Photocopying permitted by license only

Finding Combined L_1 and Link Metric Shortest Paths in the Presence of Orthogonal Obstacles: A Heuristic Approach

JOON SHIK LIM [a,*], S. SITHARAMA IYENGAR [b] and SI-QING ZHENG [b]

[a] *Department of Computer Science, Kyung Won University, Sung Nam 461-701, Korea;*
[b] *Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803*

(Received 13 March 1997)

This paper presents new heuristic search algorithms for searching combined rectilinear (L_1) and *link metric* shortest paths in the presence of orthogonal obstacles. The *Guided Minimum Detour* (*GMD*) algorithm for L_1 metric combines the best features of maze-running algorithms and line-search algorithms. The *Line-by-Line Guided Minimum Detour* (*LGMD*) algorithm for L_1 metric is a modification of the *GMD* algorithm that improves on efficiency using line-by-line extensions. Our *GMD* and *LGMD* algorithms always find a rectilinear shortest path using the *guided* A^* search method without constructing a *connection graph* that contains shortest paths. The *GMD* and the *LGMD* algorithms can be implemented in $O(m+e\log e+N\log N)$ and $O(e\log e+N\log N)$ time, respectively, and $O(e+N)$ space, where m is the total number of searched nodes, e is the number of boundary sides of obstacles, and N is the total number of searched line segments. Based on the *LGMD* algorithm, we consider not only the problems of finding a link metric shortest path in terms of the number of bends, but also the combined L_1 metric and link metric shortest path in terms of the length and the number of bends.

Keywords: L_1 and link metric shortest paths, maze-running algorithms, line-search algorithms

1. INTRODUCTION

The problem of finding a shortest path in the presence of rectilinear obstacles has applications in *robotics*, *VLSI design*, and *geographical information systems* [13]. In VLSI design, there are two basic classes of sequential algorithms aimed mostly at finding an obstacle-avoiding path, preferably the shortest one, between two given points: *maze-running* algorithms and *line-search* algorithms. The maze-running algorithms can be characterized as target-directed grid extension. The first such algorithm is *Lee* algorithm [12], which is an application of the *breadth-first* shortest path search algorithm. The major disadvantage of the original Lee algorithm is that it requires $O(n^2)$ memory

*Corresponding author.

and running time in the worst case for $n \times n$ grid graphs. There are a large number of variations (*e.g.* [1, 6–8, 10, 13, 14, 18–21, 23, 24]) of the original Lee algorithm. Hart *et al.* [8] proposed the idea of using a lower bound on the *Manhattan distance* between a source node and a target node. Hadlock applied this to the shortest path algorithm, called *Minimum Detour* (*MD*) *algorithm* [7]. For each searched grid node in a grid graph, he used a new labeling method called *detour number* which is the total number of grid nodes moves away from a target node t during the search. Soukup [24] incorporated the depth-first search with the breadth-first search to reduce search space and time. This algorithm guarantees finding a path if it exists, but not necessarily the shortest one.

Since all partial paths generated by maze-running algorithms are represented by unit grid line segments, these algorithms are still considered memory-and-time inefficient. Line-search algorithms [9, 16] have been proposed to achieve improved performance. Since such algorithms search a path as a sequence of line segments, they save memory and quickly find a simple-shaped path. The idea behind these algorithms is to reduce the size of representation for all searched grid nodes by a set of long line segments. The major drawback of the line-search algorithms is that they usually do not guarantee finding a shortest path. Several recent line-search algorithms (*e.g.* [4, 13, 17, 22, 25]) are based on powerful computational geometry techniques. Wu *et al.* [25] introduced a rather small connection graph, the *track graph*, which may contain all possible paths from a start point to a target point including the shortest path, but it is not a strong connection graph. The run time of their algorithm is $O((e+k)\log t)$, where e is the total number of boundary sides of obstacles, t is the total number of extreme edges of all obstacles, and k is the number of intersections among obstacle tracks, which is bounded by $O(t^2)$ Zheng *et al.* [27] proposed an efficient geometric algorithm for constructing a connection graph G_c. They presented a framework for designing a class of time-and-space efficient rectilinear shortest path and rectilinear minimum spanning tree algorithms based on G_c. De Rezende *et al.* [22] considered a special case that all obstacles are rectangles. Their algorithm constructs a strong connection graph and finds a shortest path from s to t in time $O(n \log n)$, where n is the number of obstacles. Clarkson *et al.* [4] generalized the shortest path problem to the case of arbitrarily shaped obstacles. Their algorithm runs in time $O(n \log^2 n)$. For the special case where obstacles are just rectilinear line segments, Berg *et al.* [2] studied the shortest path problem in a combined metric that generalizes the L_1 metric and the rectilinear link metric. A good survey of algorithms for the rectilinear shortest path problem can be found in [13].

In this paper, we introduce new heuristic algorithms, the *Guided Minimum Detour* (*GMD*) algorithm and the *Line-by-Line Guided Minimum Detour* (*LGMD*) algorithm. The *GMD* algorithm incorporates the best features of maze-running algorithms and line-search algorithms. The *GMD* algorithm uses a heuristic search method called *guided* A^* that uses the A^* search [8] with the heuristic "*don't change direction*". The *GMD* algorithm reduces space, compared with the existing maze-running algorithms without losing its optimality. On the basis of *GMD* algorithm, we present a modified algorithm called the *LGMD* algorithm. The *LGMD* algorithm is a line-search algorithm, which replaces the extended grid nodes in the *GMD* algorithm to line segments. In the worst case, our *LGMD* algorithm has the time and space complexities comparable to those of existing algorithms.

