지능형 최단 경로, 최소 꺾임 경로 및 혼합형 최단 경로 찾기

임 준 식†

요 약

이 논문은 새로운 휴리스틱 탐색(heuristic search) 방법을 이용하여, 수평 및 수직선으로 이루어진 방해물들이 놓인 가운데 수평 및 수직선으로 구성된 최단 거리(*rectilinear shortest path*)와 꺾이는 회수가 가장 적은 최소 꺾임 경로(*link metric shortest path*) 및 이 둘을 혼합시킨 혼합형 최단 경로를 구하는 알고리즘을 서술하고 있다. 최단 경로를 구하는 방법으로 미로 찾기형 알고리즘(*maze-running algorithms*)과 선형 탐색 알고리즘(*line-search algorithms*)의 장점만을 이용한 GMD 알고리즘(*Guided Minimum Detour algorithm*)을 제안하고 있으며 이를 더욱 효율적으로 개선한 LGMD 알고리즘(*Line-by-Line Guided Minimum Detour algorithm*)을 개발하였다. 이들 GMD와 LGMD 알고리즘은 기존의 최단 경로를 내포하고 있는 *conection group*를 이용하지 않고서도 휴리스틱을 사용한 *guided A** 탐색(*guided A* search*)을 이용하여 최적의 최단 경로를 구할 수 있는 장점이 있으며 시간과 메모리 면에서 효율을 극대화하였다. 이들 GMD와 LGMD 알고리즘은 각각 $O(m+eloge+NlogN)$와 $O(eloge+NlogN)$의 시간과 $O(e+N)$의 메모리를 사용한다. 여기서 m은 탐색에 사용된 총 노드의 수이며 e는 장애물(obstacle)들이 갖고 있는 각 변들의 수이다. 또한 N은 탐색에 사용된 직선(line segment)들의 수이다. 또한 *LGMD*는 최소 꺾임 경로(*link metric* shortest path)와 최단 경로와 최소의 꺾임을 조합한 혼합형 최단 경로를 구하는 데에도 적용될 수 있는 확장성을 가지고 있다.

Finding Rectilinear(L_1), Link Metric, and Combined Shortest Paths with an Intelligent Search Method

Joon Shik Lim†

ABSTRACT

This paper presents new heuristic search algorithms for searching rectilinear(L_1), *link metric*, and combined shortest paths in the presence of orthogonal obstacles. The *GMD*(*Guided Minimum Detour*) algorithm combines the best features of maze-running algorithms and line-search algorithms. The *LGMD* (*Line-by-Line Guided Minimum Detour*) algorithm is a modification of the *GMD* algorithm that improves 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 a shortest path. The *GMD* and the *LGMD* algorithms can be implemented in $O(m+eloge+NlogN)$ and O (eloge+NlogN) 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.

1. 서 론

수평 및 수직선으로 구성된 다각형 장애물(obstacle)들이 있는 가운데 최단 경로를 구하는 문제는 로보트학(*robotics*), *VLSI* 디자인, 그리고 지리 정보 시스템(*geographical information systems*)에서 많이 응용되고 있다[13]. 특히 *VLSI* 디자인 부분에서 미로 찾기형 알고리즘(*maze-*

† 정 회 원 : 경원대학교 전자계산학과 전임강사
논문접수 : 1995년 9월28일, 심사완료 : 1995년 12월13일

running algorithms)과 선형 탐색 알고리즘(*line-search algorithms*)이라는 두 가지의 기본적인 알고리즘들이 사용되고 있으며 통상 장애물(obstacle)을 피한 두 점 간의 최단 경로를 찾아 낸다. 미로 찾기형 알고리즘은 목적 지점을 향하여 바둑판 위에 한 점씩 두듯이 찾아가는 방법이다. 이 방식은 *Lee*[12]에 의해 최초로 소개되어 Lee algorithm 이라고 부르는데 이는 인공 지능의 탐색 방법의 하나인 *breadth-first* 탐색 방법을 응용한 것이다. 그러나 Lee algorithm은 $n \times n$의 바둑판 모양의 그래프(*grid graph*)에서 $O(n^2)$의 메모리와 시간을 요하는 결정적인 단점을 가지고 있다. 이와 같은 바둑판 모양의 그래프(*grid graph*)를 이용한 최단 경로 찾기는 여러 가지 방법으로 변형되어 발표되었다[1, 6, 7, 8, 10, 13, 14, 17, 18, 19, 20, 22, 23].

특히 Hart et al.[8]는 시작점과 목적점 간의 최저 임계점(lower bound)으로써 the *Manhattan distance*의 사용을 제안하였으며 이를 Hadlock이 응용하여 *Minimum Detour algorithm*[7]을 발표하였다. 그는 각 노드(grid node)에 대해 *detour number*라는 수를 붙였는데 이는 현재의 찾아가고 있는 경로가 목적점에 대해 반대 방향으로 가고 있는 점들의 총 개수를 의미한다.

또한 Soukup[23]은 depth-first 탐색 방법과 *breadth-first* 탐색 방법을 함께 사용함으로써 메모리와 시간을 대폭 줄였지만 최적화된 경로를 구하지 못하는 단점을 갖고 있다.

미로 찾기형 알고리즘(maze-running algorithms)에 의해 탐색된 각 각의 노드(grid node)들은 자신의 정보를 갖고 있어야 하므로 메모리와 실행 시간에 대해 매우 비효율적이다. 이를 해결하기 위해 선형 탐색 알고리즘(line-search algorithms)[9, 15]이 제안되었는데 각각의 노드(grid node)들 대신 직선을 통해 탐색하기 때문에 메모리와 실행 시간을 효율적으로 줄일 수가 있다. 반면에 이들은 통한 최적화된 최단 경로를 구하지 못하는 단점을 지니고 있다. 이어 여러가지 변형된 computational geometry 기법을 이용한 선형 탐색 알고리즘[4, 13, 16, 21, 24]이 발표되었으며 이들 중 현재 Wu et al.[24]의 방법이 가장 강력한 방법으로 알려져 있다. Wu는 *track graph* 라는 최단 경로를 포함하는 그래프를 먼저 작성하였는데 이를 통해 두 점 간의 최단 경로를 찾아 내고 있다. 그러나 그 *track graph*는 최적화된 최단 경로를 보장하지는 못한다. Wu 알고리즘의 실행 시간은 $O((e+k)\log t)$으로 표현되는데 여기서 e는 장애물들의 변들의 합이며 t는 장애물들의 최 외곽에 있는 변들의 합이며, 또한 k는 *track graph*에 있는 교차점들의 합이다.