2. A NEW ALGORITHM: GUIDED MINIMUM DETOUR ALGORITHM (*GMD*)

2.1. Definitions and Implementation

Let G be an $n \times n$ uniform grid graph that consists of a set of *grid nodes* $\{(x, y) | x$ and y are integer coordinates such that $1 \leq x \leq n$ and $1 \leq y \leq n\}$ (see Fig. 1). For example, a grid node (3, 4) is located in the third of x-axis and the fourth of y-axis. The

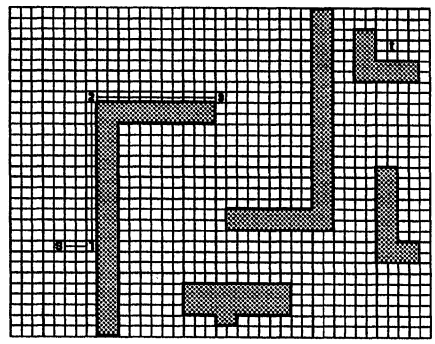


FIGURE 1 A path $[s \rightarrow 1 \rightarrow 2 \rightarrow 3]$ and obstacles in a grid graph G.

length between any two adjacent grid nodes in G is assumed to be 1. A horizontal or a vertical line segment depicted by $a \rightarrow b$ in which all grid nodes between a and b make a horizontal or a vertical line in G. For example, there are three line segments ($s \rightarrow 1$, $1 \rightarrow 2$, and $2 \rightarrow 3$) over a path from s to 3 in the Figure 1. Let $B = \{B_1, B_2, \ldots, B_p\}$ be a set of mutually disjoint rectilinear simple polygons with boundaries on G. Each polygon in B is an obstacle.

A path P in G is represented by $P = [v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k]$ with a set of directed line segment $\{v_i \rightarrow v_{i+1} | i = 1, \ldots, k-1$ and v_i represents a grid node and v_{i+1} are adjacent for $1 \leq i \leq k-1\}$. The length of, denoted by $L(P)$ is k. The length k can be calculated using the Manhattan Distance and the detour length by the following Theorem 1.

For any path P in G, the *detour length* of P, denoted by $DL(P)$, is the total number of grid nodes that proceed away from t in P. Let $M(s, t)$ denote the Manhattan Distance between the start node s and the target node t in G. Clearly, $L(P) = M(s, t) + 2 \times DL(P)$ is the length of a shortest path P from s to t if $DL(P) \leq DL(P')$, where P' is any path from s to t. In the following theorem, we restate the main results of [7].

THEOREM 1 [7]

1. *A path* $P = [s \rightarrow \cdots \rightarrow t]$ *has a length* $L(P) = M(s, t) + 2 \times DL(P)$.
2. *If* P *is a shortest path from* s *to* t, *then* $DL(P) = min\{DL(P) | P$ *is a set of all paths from* s *to* $t\}$.
3. *The path generated by the minimum detour algorithm of* [7] *is a shortest one with the minimized* $DL(P)$.

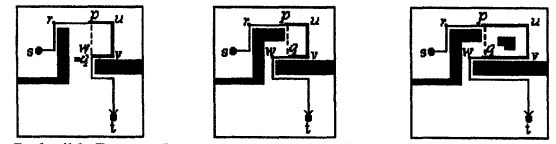
A path P can be represented as a sequence of directed line segments such that no two consecutive line segments have the same direction. A subpath $D = [r \rightarrow u \rightarrow v \rightarrow w]$ in P is called a *detour* (Figs. 2(a) and (b)), if directions of the three consecutive line segments $r \rightarrow u$, $u \rightarrow v$, and $v \rightarrow w$ are different exclusively. We say that a detour D is *reducible* if

(i) there exists a detour $R = [p \rightarrow u \rightarrow v \rightarrow q]$ of a detour $D = [r \rightarrow u \rightarrow v \rightarrow w]$ where p is on $r \rightarrow u$, q is on $v \rightarrow w$, and $L(p \rightarrow u) = L(v \rightarrow q) > 0$, and

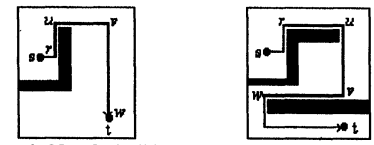
(ii) the vertices of p, u, v, and q make the maximum size of rectangle without intersecting any obstacles on $p \rightarrow q$.

Otherwise, D is a *non-reducible*. Examples of reducible detours are shown in the Figure 2(a). Reducible detours should be reduced prior to the generation of the path $[w \rightarrow \cdots \rightarrow t]$ in the Figure 2(a). The modified paths $[r \rightarrow p \rightarrow q \rightarrow w]$ in Figure 2(a) are reduced detours. Examples of non-reducible detours are shown in Figure 2(b).

The *base node* is generated when an extending line segment l meets one of the following conditions:



a. Reducible Detours $[r \rightarrow u \rightarrow v \rightarrow w]$ and Reduced Detours $[r \rightarrow p \rightarrow q \rightarrow w]$



b. Non-Reducible Detours $[r \rightarrow u \rightarrow v \rightarrow w]$

FIGURE 2 Detours.

(i) l hits an obstacle or border of the graph G,
(ii) l hits a line segment passing through t and s, or
(iii) l passes a corner of obstacle.

The Figure 3 shows example of all possible candidates of base nodes for the given graph.

Algorithm *GMD* (s, t)

// for brevity, "$S\Leftarrow$" and "$S\Rightarrow$" indicate addition to and taking-out from S, respectively //
// $u \rightarrow v$ in *COMPLETE* consists of a 4-tuple (*dir*, *C*, *DL*, *ptr*)//
1 **if** $s = t$ **then** stop;
endif;
2 *NEW*: = null; *OLD* $\Leftarrow s \rightarrow s$; *COMPLETE*: = null; d: = 0; // initializations //
3 **while** *OLD* is not empty **do** // *OLD* contains line segments to be extended //
4 *OLD*$\Rightarrow u \rightarrow v$; // from *OLD*, a line segment $u \rightarrow v$ is taken out //
5 *SEARCH* ($u \rightarrow v$);
endwhile;
6 **if** *NEW* is empty then stop; // no path from s to t exists //
endif;
7 d: = d + 1; // increase d, a lower bound of *DL*, by 1 //
8 *OLD*: = *NEW*; *NEW*: = null;
// when *OLD* becomes empty, all line segments in *NEW* are moved into *OLD*, then *NEW* is reset to empty //
9 **go to**3;
end *GMD*
procedure *SEARCH*($u \rightarrow v$);
1 **if** $DL(u+ \rightarrow v) > d$ **then** *NEW* $\Leftarrow u \rightarrow v$; $DL(u \rightarrow V)$ is a detour length of $P = [s \rightarrow \cdots \rightarrow u \rightarrow v]$ //
2 **elseif** v is a base node **then**
3 *COMPLETE* $\Leftarrow u \rightarrow v$; // no more extensions for $u \rightarrow v$ //
4 **for** each unvisited neighbor node w of v **do**;
5 create a line segment $v \rightarrow w$; // since v is a base node, new line segments from v to four possible directions (north, south, east, and west) are created //
6 **if** w is t **then** stop; // a path from s to t is found //
7 **elseif** $v \rightarrow w$ makes a detour $[r \rightarrow u \rightarrow v \rightarrow w]$ **then**
8 **if** w is an unvisited base node **then** change $v \rightarrow w$ to $v \rightarrow w'$
9 **else** extend $v \rightarrow w$ to $v \rightarrow w'$ until a visited node or an unvisited base node is reached; // use don't change direction //
endif;
10 **if** w' is an unvisited base node **and** $|v \rightarrow w' < |r \rightarrow u|$ **then**
11 $v \rightarrow w$: = *DEL_RD* ($[r \rightarrow u \rightarrow v \rightarrow w']$); // detect a reducible detour, then a new line segment is returned when it is detected by *DEL_RD* //
12 update $DL(v \rightarrow w)$;
13 *SEARCH* ($v \rightarrow w$);
14 **else return**(); // w' is a visited node //
endif;
15 *SEARCH* ($v \rightarrow w$);
endif;
endif;
endfor;

```
16        elseif a neighbor node w of v in direction u → v is unvisited then
17          if w = t then stop; // a path from s to t is found //
18            else extend u → v to u → w; // use don't change direction //
19              SEARCH (u → w);
            endif;
          endif;
        endif;
      endif;
20 return ();
end SEARCH
```

```
procedure DEL_RD ([r → u → v → w']); // deleting reducible detour if exists //
1 emanate an orthogonal line, U, from w' toward r → u until r → u is hit;
                // the line orthogonal U is created from w' toward r → u until r → u is hit //
2 move U toward u → v until no obstacles are intersected;
3 return (U)
end DEL_RD
```

2.2. Guided Minimum Detour (*GMD*) Algorithm

The following procedures are called *Guided Minimum Detour* (*GMD*) Algorithm that find an optimal shortest path using the A^* search [8] with the heuristic "*don't change direction*".

In the *GMD* algorithm, each extended line segment $u \rightarrow v$ in the datat structure *COMPLETE* explained in Section 3 consists of a 4-tuple (*dir*, *C*, *DL*, *ptr*), where

(i) *dir* is the direction of $u \rightarrow v$,
(ii) *C* is coordinates of the two end points of $u \rightarrow v$ such that $\{(x_1, y_1), (x_2, y_2)\}$,

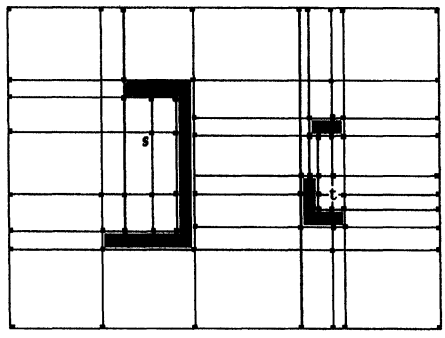


FIGURE 3 Example of possible candidates of base nodes (■).

(iii) $DL(u \rightarrow v)$ is a detour length of the path $P' = [s \rightarrow \cdots \rightarrow u \rightarrow v]$, *i.e.*, $DL(u \rightarrow v) = DL(P')$, and
(iv) *ptr* is a pointer that points a predecessor line segment of $u \rightarrow v$ in *COMPLETE*.

The line segments are extended as follows. Line segments to be extended are always taken one by one from the quene *OLD*. When a line segment $u \rightarrow v$ is taken from *OLD*, the node v is checked as to whether it is a base node or not. If it is a base node, then extensions from v to open directions (north, south, west, and east) are considered. Then, $u \rightarrow v$ is stored in *COMPLETE* and the line segments from v to the open neighbors ($v \rightarrow w$'s) are created. If v is not a base node, the "*don't change direction*" heuristic is enforced by extending $u \rightarrow v$ to $u \rightarrow w$, where w is a neighbor of v in the direction of $u \rightarrow v$. Each line segment is extended to one grid node at a time and controlled by the value of the global detour length d, the lower bound of *DL*. Line extensions from v of $u \rightarrow v$ keep proceeding until a base node or a visited node is hit. When a line segment is extending one node away from the target node t, the detour length (*DL*) of the line segment is increased by 1. Then, if the detour length of the

line segment is greater than d, the line segment is added to a queue *NEW* for the next iteration; otherwise, the line segment continues its extensions. When *OLD* becomes empty, all line segments in *NEW* are moved into *OLD* increasing the lower bound d by 1, then *NEW* is reset to empty.

An important operation that reduces the search space and ensures the shortest path is the elimination of the *reducible detours* defined above. This operation is also applied to the *LGMD* algorithm, which will be explained in Chapter 3. A detour $[r \to u \to v \to w]$ can be easily detected during the search by tracing two segments backward. To detect and delete a reducible detour, the procedure *DEL_RD* in *Guided Minimum Detour* Algorithm is called only when the length of $v \to w'$ (represented by $|v \to w'|$) is less than $|r \to u|$, where w' is the first unvisited base node from w in direction of $v \to w$. The reason *DEL_RD* is called only when $|v \to w'| < |r \to u|$ is as follows. If $|r \to u| \leq |v \to w'|$, then a non-reducible detour can be generated when the path $r \to u \to v \to w'$ is constructed. Let w^* be an intersected point on $v \to w'$ by a perpendicular line segment from r toward $v \to w'$ (see Fig. 4). There are two cases that cause $|r \to u| \leq |v \to w'|$:

(i) No obstacle on $r \to w^*$ (Fig. 4(a)). The path $r \to w^*$ has been generated before $r \to u \to v \to w^*$ is constructed, since $DL(r \to w^*)$ is smaller than $DL([r \to u \to v \to w^*])$.

(ii) Obstacle(s) on $r \to w^*$ (Fig. 4b). The path $[r \to o \to p \to q]$ has been generated before $r \to u \to v \to q]$) is constructed, since $DL([r \to o \to p \to q)$ is smaller than $DL([r \to u \to v \to q])$.

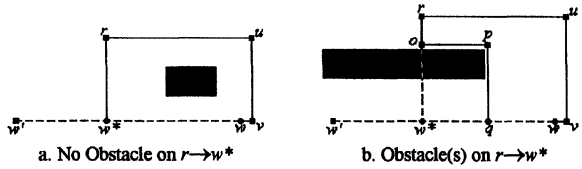


FIGURE 4 No calling the procedure *DEL_RD* for these Detours $[r \to u \to v \to w]$.