Zheng et al.[26]은 효율적인 computational geometry 기법을 이용하여 connection graph를 만들어 최단 경로를 구하였고, De Rezende et al.[21]은 장애물이 모두 n개의 직사각형인 경우에 한하여 최단 경로를 $O(n\log n)$의 실행 시간안에 구하였다. Clarkson et al.[4]은 임의로 된 장애들에 대해 일반화시킨 알고리즘을 발표하였다. 더 자세한 내용은 T. Lengaure[13]의 저서에 상술되었다.

본 논문에서 새로운 휴리스틱 탐색 방법을 이용한 *Guided Minimum Detour*(*GMD*) 알고리즘과 *Line-by-Line Guided Minimum Detour*(*LGMD*) 알고리즘을 제안하였다. *GMD* 알고리즘은 미로 찾기형 알고리즘(maze-running algorithms)과 선형 탐색 알고리즘(line-search algorithms)의 장점들만을 선택하여 최적화된 최단 경로를 찾게 되며 이를 위해 새로운 탐색 방법인 *guided* A^* 탐색을 사용한다.

이러한 *GMD* 알고리즘을 다시 응용하여 더욱 효율적인 *LGMD* 알고리즘을 제안하였다. *LGMD* 알고리즘은 Wu가 사용한 *track graph*를 사용하지 않았을 뿐만 아니라 최적화된 최단 경로를 제공하며 현존의 어느 알고리즘에 비해 메모리와 실행 시간을 줄인 알고리즘이다.

또한 *LGMD* 알고리즘을 변형한 혼합형 최단 경로 알고리즘들을 제시하여 꺾임의 수를 최단 경로에 고려하는 알고리즘들을 제안하였다.

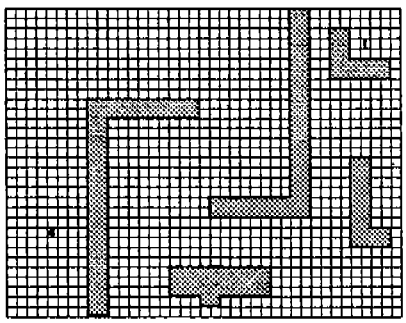
2. 새로운 알고리즘 : *GUIDED MINIMUM DETOUR* 알고리즘(*GMD*)

2.1 정의와 구현

G를 $n \times n$ 바둑판 모양의 그래프(grid graph)

라 하고 각 각의 바둑점이 놓이는 부분을 노드(*grid nodes*)라고 한다면 G는 노드들의 집합인 $\{(x, y) \mid x \text{ and } y \text{ are integers such that } 1 \le x \le n \text{ and } 1 \le y \le n\}$ (그림 1 참조)로 표현될 수 있다. 여기서 이웃 노드들 간의 거리는 1이라 하자. 또 수평선(혹은 수직선)은 일렬로 된 노드들로 구성되어 있다. $B = \{B_1, B_2, \cdots, B_b\}$는 G 위에 놓여진 수평 및 수직선으로 구성된 다각형 모양인 서로 떨어진 장애물들의 집합이다.

그래프 G 상에서 방향을 가진 경로 P는 노드의 집합인 $\{v_1, v_2, \cdots, v_m\}$와 수평 혹은 수직선인 선분(*line segment*)의 집합인 $\{(v_i \rightarrow v_{i+1}) : i = 1, \cdots, m-1\}$을 이용하여 $P(v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_m)$으로 나타낸다. 이 때 v_1과 v_2가 이웃 한다면 선분 $v_1 \rightarrow v_2$는 길이가 1인 단위 선분이라 한다. 또한 경로 P의 길이는 $L(P)$로 표현한다. 또한 경로 P의 우회 길이(*detour length*)는 $DL(P)$로 나타내는데 이는 어느 경로 P 중에서 목적점 t로부터 멀어지는 노드들의 총 개수를 말한다. $DL[u \rightarrow v]$는 어느 경로를 $P = (s \rightarrow \cdots \rightarrow u \rightarrow v \rightarrow \cdots \rightarrow t)$라 할 때 P의 시작점인 s로부터 v 노드까지의 우회 길이를 의미한다. 이 때 $P' = (s \rightarrow \cdots \rightarrow u \rightarrow v)$라 한다면 $DL[u \rightarrow v] = DL(P')$가 된다. $M(s, t)$는 두 노드 s와 t 사이의 *Manhattan Distance*를 의미하며 경로 P의 길이는 $L(P) = M(s, t) + 2 \times DL(P)$로 표현되며 만일 $DL(P)$가 최소화 된다면 그 경로 P는 최단 경로이다. 다음의 정리[7]를 통해 이를 다시 표현하고자 한다.



(그림 1) 바둑판 모양의 그래프 G와 장애물 B
(Fig. 1) Definitions of the Grid Graphs

정리 1[7].

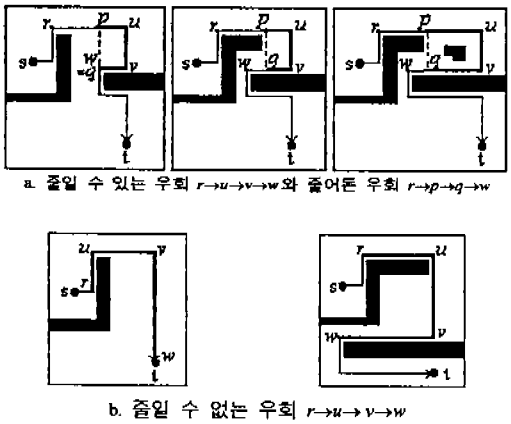
1. 경로 $P = (s \rightarrow \cdots \rightarrow t)$는 $L(P) = M(s, t) + 2 \times DL(P)$의 길이를 가진다.
2. 만일 P가 두 노드 s와 t사이의 최단 경로라면, $DL(P) = \min\{DL(P') \mid P' \text{ is a set of all paths from } s \text{ to } t\}$.
3. *minimum detour* 알고리즘[7]에 의해 찾은 경로는 $DL(P)$가 최소화 되었기에 최단 경로이다.