When *DEL_RD* is called, a reducible detour is changed to a non-reducible detour. Figure 5 shows two examples solved by the *GMD* algorithm, then the codes for *GMD* algorithm is presented.

2.3. Analysis of the *GMD* Algorithm

For the length of a path from s to t, an obvious lower bound is $M(s, t)$, the *Manhattan distance* from s to t. By Theorem 1, if a path from s to t with length $M(s, t) + 2d$, where d is a lower bound (positive integer), does not exist, then the length of shortest path from s to t is greater than or equal to $M(s, t) + 2(d + 1)$. Our *GMD* algorithm uses the similar principle of the *MD* algorithm [7] so that it searches essential paths, which implies all paths in the *MD* algorithm, of length $M(s, t) + 2d$ before searching for paths of length $M(s, t) + 2(d + 1)$. By Theorem 1, we have the following claim:

THEOREM 2 *The path* $P = [s \to \cdots \to t)$ *generated by the GMD algorithm is an obstacle avoiding shortest path.*

The performance of the *GMD* algorithm can be expected much better than the *MD* algorithm, which is proved by Theorem 3.

THEOREM 3 *The set of searched nodes by the GMD algorithm is a subset of the set of searched nodes by the MD algorithm.*

Proof Let S_{GMD} be the set of the searched space of the *GMD* algorithm and S_{MD} be the set of the searched space of the *MD* algorithm. Let β be a set of base nodes such that $\beta \subseteq S_{GMD}$. Then $\beta \subseteq S_{MD}$, since the detour length of the path from the start node s to a base node in S_{GMD} is minimized. Let g be a node such that $g \in S_{GMD}$, $g \in S_{MD}$, and $g \notin \beta$. Assume there is no obstacle around g. Since g is not a base node, g has only one choice, g', to be extended toward a goal node by the "*don't change direction*" heuristic in *GMD*. However, g has two choices, g' and g'' toward a goal node by the *MD*. Then $g \notin S_{GMD}$ and $g \in S_{MD}$. So, $S_{GMD} \subseteq S_{MD}$.

Now let us analyse the time complexity of the *GMD* algorithm. First, consider the time for node-

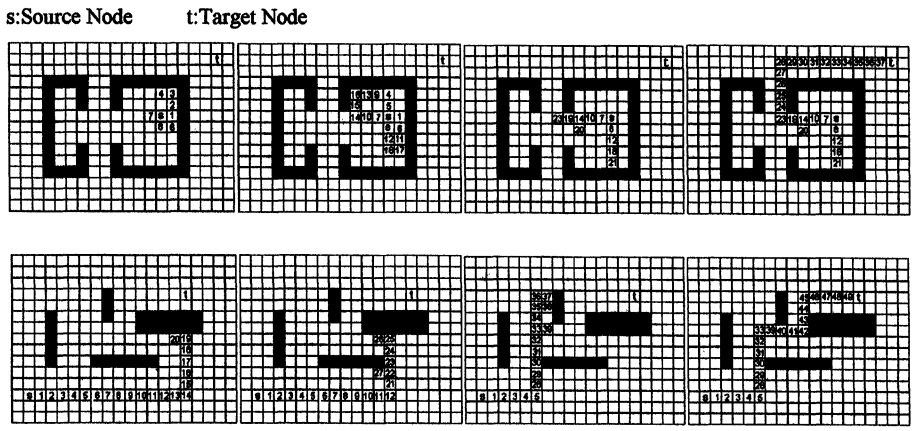


FIGURE 5 Snapshots for the *GMD* algorithm.

by-node extension operations. We define the following basic operations related to the *GMD* algorithm:

All line segments in G (line segments of obstacles, boundaries of the graph, and vertical and horizontal lines through s and t) are stored in the data structure named *CRITICAL*. Then, a base node can be found using *CRITICAL*. The line segments extended during the search are stored in the data structure *COMPLETE*.

(i) Given grid node p with a direction d, find the first base node encountered by a line emanating from p in direction d. We refer to this operation as *finding the first base node*.

(ii) For the interval I from the grid node p to the base node b found in *finding the first base node* operation, check whether a segment in *COMPLETE* is intersected or not. We refer to this operation as *check intersection in COMPLETE*.

(iii) Given grid node p with a direction d, find the first line segment encountered by a line emanating from p in direction d. We refer to this operation as *finding the first obstacle line segment*.

THEOREM 4 [5] *Finding the first base node can be done in $O(\log e)$ time, where e is the total number of line segments in CRITICAL. The data structure of CRITICAL can be built in $O(e \log e)$ time using $O(e)$ space.*

THEOREM 5 [15] *Checking intersection in COMPLETE can be done in $O(\log N)$ time using $O(N)$ space, where N is the number of extended line segments in COMPLETE. Insert operation for storing an extended line segment in COMPLETE can be executed in $O(\log N)$ time.*

THEOREM 6 [5] *Finding the first obstacle line segment can be done in $O(\log e)$ time, where e is the total number of line segments in CRITICAL.*

First, consider the time for node-by-node extension operations. The data structure of *CRITICAL* is a static data structure, which can be constructed in $O(e \log e)$ time and $O(e)$ space as a pre-processing by Theorem 4.

Using the data structure in [5], each operation of *finding the first base node* in *CRITICAL* can be carried out in $O(\log e)$ time by Theorem 4. The searched line segments in the priority search tree [15], named *COMPLETE*, are inserted whenever it is created. Then each operation of *checking intersection in COMPLETE* can be carried out in $O(\log N)$ time, where N is the total number of created line segments. This operation is used for investigating whether a current line segment hits a line segment in *COMPLETE* or not. Let m be the total number of nodes of G visited by grid

expansions, *i.e.*, $m = |S_{GMD}|$. Since there are $O(e)$ base nodes among m nodes and $O(N)$ line segments, then the total time for grid expansions is

$$O(m + e \log e + N \log N). \quad (1)$$

The rest of the computations are associated with the reducible detour detection and deletion operations. There are two related basic operations (i) and (iii) defined above. When *DEL_RD* is called, a reducible detour has to be changed to a non-reducible detour. First, the first base node w' in Figure 6(a) can be found, which takes $O(\log e)$ time by Theorem 4. Second, the line segment $u' \rightarrow v'$ of the non-reducible detour in Figure 6(c) can be found satisfying the following conditions such that:

(i) $u' \rightarrow v'$ is parallel to $u \rightarrow v$,
(ii) $u' \rightarrow v'$ does not intersect any obstacle, and
(iii) the length of $w' \rightarrow v'$ should be minimized.

In example of Figure 6, the nine lines (dotted lines) are generated to find $u' \rightarrow v'$. By the dotted line 1, an end point l is obtained from the hit line segment such that one of its two end points is the closer to the line segment $u \rightarrow v$. Then, the dotted line 2 is emanated from the closest end point l. By the same way, repeatedly, $u' \rightarrow v'$ (dotted line 9) that does not intersect any obstacle is found. If the final emanating line segment, $u' \rightarrow v'$, overlaps $u \rightarrow v$, the detour is not reducible. Otherwise, the reducible detour $[r \rightarrow u \rightarrow v \rightarrow w']$ is reduced to $[r \rightarrow u' \rightarrow v' \rightarrow w']$ as shown in Figure 6(c). Since $O(\log e)$ is required for *finding the first obstacle line segment* by Theorem 6, the time required to reduce a reducible detours is $O(l_t \log e)$ where l_t, is the number of dotted lines in Figure 6. The sum of l_t for all the detours constructed by the *GMD* algorithm cannot exceed $O(e)$ so that the total time required for reducing detours is

$$O(e \log e). \quad (2)$$

Taking into account all the time required for grid extensions(1) and reducing reducible detours(2), the time complexity ((1) + (2)) of the *GMD* algorithm is $O(m + e \log e + N \log N)$. The memory space required is $O(e + N)$. On the basis of above analysis, we have the following claim.

THEOREM 7 [5] *The GMD algorithm can be implemented in* $O(m + e \log e + N \log N)$ *time and* $O(e + N)$ *space, where e is the number of line segments in CRITICAL, m is the total number of visited grid nodes, and N is the total number of searched line segments.*

Figure 7 shows how the same example in [24] is solved using the four variant maze-running algorithms. The size of their expanded nodes is shown in Figure 8. Figure 8 summarizes some experimental results we have conducted with the randomized obstacles in a 30×40 grid graph. Column 2, "*shortest path length*", shows the length of the shortest path for each example. The performances over the *GMD* algorithm is shown in the last column "*Performance* (*times*)". For each algorithm, we give the total number of the expanded nodes and percentage of the searched portion over the total number of nodes respectively.

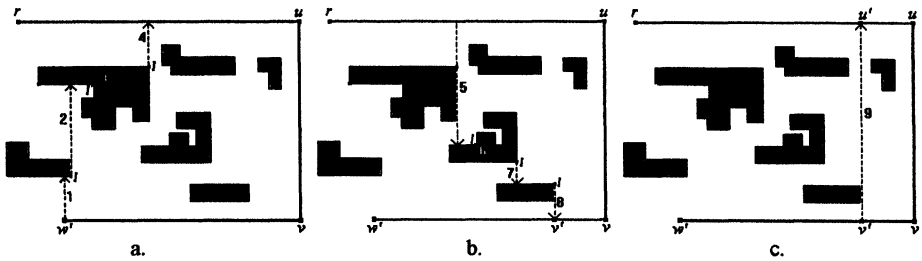
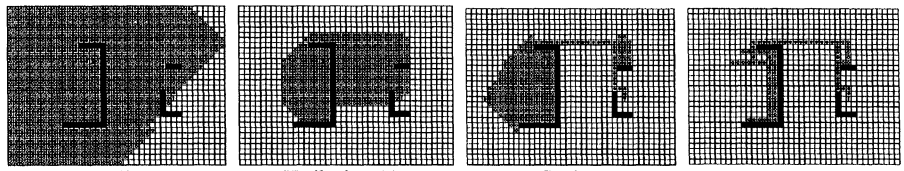


FIGURE 6 Deleting the reducible detour $[r \rightarrow u \rightarrow v \rightarrow w']$ to $[r \rightarrow u' \rightarrow v' \rightarrow w']$.



Lee (917) Hadlock (313) Soukup (215) *GMD* (79)
(# of searched nodes are represented in parentheses)

FIGURE 7 Expanded nodes of the four variants for the example of soukup [24].

*Example	Shortest Path Length	*Lee*		*Hadlock*		*Soukup*		*GMD*		*Performance (times)*		
		# of Searched Nodes	% of Searched Portion	# of Searched Nodes	% of Searched Portion	# of Searched Nodes	% of Searched Portion	# of Searched Nodes	% of Searched Portion	*Lee* / *GMD*	*Hadlock* / *GMD*	*Soukup* / *GMD*
1	35	917	79%	313	27%	215	19%	79	7%	11.3	3.9	2.7
2	36	1060	95%	566	51%	244	21%	53	5%	20.0	10.7	4.2
3	48	1079	96%	700	62%	323	29%	169	15%	6.4	4.1	1.9
4	53	1093	97%	673	60%	573	51%	152	13%	7.5	4.6	3.9
5	54	1067	94%	859	74%	440	39%	202	18%	5.3	4.2	2.2
6	59	1101	96%	774	68%	387	34%	193	17%	5.7	4.0	2.0
7	67	942	83%	679	60%	511	45%	228	20%	4.2	3.0	2.3
8	71	921	84%	680	62%	609	56%	207	19%	4.4	3.3	2.9
9	72	1024	93%	531	48%	404	37%	225	20%	4.7	2.4	1.9
10	74	1070	96%	813	73%	812	73%	185	17%	5.6	4.3	4.3
11	78	1126	95%	823	70%	836	71%	150	13%	7.3	5.4	5.5
12	150	1087	97%	966	86%	881	78%	265	24%	4.0	3.6	3.3
Average	66	1041	92%	698	62%	520	46%	176	16%	**7.2**	**4.5**	**3.1**

***using 30×40 grid graph with randomized obstacles**

FIGURE 8 Comparisons of the experimental results.