만일 어느 경로 P가 연속된 세 개의 선분(line segment) $D = (r \rightarrow u \rightarrow v \rightarrow w)$를 갖고 세 선분(line segment) $r \rightarrow u$, $u \rightarrow v$ 및 $v \rightarrow w$의 방향(direction)이 각기 서로 다르다면 D를 우회(*dictour*)(그림 2.1와 2.b 참조)라고 한다. 또한 우회(detour) D가 다음을 만족할 때 그것을 줄일 수 있는 우회(*reducible detour*)라고 하고 R이라 쓴다.

(i) $D = (r \rightarrow u \rightarrow v \rightarrow w)$의 한 부분인 $R = (p \rightarrow u \rightarrow v \rightarrow q)$이 존재하고 p가 $r \rightarrow u$ 상에 있고, q가 $v \rightarrow w$ 상에 있으며 $L(p \rightarrow u) = L(v \rightarrow q) > 0$을 만족하고,

(ii) $R = (p \rightarrow u \rightarrow v \rightarrow q)$이 우회(detour)이어야 하며,

(iii) $R = (p \rightarrow u \rightarrow v \rightarrow q)$이 를 장애물(obstacle) 없이 이었을 때 가능한 한 가장 넓은 직사각형을 이룬다.



(그림 2) 우회
(Fig. 2) Detours

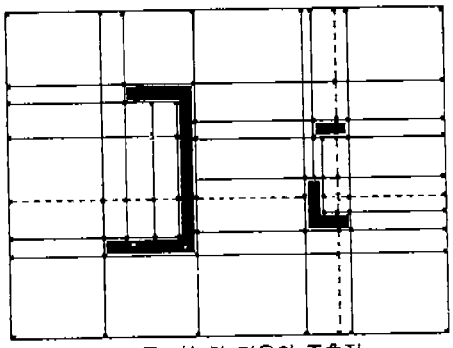
줄일 수 있는 우회(reducible detour)의 예는 (그림 2.a)의 $r \rightarrow u \rightarrow v \rightarrow w$로 나타나는데 반드시 이를 줄일 수 없는 우회로 바꾸어야 하며 줄어든 우회는 (그림 2.a)에서 $r \rightarrow p \rightarrow q \rightarrow w$로 나타난다. 또한 줄일 수 없는 우회의 예는 (그림 2.b)에서 보여 준다.

만일 현재 만들어지는 선분(line segment) l이 다음 조건을 만족하는 점 b를 만난다면 그것을 주춧점(*base node*)라 한다.

(i) l이 그래프 G의 외곽선을 만나거나,
(ii) l이 장애물의 꺾인 부분을 만나거나, 혹은
(iii) l이 목적점 t를 지나는 수직 혹은 수평선 (그림 3의 점선)을 만날 때

또한 시작점은 특별한 경우의 주춧점(base node). (그림 3)은 모든 가능한 경우의 주춧점을 보여준다.

GMD 알고리즘은 *Minimum Detour*(*MD*)[7] 알고리즘과 A^* 탐색 방법을 사용한다는 점에서 유사하다. 두 알고리즘은 최단 경로가 존재한다면 항상 그것을 찾아내는 특성이 있다. 그러나 *GMD* 알고리즘은 *MD* 알고리즘과 달리 선형 탐색 알고리즘(line-search algorithms)을 함께 사용한다는 점이 다르며 이는 메모리를 선형 탐색 알고리즘의 수준만큼 줄이는 효과를 준다. 이는 *GMD* 알고리즘이 탐색한 경로를 *MD* 알고리즘의 노드(grid nodes) 대신 "곧 바로"("*don't change direction*")라는 휴리스틱을 사용하여 만들어 내는 선분(line segments)들의 집합으로 나타내 주기 때문이다. 각 선분은 네가지 요소로



•: 모든 가능한 경우의 주춧점

(그림 3) 주춧점
(Fig. 3) Base Nodes

구성된(dir, C, DL, ptr)로 구성된 *COMPLETE*라는 자료 구조에 저장되는데, 여기서

(i) dir은 $u \rightarrow v$의 방향이고
(ii) C는 $u \rightarrow v$의 두 끝 점인 u와 v의 x, y 좌표이며,
(iii) DL는 우회 길이(detour length)이며
(iv) ptr은 $u \rightarrow v$의 바로 전 선분을 가리키는 포인터이다.

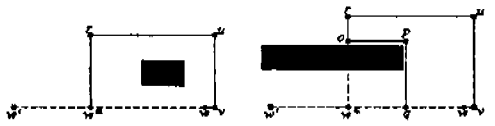
각 선분은 다음과 같이 뻗어 나가게 된다. 어느 선분 $u \rightarrow v$는 *OLD*라는 queue로부터 하나씩 나오게 되는데 이 때 노드 v가 주춧점(base node)인지를 확인하게 된다. 만일 그것이 주춧점이라면 v로부터 가능한 모든 방향으로 새로운 선분이 만들어 질 수 있다. 동시에 $u \rightarrow v$는 *COMPLETE*라는 자료 구조에 저장된다. 반면 v가 주춧점(base node)이 아니라면 "곧 바로"("*don't change direction*")라는 휴리스틱에 의해서 $u \rightarrow v$는 $u \rightarrow w$로 뻗어 나게 되는데 이 때 우회 길이(detour length)의 최저 임계치인 d를 넘지 않는 한 이미 탐색에 사용된 노드(visited node)나 주춧점을 만날 때까지 계속 나아 간다. 최저 임계치인 d와 비교되는 것은 우회 길이(detour length)를 나타내는 DL인데 DL은 어느 선분이 한 노드 만큼 뻗어 나갈 때 목적점인 t로부터 멀어지게 되면 1씩 증가하며 DL이 d를 넘게 되면 *NEW*라는 queue로 다음 단계의 확장을 위해 저장된다. 한편 *OLD*가 비게 되면 *NEW*에 저장된 모든 선분들은 *OLD*로 옮겨지며 최저 임계치 d는 1이 증가되며 *NEW*는 빈 상태가 된다. 여기서 *GMD* 알고리즘에서 찾은 경로가 최단 경로임과 탐색에 필요한 노드의 수를 줄이는 데에 있어 중요한 과정은 바로 줄일 수 있는 우회(reducible detours)를 모두 없애는 것이다. 이 과정은 앞으로 설명될 *LGMD* 알고리즘에도 적용되는데 제4장에서 자세히 다룬다. 어느 우회(detour) $r \rightarrow u \rightarrow v \rightarrow w$는 탐색 중에 이번 장처을 부분에 설명한 대로 쉽게 발견된다. 우회를 발견하고 그것이 줄일 수 있는 우회(reducible detour)일 경우 줄일 수 없는 우회(non-reducible detour)로 만드는 작업은 다음에 나오는 procedure *DLE-RD*에서 수행되게 된다. 여기서 효율을 높이기 위해 단지 (그림 4)에서 $v \rightarrow w'$ 의