3. A MODIFIED ALGORITHM: LINE-BY-LINE GUIDED MINIMUM DETOUR ALGORITHM (*LGMD*)

Let us now consider a modification of the *GMD* algorithm. Now, without losing the general features of the *GMD* algorithm, we contemplate *line-by-line* extensions rather than node-by-node extensions to generate line segments. Each line segment in *COMPLETE* must be from a base node to a base node except the line segments constructed by deleting reducible detour. In other words, a line segment is extended until a base node is hit. A 4-tuple (*dir*, *C*, *DL*, *p*) information (refer to the definition in chapter 2) is assigned to each extended line segment $u \rightarrow v$. The line segment that has the lowest detour length will be chosen for the next extensions. To implement this modification, we use a *priority queue*, called *OPEN*, to select the line segment that has the lowest detour length instead of the queues *OLD* and *NEW* in the *GMD* algorithm. By the queue *OPEN*, the global variable *d*, detour length, in the *GMD* algorithm is not needed. Such a modified algorithm is called the *Line-by-Line Guided Minimum Detour* (*LGMD*) algorithm. The *LGMD* algorithm not only compromises the existing *GMD* algorithm's drawback – the running time – but also shares the solution optimality of the *GMD* algorithm.

Following are the detailed procedures of the *LGMD* algorithm including the above operations. For the same example in Figure 7, the generated whole line segments with sequence numbers and detour lengths (n_1/n_2) by the *LGMD* algorithm are shown in Figure 9.

By an analysis similar to that of the *GMD* algorithm, we conclude the performance of the *LGMD* algorithm by the following theorem.

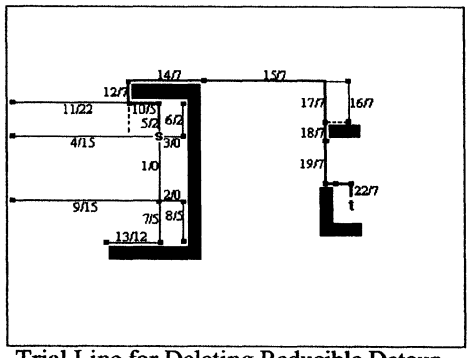
THEOREM 8 *The LGMD algorithm can be implemented* $O(e \log e + N \log N)$ *time and* $O(e + N)$ *space, where e is the number of line segments in CRITICAL and N is the total number of searched line segments.*

4. A COMBINED LENGTH AND BENDS SHORTEST PATH

The objective of this chapter is to develop an efficient combined length and bends shortest path problem using the *LGMD* algorithm shown in Chapter 3. The number of bends on paths gains more attention recently [2, 26]. The current short-

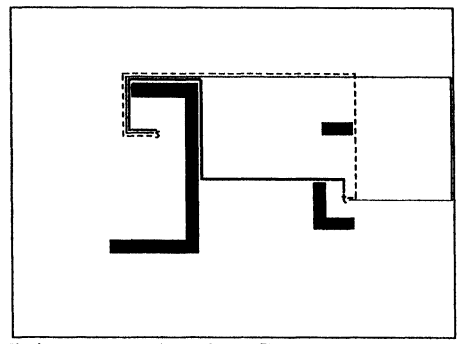
Line-by-Line Guided Minimum Detour (*LGMD*) Algorithm

```
// for brevity, "S⇐" and "S⇐" indicate addition to and taking-out from S, respectively //
// u → v in COMPLETE consists of a 4-tuple (dir, C, DL, ptr)//
algorithm LGMD(s, t);
1  if s = t then stop; endif;
2        OPEN ⇐ s → s; COMPLETE null;
3  while OPEN is not empty do
4        OPEN ⇒ u → v COMPLETE ⇐ u → v;
5        SEARCH (u → v);
   endwhile;
6  stop; // OPEN is empty; no path from s to t exists//
end LGMD
procedure SEARCH_L (u → v);
  // let b be the set of nearest unvisited base nodes from v in all possible directions//
1  for each base node w' in b do;
2    if there is no intersections on v → w' then create a line segment v → w',
3      if w' is t then stop; // a path from s to t is found//
4        elseif v → w' makes a detour [r → u → v → w'] then
5          if L(v → w') < L(r → u) then
6            v → w' := DEL_RD([r → u → v → w']);
7                  update DL(v → w);
8            OPEN ⇐ v → w';
           endif;
9          else OPEN ⇐ v → w';
         endif;
       endif;
     endif;
  endfor;
10 return();
end SEARCH_L
```



--- Trial Line for Deleting Reducible Detour
▪ Base Node
n_1/n_2 Sequence Number of Extensions/Detour Length

FIGURE 9 Extended line segments for the *LGMD* algorithm.



— A Minimum-Bend Path (*LGMD_MB*)
--- A Shortest Minimum-Bend Path (*LGMD_SMB*)
▬ A Minimum-Bend Shortest Path (*LGMD_MBS*)

FIGURE 10 Examples of different shortest paths.

est path algorithms find a shortest path but it leaves the number of bends in the solution path uncertain. Yang *et al.* [26] provide a unified approach by constructing a *path-preserving graph* guaranteed to preserve all these kinds of paths and give an $O(k + e \log e)$ algorithm to find them, where e is the total number of obstacle edges, and k is the number of intersections between *tracks* from *extreme point* and other tracks. k is bounded by $O(ne)$ where n is the number of obstacle. We will consider, specifically, the problems of finding a minimum-bend path, a minimum-bend shortest path, and a shortest minimum-bend path without constructing any track graph. In the dynamic environment like with mobile obstacles, the track graph (*path-preserving*) has to be reconstructed whenever any obstacle is moved. However, the data structure for *LGMD* without track graph needs only a few operations of insertion or deletion for line segments of a moved or changed obstacle. The problems to be considered in this chapter for shortest paths are as follows (refer to Fig. 10):

(i) *LGMD_MB*: a path with a minimum number of bends
(ii) *LGMD_MBS*: a path with a minimum-bend path and shortest length
(iii) *LGMD_SMB*: a shortest path with minimum-bend path

The procedures for the *LGMD_MB* and *LGMD_SMB* are similar to the *LGMD* algorithm in Chapter 3. Let us discuss the *LGMD_MB* algorithm. Each line segment in *COMPLETE* must be from a base node to a base node. For each line segment $u \rightarrow v$ in *COMPLETE*, a 4-tuple (*dir*, *C*, *MB*, *p*) information (refer to the definition in Section 2.1 for *dir*, *C*, and *p*) is assigned to each extended line segment $u \rightarrow v$, where *MB* is a number bends of a path $P = [s \rightarrow \cdots \rightarrow u \rightarrow v]$.