길이($|v\rightarrow w'|$)가 $|r\rightarrow u|$보다 짧을 때에만 이 procedure를 부르게 되는데(여기서 w'는 w로부터 $v\rightarrow w$의 방향으로 나아갈 때 발견되는 첫 번째 탐색에 사용되지 않은 주춧점임) 그 이유는 $|r\rightarrow u|\leq|v\rightarrow w'|$일 경우 어느 다른 경로가 이미 그 길을 선점하기 때문이다. 여기서 w^*를 $v\rightarrow w'$와 r로부터 $v\rightarrow w'$를 수직으로 향한 선과의 교차점이라고 하자(그림 4 참조). 이때 $|r\rightarrow u|\leq|v\rightarrow w'|$를 만족하는 두 가지의 경우가 생기는 데 :

(i) $r\rightarrow w^*$선 상에 장애물이 없는 경우(그림 4.a) : $DL(r\rightarrow w^*)<DL(r\rightarrow u\rightarrow v\rightarrow w^*)$이기 때문에 경로 $r\rightarrow w^*$이 먼저 생긴다.

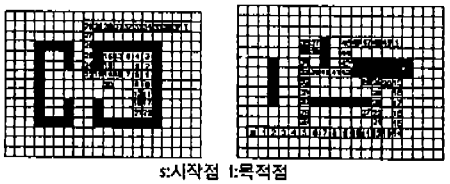
(ii) $r\rightarrow w^*$선 상에 장애물이 있는 경우(그림 4.b) : $DL(r\rightarrow o\rightarrow p\rightarrow q)<DL(r\rightarrow u\rightarrow v\rightarrow q)$이기 때문에 경로 $r\rightarrow o\rightarrow p\rightarrow q$이 먼저 생긴다.

*DEL_RD*이 불리우게 되면 줄일 수 있는 우회(reducible detour)일 경우 줄일 수 없는 우회(non-reducible detour)로 바뀌게 된다. (그림 5)는 *GMD* 알고리즘을 이용하여 최단 경로를 찾는 순서를 각 노드 안의 숫자를 통해 보여 주고 있으며 이어서 *GMD* 알고리즘의 코드를 기술하였다.



a.$r\rightarrow w^*$ 선 상에 장애물이 없는 경우 b.$r\rightarrow w^*$ 선 상에 장애물 이 있는 경우

(그림 4) *DEL_RD* Procedure를 부르지 않는 경우
(Fig. 4) No Calling the Procedure *DEL_RD*



s:시작점 t:목적점

(그림 5) *GMD*알고리즘을 이용하여 찾은 최단 경로의 예
(Fig. 5) Examples for the *GMD* Algorithm

GMD (Guided Minimum Detour) 알고리즘

```
algorithm GMD (s,t);
// "S<=" and "S=>" indicate addition to and taking-out from S, respectively //
1   if s = t then stop;
    endif;
2   NEW<=null; OLD<=s→s; COMPLETE<=null; d:=0;   // initializations //
3   while OLD is not empty do // OLD contains line segments to be extended //
4       OLD=>u→v;   // from OLD, a line segment   u→v is taken out //
5       SEARCH (u→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→v);
1  if DL[u→v]>d then NEW<=u→v;
     // DL[u→v] is a detour length of P=(s→...→u→v) //
2    elseif v is a base node then
3      COMPLETE<=u→v; // no more extensions for u→v //
4      for each unvisited neighbor node w of v do;
5        create a line segment v→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→w makes a detour r→u→v→w then
8            if w is an unvisited base node then change v→w to v→w';
9              else extend v→w to v→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→w'|<|r→u| then
11               v→w := DEL_RD (r→u→v→w');
                 // detect a reducible detour, then a new   line segment   is
                    returned when it is detected by DEL_RD //
12               update DL[v→w];
13               SEARCH (v→w);
14               else return(); // w' is a visited node //
               endif;
           endif;
         endif;
       endfor;
15     elseif a neighbor node w of v in direction u→v is unvisited then
16       if w = t then stop; // a path from s to t is found //
17         else extend u→v to u→w; // use don't change direction //
18           SEARCH (u→w);
         endif;
       endif;
     endif;
   endif;
19 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;
2    move U toward u→v until no obstacles is intersected;
3    return (U)
end DEL_RD
```

2.2 *GMD* 알고리즘의 분석

s로부터 t까지의 어떤 경로에 있어서 그 길이의 최저치는 분명히 *Manhattan distance*이며 $M(s, t)$로 표기된다. 만일 최저 임계치인 $d(d\geq 0)$를 가진 길이 $M(s, t)+2d$인 경로가 최단 경로를 찾지 못한다면 최단 경로는 $M(s, t)+2(d+$

1) 이상일 것이다. *GMD* 알고리즘은 *MD* 알고리즘[7]이 사용한 최저 임계치인 d를 사용하면서, 탐색 중 만들어진 선분은 *MD* 알고리즘으로 탐색한 경로 중에서 특수한 경우에 해당되므로 다음과 같은 정리 2와 3을 가질 수 있다.

정리 2. *GMD* 알고리즘으로 탐색된 노드의 집합은 *MD* 알고리즘으로 탐색한 노드 집합의 부분집합니다.

정리 3. *GMD* 알고리즘으로 만들어진 경로 $P=(s\rightarrow\cdots\rightarrow t)$는 장애물을 피해 가는 최단 경로이다.

지금부터 *GMD* 알고리즘의 실행 시간을 분석하기로 한다. 우선 한 노드씩 진행하는 과정을 살펴 보기 위해 다음의 몇 가지 기본 과정을 정의하고자 한다. 우선 그래프 G 위에 있는 장애물과 G의 외곽 변, 그리고 목적점 t를 지나는 수직 혹은 수평선(그림 3의 점선) 등 모든 선분을 *CRITICAL*이라는 자료 구조 속에 넣는다. 이 *CRITICAL*을 이용하여 탐색 중 주춧점을 찾을 수 있다. 또한 모든 탐색된 선분(line segment)는 *COMPLETE*라는 자료 구조 속에 넣는다.

(i) 주춧점 찾기(*finding the frst base node*) : 주어진 어떤 d라는 방향을 가진 노드 p가 d의 방향으로 처음 만나게 되는 주춧점을 찾는 것.

(ii) *COMPLETE* 내에서 교차점 확인(*checking intersection in COMPLETE*) : (i)에서 찾은 주춧점을 b라 할 때 선분 $p\rightarrow b$와 *COMPLETE* 내에 있는 선분들과 교차점이 있는 지를 확인하는 것.

(iii) 최초의 장애물 찾기(*finding the first obstacle line segment*) : 주어진 어떤 d라는 방향을 가진 노드 p가 d의 방향으로 처음 만나게 되는 장애물을 찾는 것.

정리 4[5]. 주춧점 찾기(*Finding the first base node*)는 $O(\log e)$ 시간 안에 수행되며, 여기서 e는 *CRITICAL* 내에 있는 모든 선분의 수이다. 또, *CRITICAL*이란 자료 구조는 $O(e\log e)$의 시간과 $O(e)$의 메모리로 만들어 진다.

정리 6[5]. 최초의 장애물 찾기(*Finding the first obstacle line segment*)는 $O(\log e)$ 시간 안에 수행되며, 여기서 e는 *CRITICAL* 내에 있는 모든 선분의 수이다.

우선 pre-processing 작업으로써 *CRITICAL*이 만들어지는데 *CRITICAL*의 자료 구조는 Edelsbrunner[5]에 의해 $O(e\log e)$의 시간과 $O(e)$의 메모리로 구현된다. 정리 4에 의해 주춧점 찾기(*finding the first base node*)는 $O(e\log e)$ 시간 안에 수행되며 완성된 선분(line segment)은 *priority search tree*[27]라는 자료 구조를 가진 *COMPLETE*에 저장되며 이는 $O(\log N)$의 시간을 요한다. 또한 그 선분이 *COMPLETE* 내에 있는 선분들과 교차하는 지는 정리 5의 *COMPLETE* 내에서 교차점 확인(*Checking intersection in COMPLETE*)에 의해 $O(\log N)$ 시간안에 수행된다. m을 탐색에 사용된 총 노드의 수라 한다면 $m=|S_{CMD}|$로 표현하도록 하자. 탐색에 사용된 노드는 $O(e)$ 개의 주춧점을 가지며 $O(N)$ 개의 선분들을 구성하므로 총 탐색 시간은 다음과 같이 나타낸다.

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

나머지 수행 시간은 줄일 수 있는 우회(reducible detour)를 찾고 그것을 줄일 수 없는 우회(non-reducible detour)로 만들어 주는데 걸리는 시간이다. 여기에는 위에서 정의한 (i)과 (iii)의 방법이 사용된다. 우선 우회가 발견되면 (그림 6)에서 보여 주듯이 v로부터 첫 번째 주춧점인 w'를 찾게 되는데 이는 정리 4에 언급한 대로 $O(\log e)$의 시간이 소요된다. 다음 (그림 6.c)에 있는 선분 $u'\rightarrow v'$을 찾아야 하며 이때 $r\rightarrow u'\rightarrow v'\rightarrow w'$는 줄일 수 없는 우회(non-reducible detour)가 되며 선분 $u'\rightarrow v'$는 다음 조건을 만족하여야 한다.

(i) $u'\rightarrow v'$는 $u\rightarrow v$와 평행하며,

(ii) $u'\rightarrow v'$는 어느 장애물과도 충돌이 없으며, 그리고,

(iii) $w'\rightarrow v'$의 길이는 최소화되어야 한다.

다음 예(그림 6)에서는, 점선으로 된 9개의 선분이 $u'\rightarrow v'$를 발견하기 위해 사용되었다.

t를 (그림 6)에서와 같이 어떤 줄일 수 있는

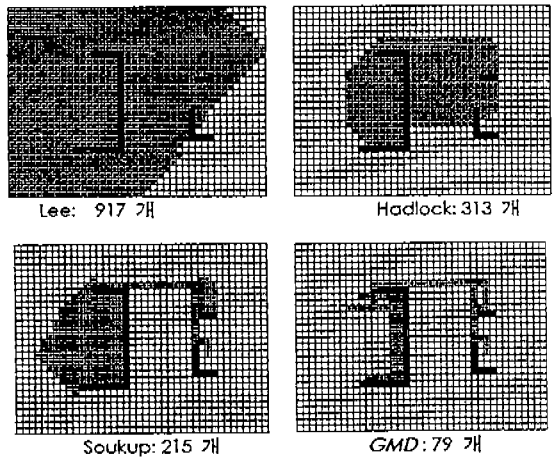
우회에 대한 점선의 총 개수라고 하면 정리 6에 의해 최초의 장애물 찾기(finding the first obstacle line segment)는 $O(\log e)$의 시간을 요하므로 줄일 수 없는 우회로 만드는 시간은 $O(t\log e)$이다. 따라서 전체 그래프에서 t값들의 총 합은 $O(e)$를 초과할 수 없으므로 줄일 수 없는 우회를 만드는 시간은 전체적으로 다음으로 나타낸다.

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

(그림 6) 줄일 수 있는 우회 없애기
($r\rightarrow u\rightarrow v\rightarrow w'$ 를 $r\rightarrow u'\rightarrow v'\rightarrow w'$ 로)
(Fig. 6) Deleting the Reducible Detour
($r\rightarrow u\rightarrow v\rightarrow w'$ to $r\rightarrow u'\rightarrow v'\rightarrow w'$)

위에서 제시한 GMD 알고리즘의 총 탐색 시간은 모든 노드를 확장하는데 걸리는 시간인 (1)에서의 $O(m+e\log e+N\log N)$과 줄일 수 없는 우회를 만드는 시간인 (2)에서의 $O(e\log e)$의 합인 $O(m+e\log e+N\log N)$으로 나타내며 $O(e+N)$의 메모리를 요한다. 이와 같은 사실에 근거하여 다음의 정리를 갖게 된다.