The line segment that has the lowest number of bends will be chosen for the next extensions. We use a *priority queue*, called *OPEN*, to select the line segment that has the lowest *MB* as in the *LGMD* algorithm. Such a modified algorithm is called the *LGMD_MB* algorithm. The difference from the *LGMD* algorithm is that we substitute *DL* to *MB* as a lower bound.

Followings are the detailed procedures of the *LGMD_MB* algorithm. For the same example in Figure 7, the generated whole line segments with generated sequence numbers and *MB* by the *LGMD_MB* algorithm are shown in Figure 10.

LGMD_MB Algorithm
algorithm *LGMD_MB* (*s*, *t*);
// same to the lines 1–6 in **algorithm** *LGMD* (*s*, *t*) described in Section 4 //
end *LGMD_MB*

procedure *SEARCH_MB* ($u \rightarrow v$);
// same to the lines 1 – 6 and 8 – 10 in the **procedure** *SEARCH_L* described in Section 4 //
7 update *MB* ($v \rightarrow w$);
end *SEARCH_MB*

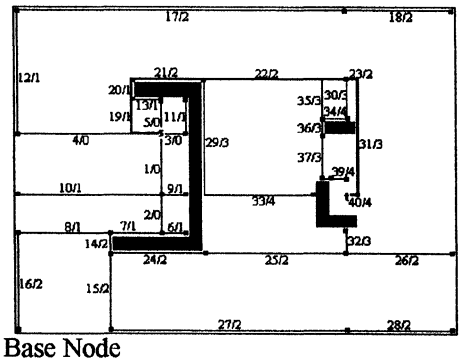
By an analysis similar to that of the *LGMD* algorithm, we conclude the performance of the *LGMD_MB* algorithm by the following theorem.

THEOREM 9 *The LGMD_MB algorithm can be implemented in* $O(e \log e + N \log N)$ *time and* $O(e + N)$ *space, where e is the number of line segments in CRITICAL and N is the total number of searched line segments.*

The Figure 11 shows an example to find a shortest path using *LGMD_MB* algorithm. The bolded line-segments from s to t is the minimum bend path that has the length 40 and four bends, represented by $n_1/n_2 = 40/4$.

The procedures for the *LGMD_MBS* algorithm are same to the *LGMD_MB* algorithm except the lower bound. For each line segment $u \rightarrow v$ in *COMPLETE*, a 5-tuple (*dir*, *C*, *DL*, *MB*, *p*) information is assigned to each extended line segment $u \rightarrow v$. Among the line segments that have the lowest *MB*, a line segment with the lowest *DL* will be chosen for the next extensions.

Similarly, the procedures for the *LGMD_SMB* algorithm can find a shortest path with minimum



▪ Base Node
n_1/n_2 Sequence Number of Extensions/*MB*

FIGURE 11 Extended line segments for the *LGMD_MB* algorithm.

number of bends using a 5-tuple (*dir*, *C*, *DL*, *MB*, *p*) information for each line segment $u \rightarrow v$ in *COMPLETE*. Among the line segments that have the lowest *DL*, a line segment with the lowest *MB* will be chosen for the next extensions.

By an analysis similar to that of the *LGMD_MB* algorithm, the performance of the *LGMD_MBS* algorithm and the *LGMD_SMB* algorithm are concluded by the following theorem.

THEOREM 10 *The LGMD_MBS (or LGMD_SMB) algorithm can be implemented in* $O(e \log e + N \log N)$ *time and* $O(e + N)$ *space, where e is the number of line segments in CRITICAL and N is the total number of searched line segments.*

5. SUMMARY AND CONCLUSIONS

We introduced a heuristic approach to find rectilinear (L_1) shortest path with presence of obstacles. The *GMD* algorithm combines the best features of maze-running algorithms and line-search algorithms. The *LGMD* algorithm is a modification of the *GMD* algorithm that improves on its efficiency. A comparison of the new algorithms with the existing algorithms is presented in Figure 12.

Let us compare the *LGMD* algorithm with the algorithm given by Wu *et al.* [25]. Before the search for a shortest path from s to t starts, the algorithm in [25] constructs a track graph G_T. The space for storing G_T is $O(e + k)$, and the time for constructing G_T and finding a shortest path from s to t is $O((e + k) \log t)$, where e is the total number of boundary sides of obstacles, k is the number of nodes in G_T, and t is the total number of extreme edges in the obstacles (for the definition of extreme edges, refer to [25]). Our *LGMD* algorithm takes $O(e + N)$ space and $O(e \log e + N \log N)$ time. In the worst case, $t = O(e)$, $k = O(e^2)$, and the space and time complexities of the algorithm in [25] are $O(e^2)$ and $O(e^2 \log e)$. The performance of our *LGMD* algorithm depends on N, the total number of searched line segments. Since our *LGMD*

	Lee	*Hadlock*	*Wu et al.*	*GMD*	***LGMD***
Time	$O(n^2)$	$O(n^2)$	$O((e+k)\log t)$	$O(m+ e\log e+N\log N)$	$O(e\log e+N\log N)$
Space	$O(n^2)$	$O(n^2)$	$O(e+k)$	$O(e+N)$	$O(e+N)$
Search Method	Breadth-First	A*	Dijkstra's Search	Grid-by-Grid Guided A*	Line-by-Line Guided A*
Solution Optimality	Optimal	Optimal	Suboptimal	Optimal	Optimal
Connection Graph	Grid Graph	Grid Graph	Track Graph	Grid Graph	Not Needed

FIGURE 12 Bounds on the algorithms discussed in the previous sections.

algorithm does not have a preprocessing phase for generating G_T, the total number N of searched line segments tends to be much smaller than $O(e^2)$. The use of detour length, "*don't change direction*" heuristic, and reducible detour deletion operations is another factor resulting in a small N. Therefore, our *LGMD* algorithm can be expected to outperform the algorithm given in [25].

Since the detour length as a lower bound in our algorithms can be substituted for the number of bends in the rectilinear link metric [2, 11, 26] or the channel wiring density [3], our algorithms can be easily extended to these problems. We described the problem of finding a shortest path in terms of the number of bends and combined length and bends in Section 5.