정리 7. GMD 알고리즘은 $O(m+e\log e+N\log N)$의 시간과 $O(e+N)$의 메모리로 구현된다. 여기서 e는 $CRITICAL$ 내에 있는 모든 선분의 수, m은 탐색에 사용된 총 노드의 수, 그리고 N은 $COMPLETE$ 내에 있는 모든 선분의



(그림 7) 탐색에 사용된 총 노드의 수
(Fig. 7) Expanded Nodes of the Four Variants for the Example of Soukup[23]

*Example	Shortest Path Length	*Lee*[12]		*Hadlock*[7]		*Soukup*[23]		*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**

* 30×40그래프 상에 임의의 장애물로 만든 예

(그림 8) 실험 결과 비교 분석
(Fig. 8) Comparisons of the Experimental Results

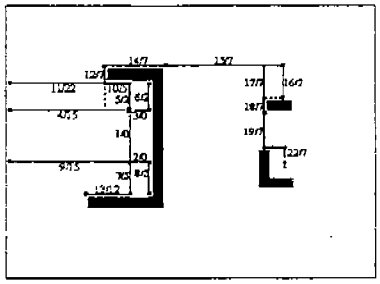
수이다.

(그림 7)은 *Soukup*[23]이 사용한 예를 지금까지 대표적으로 알려진 세 개의 미로 찾기형 알고리즘(maze-running algorithms)과 *GMD* 알고리즘으로 탐색한 결과이다. 여기서 *GMD*의 경우 가장 적은 노드의 수를 사용하여 목점덤을 찾았다. (그림 8)은 30×40 크기의 그래프 위에 임의의 장애물을 만들어 탐색을 수행한 실험을 정리한 것이다. 2번째 열 "*shortest path length*"는 각 보기들에 대한 최단 경로의 길이를 나타내며, 다음 열 "# *of Searched Nodes*"은 탐색에 사용된 총 노드의 수, 그리고 "*% of Searched Portion*"은 탐색 전의 그래프 공간에 대해서 탐색에 사용된 공간에 대한 비율을 나타낸다. 또한 *GMd* 알고리즘의 다른 알고리즘에 대한 효율은 마지막 열인 "*Performance(times)*"에서 보여 주고 있다. 이 결과에서 *GMD*는 *Lee* 알고리즘에 비해 7.2배의 효율을 나타냈으며 *Hadlock*이나 *Soukup*의 방법에 비해서는 각 각 4.5배와 3.1배의 효율을 나타내었다.

3. 개선된 알고리즘 : *LINE-BY-LINE GUIDED MINIMUM DETOUR*트 알고리즘(*LGMD*)

지금부터 *GMD* 알고리즘의 개선을 꾀하고자 한다. *GMD* 알고리즘의 일반적인 특성을 유지하면서 한 노드씩 탐색해 나가는 대신 한 선분씩 확장하여 실행 시간을 줄이고자 한다. 일반적으로 *COMPLETE* 안에 저장되는 선분은 반드시 주춧점으로 시작하여 주춧점으로 끝나게 된다. 이 때 각 선분은 4가지 요소로 구성된(*dir*, *C*, *DL*, *p*)라는 정보를 갖게 된다(2장 참조). 다음에 확장될 선분을 선택하기 위해서 *GMD*의 *OLD*나 *NEW* 안에 있는 선분은 *OPEN*이라는 *priority queue*를 사용하게 된다. *OPEN*에는 우회길이를 나타내는 *DL*에 의해 저장이 되는데 *priority queue*의 특성상 가장 그 값이 작은 선분이 가장 먼저 다음에 확장될 선분으로 채택된다. 따라서 *GMD*에서 사용된 최저 임계치 *d*는 더 이상 필요가 없게 된다. 바로 이 알고리즘을 *LGMD*(*Line-by-Line Guided Minimum Detour*) 알고리즘이라 한다. *LGMD*는 *GMD*의 결점인 긴 실행 시간을 줄여 줄 뿐만 아니라 *GMD*의 장점인 최적화를 동시에 갖고 있는 점이 특징이다.

다음에 기술된 *LGMD* 알고리즘은 위에서 설명된 동작을 상술하고 있으며, (그림 9)는 (그림 7)의 예제를 *LGMD*를 사용하여 최단 경로



----: 줄일 수 있는 우회를 없애기 위해 사용된 선분
■: 주춧점
n_1/n_2: 각 선분이 만들어진 순서/우회 길이

(그림 9) *LGMD* 알고리즘을 사용한 최단 경로
(Fig. 9) Extended Line Segments for the *LGMD* Algorithm

	Lee[12]	*Hadlock*[7]	*Wu et al.*[24]	*GMD*	*LGMD*
Time	$O(n^2)$	$O(n^2)$	$O((e+k)\log e)$	$O(m+e\log e+M\log N)$	$O(e\log e+M\log N)$
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
Graph	Grid Graph	Grid Graph	Track Graph	Grid Graph	Gridless Graph

(그림 10) 최단 경로 알고리즘들의 비교
(Fig. 10) Bounds on the Algorithms

를 찾은 것이다. 각 선분 위에 첨자된 두 숫자 n_1/n_2은 각 선분이 만들어진 순서와 우회 길이 (DL)를 나타낸다.

LGMD (Line-by-Line Guided Minimum Detour) 알고리즘

```
// for brevity, "S⇐" and "S⇒" indicate addition to and taking-out from S,
   respectively //
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   if OPEN is empty then stop; // 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
```

GMD 알고리즘의 분석을 이용하여 다음과 같이 *LGMD* 알고리즘의 성능을 유도하게 되는데 *GMD*의 한 노드씩의 확장 대신 한 선분을 한 번에 확장하므로 m을 삭제하면 된다.

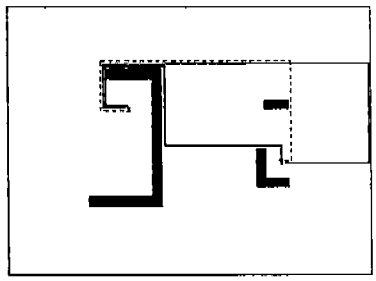
정리 8. *LGMD* 알고리즘은 $O(e\log e+N\log N)$의 시간과 $O(e+N)$의 메모리로 구현된다. 여기서 e는 *CRITICAL* 내에 있는 모든 선분의 수이고 N은 *COMPLETE* 내에 있는 모든 선분의 수이다.

(그림 11)은 현존의 최단 경로 알고리즘들과 본 논문에서 제안한 *GMD* 및 *LGMD* 알고리즘을 비교한 것이다. 이는 *GMD*와 *LGMD* 알고리즘이 실행 속도면에서나 메모리 면에서 우수할 뿐만 아니라, 아울러 최적화된 경로를 보장함을 보여줄 뿐만 아니라 connection graph(혹은 track graph)를 만드는 preprocessing 없이도 수행됨을 나타내 준다.

4. 혼합된 최단 경로

이번 장은 전 장에서 보여 준 *LGMD* 알고리즘을 기초로 하여 길이 뿐만 아니라 꺾이는 횟수까지 고려한 혼합형 최단 경로 알고리즘을 제안하였다. 꺾이는 횟루를 고려한 경로는 최근에 많은 관심이 되고 있는 분야이다[2, 25]. Yang et al.[25]은 *path-preserving graph*라는 모든 최소 꺾임 경로가 내재되어 있는 그래프를 이용하여 $O(k+e\log e)$의 실행 시간을 갖는 알고리즘을 발표하였는데 여기서 e는 장애물의 모든 변들의 합이고, k는 *extreme point*로부터의 *track*들과의 교차점들의 합이다. k는 $O(ne)$로 다시 표현되는데 여기서 n은 장애물의 수이다.

특별히 이 장에서 다룰 경로는 track 그래프(*path-preserving* 그래프)를 먼저 만들지 않고도 찾을 수 있는 최소 꺾임 경로(*minimum-bend path*), 최소 꺾임-최단 경로(*minimum-bend shortest path*), 그리고 최단 경로-최소 꺾임(*shortest minimum-bend path*)이다. 장애물이 움직이는 동적 환경(dynamic environment)에서는 어느 장애물이 움직인다면 보통 track 그래프를 다시 만들어야 하지만 *LGMD*에 사용된 또 위에 기술한 알고리즘은 그러한 track 그래프 없이 단지 움직인 장애물을 첨가(insertion) 혹은 제거(deletion)의 동작만으로도 수행되는 장점이 있다.



LGMD_MB: 최소 꺾임 경로(minimum-bend path)
LGMD_MBS: 최소 꺾임-최단 경로(minimum-bend shortest path)
LGMD_SMB: 최단 경로-최소 꺾임(shortest minimum-bend path)

(그림 11) 혼합형 최단 경로들
(Fig. 11) Combined Shortest Paths

다음은 이러한 알고리즘들을 표현하는 이름들이다(그림 11 참조).

(i) *LGMD_MB*: 최소 꺾임 경로(minimum-bend path)

(ii) *LGMD MBS*: 최소 꺾임-최단 경로(minimum-bend shortest path)

(iii) *LGMD_SMB*: 최단 경로-최소 꺾임(shortest minimum-bend path)

이들 알고리즘은 최저 임계치의 값을 제외하고는 *LGMD* 알고리즘과 유사하다. 먼저 *LGMD_MB* 알고리즘에서는 각 선분 $u \rightarrow v$에 대해 *LGMD*에서의 4가지 요소로 구성된(*dir*, *C*, *DL*, *p*)라는 정보(2장 참조)대신 (*dir*, *C*, *MB*, *p*)를 사용하게 되는데 여기서 최저 임계치인 *MB*는 경로 $P=(s \rightarrow \cdots \rightarrow u \rightarrow v)$가 가진 꺾임의 총수이다. 따라서 최소의 *MB*를 가진 선분이 다음 확장에 사용되게 된다. 이를위해 *LGMD*에서 사용된 *OPEN*이라는 *priority queue*를 이용하게 된다.

다음은 *LGMD_MB* 알고리즘을 상술한 것이며 (그림 12)은 *LGMD_MB*에 의해 탐색한 예를보여 준다.

LGMD_MB 알고리즘

algorithm *LGMD_MB* (*s*,*t*);

// same to the lines 1-6 in **algorithm** *LGMD* (*s*,*t*) described in section 3 //

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 //

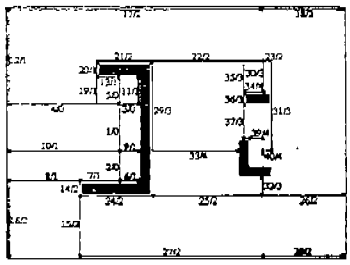
7 update *MB*[$v \rightarrow w$];

end *SEARCH_MB*

LGMD 알고리즘의 분석을 이용하여 다음과 같이 *LGMD_MB* 알고리즘의 성능을 유도하게 된다.

Theorem 9. *LGMD MB* 알고리즘은 $O(e\log e + N\log N)$의 시간과 $O(e+N)$의 메모리로 구현된다. 여기서 e는 *CRITICAL*내에 있는 모든 선분의 수이고 N은 *COMPLETE*내에 있는 모든 선분의 수이다.

LGMD_MBS 알고리즘은 최저 임계치의 값을 제외하고는 *LGMD_MB* 알고리즘과 유사하다. 먼저 *LGMD_MBS* 알고리즘에서는 각 선분 $u \rightarrow v$에 대해 5가지 요소로 구성된 (*dir*, *C*, *DL*, *MB*, *p*)를 사용하게 되는데 여기서 최저 임계치는 *DL*와 *MB*를 같이 사용하게 된다. 즉 *OPEN*에 있는 선분 중 최소의 *MB*값을 가진 선분들 가운데 다시 최소의 *DL* 값을 가진 선분이 다음 확장에 사용되게 된다.



(그림 12) *LGMD_MB* 알고리즘을 사용한 최소 꺾임 경로

(Fig. 12) Extended Line Segments for the *LGMD MB* Algorithm

LGMD SMB 알고리즘의 경우는 *OPEN*에 있는 선분 중 최소의 *DL* 값을 가진 선분들 가운데 다시 최소의 *MB* 값을 가진 선분이 다음 확장에 사용되는 것을 제외하고는 *LGMD MBS* 알고리즘과 동일하다. 따라서 다음과 같이 *LGMD MB*와 *LGMD_MBS*의 성능을 분석할 수 있다.