Our heuristic approach is designed for one-time query. If, however, the repetitive mode is needed in some applications, the heuristic search method in both the *GMD* and the *LGMD* algorithm can be performed on a connection graph for the repetitive-mode queries [27].

References

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

[2] De Berg, M. T., Van Kreveld, M. J., Nilsson, B. J. and Overmars, M. H. (1990). "Finding Shortest Paths in the Presence of Orthogonal Obstacles Using a Combined L_1 and Link Metric", In: *Proceedings of the Second Scandinavian Workshop on Algorithm Theory*, pp. 213–24.

[3] Brown, A. D. and Zwolinski, M. (1990). "Lee Router Modified for Global Routing", *Computer-Aided Design*, **22**, 296–300.

[4] Clarkson, K. L., Kapoor, S. and Vaidya, P. M. (1987). "Rectilinear Shortest Paths Through Polygonal Obstacles in $O(n(\log n)^2)$ Time", In: *Proceedings of the Third Annual Conference on Computational Geometry*, pp. 251–57, ACM.

[5] Edelabrunner, H. and Overmars, M. H. (1984). "Some Methods of Computational Geometry Applied to Computer Graphics", *Computer Vision, Graphics*, and *Image Processing*, **28**, 92–108.

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

[7] Hadlock, F. O. (1977). "The Shortest Path Algorithm for Grid Graphs", *Networks*, **7**, 323–34.

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

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

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

[11] Ke, Y. (1990). "An Efficient Algorithm for Link-Distance Problems", In: *Proceedings of 5th ACM Symp.*, Lect. Notes in Computer Science, **447**, Springer-Verlag, pp. 213–224.

[12] Lee, C. Y. (1961). "An Algorithm for Path Connections and its Applications", *IRE Transactions on Electronic Computers*, **EC-10(3)**, 346–65.

[13] Lengauer, T. (1990). "Combinatorial Algorithms for Integrated Circuit Layout", Wiley, Reading, England.

[14] Lim, J. S., Iyengar, S. S. and Zheng, S.-Q., "Rectilinear Shortest Path Problem with Rectilinear Obstacles" In *Proceedings of the Sixth International Conference on VLSI Design*, pp. 90–93, January 1993.

[15] McCreight, E. M. (1985). "Priority Search Trees", *SIAM Journal of Computers*, **14**(2), 257–276.

[16] Mikami, K. and Tabuchi, K. (1968). "A Computer Program for Optimal Routing of Printed Circuit Connectors", *IFIPS Proceedings*, **H-47**, 1475–78.

[17] Mitchell, J. S. B. (1989). "An Optimal Algorithm for Shortest Rectilinear Paths among Obstacles in the Plane", In: *Abstracts of the First Canadian Conference on Computational Geometry*, p. 22.

[18] Moore, E. F. (1959). "The Shortest Path Through a Maze", *Annals of the Harvard Computation Laboratory*, **30**, Pt.II, 285–92.

[19] Ohtsuki, T. (1986). "Maze-running and Line-search Algorithms", In: Ohtsuki T., Editor, *Advances in CAD for VLSL*, Vol. 4: *Layout Design and Verification*, pp. 99–131, North-Holland, New York.
[20] Pohl, I. (1970). "Heuristic Search Viewed as Path Finding in a Graph", *Artificial Intelligence*, **1**, 193–204.
[21] —— (1971). "Bi-Directional Search", *Machine Intelligence*, **6**, 127–40.
[22] Rezend, P. J., Lee, D. T. and Wu, Y.-F. (1985). "Rectilinear Shortest Paths with Rectangular Barriers", In: *Proceedings of the Second Annual Conference on Computational Geometry*, pp. 204–13, ACM.
[23] Rubin, F. (1974). "The Lee Path Connection Algorithm", *IEEE Transactions on Computers*, **C-23**(9), 907–14.
[24] Soukup, J. (1978). "Fast Maze Router", In: *Proceedings of the 15th Design Automation Conference*, pp. 100–102.
[25] Wu, Y.-F., Widmayer, P., Schlag, M. D. F. and Wong, C. K. (1987). "Rectilinear Shortest Paths and Minimum Spanning Trees in the Presence of Rectilinear Obstacles", *IEEE Transactions on Computers*, **C-36**(3), 321–31.
[26] Yang, C. D., Lee, D. T. and Wong, C. K. (1991). "On Bends and Lengths of Rectilinear Paths: A Graph Theoretic Approach", In: *Proceedings of Algorithms and Data Structures*, 2nd Workshop WADS '91, Lect. Computer Science, Vol. 519, Springer-Verlag, pp. 320–330.
[27] Zheng, S.-Q., Lim, J. S. and Iyengar, S. S. (1993). "Efficient Maze-Running and Line-Search Algorithms for VLSI Layout", In: *Proceedings of the IEEE Southeastcon '93*, Session M4B, IEEE.

Author's Biographies

Joon Shik Lim was born in Seoul, Korea on December 6, 1959. He received B.S. Degree in Computer Science from the Inha University (Korea) in 1986, M.S., and Ph.D. Degree in Computer Science from University of Alabama at Birmingham and Louisiana State University in 1989 and 1994, respectively. Since 1994, he has been an Associate Professor at Department of Computer Science in Kyung Won University in Korea, working on VLSI design, CAD, and speech recognition.

S. S. Iyengar is currently Professor and Chairman of the Department of Computer Science at Louisiana State University. He has published over 220 papers in scientific journals and conference proceedings. He is the author/coauthor of 4 textbooks published by John Wiley, Prentice Hall Inc. and CRC Press Inc. His research is funded by various agencies like NSF, ONR, NASA, US Army research office, DOE and LEQFS. He is Fellow of IEEE, IEEE Distinguished Visitor, NASA Summer Faculty Fellow and member of the NY Academy of Sciences, In 1996, LSU awarded Distinguished Faculty Award of excellence for his research contributions in Image processing.

Si Qing Zheng received the M.S. Degree in Mathematical Sciences from the University of Texas at Dallas in 1982, and the Ph.D. Degree in Computer Science from the University of California, Santa Barbara, in 1987. In August 1987, he joined the faculty of Department of Computer Science, Louisiana State University, where he is currently an Associate Professor of Computer Science and an Adjunct Associate Professor of Electrical and Computer Engineering. Dr. Zheng's research interests include VLSI, computer architectures, parallel and distributed computing, and computer networks.