정리 **10.** *LGMD MBS*(혹은 *LGMD_SMB*) 알고리즘은 $O(e\log e + N\log N)$의 시간과 $O(e+N)$의 메모리로 구현된다. 여기서 e는 *CRITICAL*내에 있는 모든 선분의 수이고 N은 *COMPLETE*내에 있는 모든 선분의 수이다.

5. 결 론

본 논문에서는 미로 찾기형 알고리즘(mazerunning algorithms)과 선형 탐색 알고리즘(linesearch algorithms)의 장점 만을 이용한 *GMD* 알고리즘(Guided Minimum Detour algorithm)을 제안하였으며 이의 효율을 높이기 위해 선분을 이용한 *LGMD* 알고리즘을 제안하였다. 또한 혼합형 최단 경로 알고리즘들을 제시하여 꺾임의 수를 최단 경로에 첨가하였으며 이는 *LGMD*로부터 최저 임계치만을 바꿔 줌으로써 쉽게 구현

되고 있다. 이들은 현존의 알고리즘과 비교하여 (그림 11)에서 나타난 바와같이 실행 속도면에서나 메모리 면에서 우수할 뿐만 아니라, 아울러 최적돐된 경로를 보장해 준다. 여기서 (그림 11)에서 명기된 현 선형 탐색(line-search)알고리즘 중 가장 잘 알려진 Wu et al.[24] 알고리즘과 *LGMD* 알고리즘을 비교해 본다. Wu 알고리즘은 탐색 전에 track graph인 G_T를 만든다. G_T를 위한 메모리는 $O(e+k)$이 소요되며 시간은 $O((e+k)\log t)$가 소요된다. 여기서 e는 장애물들의 각 변들의 합이고, k는 G_T의 노드롤의 합이며, 또한 t는 장애물에 대한 외곽변(extreme edges)들의 수이다(자세한 정의는 [24]참조). 최악의 경우 $t=O(e)$이며 $k=O(e^2)$이기 때문에 $O(e^2)$의 메모리와 $O(e^2\log e)$의 시간으로 다시 표현된다. 반면 *LGMD* 알고리즘은 $O(e+N)$의 메모리와 $O(e\log e+N\log N)$의 시간이 소요된다. 이는 탐색된 총 선분의 수인 N에 의존하므로 track graph를 만들지 않는 *LGMD* 알고리즘은 $O(e^2)$ 보다는 작게된다. 따라서 *LGMD* 알고리즘은 Wu 알고리즘보다 메모리와 실행시간 면에서 우수함을 보여준다.

또한 탐색방법은 본 논문에서 새로 제안한 *Guided A** 탐색 방법을 적용하였다. 특히 track 그래프(*path-preserving* 그래프)를 먼저 만들지 않고도 찾을 수 있는 최소 꺾임 경로(*minimum-bend path*), 최소 꺾임-최단 경로(*minimum-bend shortest path*), 그리고 최단 경로-최소 꺾임(*shortest minimum-bend path*) 알고리즘들을 제안함으로써 장애물이 움직이는 동적환경(dynamic environment)에서 어느 장애물이 움직인다 하더라도 track 그래프를 다시 만들지 않고서도 단지 움직인 장애물을 첨가(insertion)혹은 제거(deletion)만 한면 수행되는 장점이 있다.

참 고 문 헌

[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] M. T. De Berg, M. J. Van Kreveld, B. J. Nilsson, and M. H. Overmars, "*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, 1990.

[3] A. D. Brown and M. Zwolinski,"*Lee Router Modified for Global Routing*," Computer-Aided Design, Vol 22, No, 5, pp. 296-300, June, 1990.

[4] 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. 251-57, ACM, 1987.

[5] H. Edelsbrunner and M. H. Overmars, "*Some Methods of Computational Geometry Applied to Computer Graphics*," Computer Vision, Graphics, and Image Processing. Vol 28 pp. 92-108, 1984.

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

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

[8] 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.

[9] 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.

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

[11] Y. Ke, "*An Efficient Algorithm for Link-Distance Problems*," in Proceedings of 5th

ACM Symp., Lect. Notes in Computer Science, 447, Springer Verlag, pp. 213-224, 1990.

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

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

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

[15] K. Mikami, K. Tabuchi, "*A Computer Program for Optimal Routing of Printed Circuit Connectors*," IFIPS Proceedings, H-47, pp. 1475-78, 1968.

[16] J. S. B. Mitchell, "*An Optimal Algorithm for Shortest Rectilinear Paths Among Obstacles in the Plane*," in Abstracts of the First Canadian Conference of Computeational Geometry, p. 22, 1989.

[17] E. F. Moore, "*The Shortest Path Through a Maze*," Annals of the Harvard Computation Laboratory, Vol 30, Pt. II pp. 285-92, 1959.

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

[19] I. Pohl, "*Heuristic Search Viewed as Path Finding in a Graph*," Artificial Intelligence, Vol. 1, pp. 193-204, 1970.

[20] I. Pohl, "*Bi-Directional Search*," Machine Intelligence, Vol 6, pp. 127-40, 1971.

[21] 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. 204-13, ACM, 1985.

[22] F. Rubin, "*The Lee Path Connection Algorithm*," IEEE Transactions on Computers, C-23(9), pp. 907-14, 1974.

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

[24] 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. 321-31, 1987.

[25] C. D. Yang, D. T. Lee, and C. K. Wong, "*On Bends and Lengths of Rectilinear Paths : A GraphTheoretic Approach*," in Proceedings of Algorithms and Data Structures, 2nd Workshop WADS '91, Lect. Notes in Computer Science, 519, Springer-Verlag, pp. 320-330, 1991.

[26] S. Q. Zheng, J. S. Lim, and S. S. Iyenger, "*Efficient Maze-Running and Line-Search Algorithms for VLSI Layout*," in Proceedings of the IEEE Southeastcon '93, Session M4B, IEEE, 1993.

[27] E. M. McCreight, "*Priority Search Trees*," SIAM Journal of Computers, Vol. 14, No. 2, pp. 257-276, May 1985.

임 준 식

1986년 인하대학교 전자계산학과 졸업(학사)
1989년 University of Alabamaat Birmingham(미) 전자계산학과 졸업(석사)
1994년 Louisiana State University(미) 전자 계산학과 졸업 (Ph.D.)
1995년~현재 경원대학교 전자계산학과 전임강사
관심분야 : VLSI Design, Speech Recognition, Robotics