한국ITS학회논문지
제12권, 제5호 (2013년 10월)

http://dx.doi.org/10.12815/kits.2013.12.5.036

pp.36~45

최소 기대 부하량을 이용한 최단경로 탐색 알고리즘 개발

Development of a Shortest Path Searching Algorithm Using Minimum Expected Weights

유 영 근*
(Ryu, Yeong-Geun)

요 약

본 연구에서는 최단경로를 반드시 찾아내는 Dijkstra알고리즘의 장점과 최단경로 탐색 소요시간을 단축시키는 A*알고리즘의 장점을 결합시킨 새로운 최단경로 탐색 알고리즘을 개발하였다.개발한 알고리즘은 탐색노드에서 목적노드까지의 최소 기대 부하량을 산출하고 이 값을 이용하여 계속 탐색 또는 이전 탐색노드로의 후퇴를 결정한다.최소 기대 부하량은 목적노드까지의 직선거리에 최소 가로 부하량 원단위를 곱하여 산출하는데,적용하는 네트워크에서는 그 값 이하의 부하량이 존재할 수 없는 값이다.

개발한 알고리즘을 실제 네트워크에 적용하여 최단경로를 탐색해 본 결과,어느 정도의 탐색 소요시간은 필요로 하나 완벽하게 최단경로를 구축하는 것으로 나타났다.개발한 알고리즘은 광역의 네트워크를 이용하는 차량 경로 안내시스템 등에서 효과를 가질 것으로 판단한다.

핵심용어 : 최단경로탐색 알고리즘, 경로안내시스템, 탐색영역, A* 알고리즘, 다익스트라 알고리즘

Abstract

This paper developed a new shortest path searching algorithm based on Dijkstra's algorithm and A* algorithm, so it guarantees to find a shortest path in efficient manner. In this developed algorithm, minimum expected weights implies the value that straight line distance from a visiting node to the target node multiplied by minimum link unit, and this value can be the lowest weights between the two nodes. In behalf of the minimum expected weights, at each traversal step, developed algorithm in this paper is able to decide visiting a new node or retreating to the previously visited node, and results are guaranteed.

Newly developed algorithm was tested in a real traffic network and found that the searching time of the algorithm was not as fast as other A* algorithms, however, it perfectly found a minimum path in any case. Therefore, this developed algorithm will be effective for the domain of searching in a large network such as RGV which operates in wide area.

Keywords : Shortest path algorithm, RGV, Searching area, A* algorithm, Dijkstra algorithm

* 주저자 : 영남교통정책연구원 원장
† 논문접수일 : 2013년 05월 03일
† 논문심사일 : 2013년 09월 04일
† 게재확정일 : 2013년 09월 15일

Ⅰ. 서 론

최단경로 탐색 소요시간을 단축하기 위한 연구들은 컴퓨터공학 분야, 인공지능(AI) 분야, 교통공학 분야 등에서 지속적으로 진행되고 있다.

교통공학 분야에서 최단경로 탐색 알고리즘의 연구는 RGS(In-Vehicle Route Guidance Systems) 개선을 위한 것으로 출발노드에서 목적노드까지의 최단경로를 보다 정확하면서 보다 빠르게 탐색할 수 있도록 하기 위함이다.

최단경로를 탐색하는 가로 네트워크(Network)에서는 차량 주행속도에 절대적으로 영향을 주는 교통량의 변화가 더욱 심해지고, 예측 불가능한 교통사고 등도 더욱 빈번하게 발생함에 따라 목적노드까지의 최단경로가 바뀌는 경우가 많아지고 있다.

그리고 차량성능의 향상과 함께 운행대수의 증가, 통행거리의 증대 등으로 탐색 네트워크의 확장이 지속적으로 이루어지고 있다.

결국, 광역 네트워크에서 빠르게 변화하는 최단경로를 정확히 탐색해 낼 수 있는 알고리즘의 개발이 필요하다[1].

하나의 출발노드에서 하나의 목적노드를 찾는 기존의 최단경로 탐색 알고리즘들 중에서 가장 기본적이면서도 안정적인 알고리즘은 Dijkstra 알고리즘[2]이다.

Dijkstra 알고리즘의 장점은 출발노드에서 목적노드까지 경로가 존재하는 경우, 반드시 최단경로를 찾아내는 것이다. 그러나 전체 탐색 네트워크에서 목적노드까지의 노드 수를 기준으로 하는 동심원상에 있는 모든 노드들을 탐색해야 하므로, 최단경로 탐색 소요시간이 길어지는 단점을 가진다.

Dijkstra 알고리즘의 탐색 소요시간 감소를 위한 연구들은 지속되어 왔으며 크게 3가지 방법으로 분류할 수 있다[3, 4]. 첫 번째 방법은 출발노드와 목적노드 양측에서 동시에 최단경로를 탐색하는 양방향 탐색법(Bidirectional Search)이고, 두 번째 방법은 휴리스틱 탐색으로서 목적지 직접 탐색방법(Goal-Directed Search(A*))이다. 세 번째 방법은 탐색 네트워크를 구분하여 최단경로 탐색을 행하는 네트워크 계층구분법(Hierarchical Methods)이다.

최단경로 탐색 소요시간을 단축하기 위하여 개발된 세가지 유형의 방법들은 모두 장점과 단점을 가지고 있으며, 네트워크의 특성에 따라서 효율성을 달리한다.

본 연구는 광역 네트워크에서 최단경로를 정확하면서 빠르게 탐색하는 알고리즘 개발을 목적으로 한다. 개발하는 알고리즘은 빠른 탐색을 장점으로 가지나 목적노드 탐색의 실패 확률이 높은 휴리스틱 알고리즘(A* 알고리즘)을 개선하는 방향으로 접근하는데, 탐색의 실패를 하지 않기 위해서 최소 가로 부하량 원단위를 이용하여 알고리즘을 개발한다.

최소 가로 부하량 원단위를 이용하여 탐색영역을 축소하는 기존 연구[5]는 있으나, 본 연구에서는 이를 확장하여 최단경로 탐색에 까지 이용하는 것이다.

본 연구에서 개발하는 최단경로 탐색 알고리즘의 효율성 검증은 실제 네트워크와 실제 가로 부하량(가로 통행시간)을 이용하여 유사한 방향으로 연구된 기존 탐색 알고리즘과 탐색 결과(최단경로의 정확성, 탐색속도 등)의 비교로부터 행한다.

Ⅱ. 기존 연구

1. 양방향 탐색법(Bidirectional Search)

양방향 탐색법은 출발노드와 목적노드 양측에서 최단경로 탐색을 같이 행하는 것이다. 이 방법은 양측에서의 탐색이 중간에서 반드시 만나야 탐색영역을 줄이는 효과를 얻을 수 있다.

양방향 탐색방법이 효율성을 가지기 위해서는 해결해야 할 두 개의 과제가 있다. 첫 번째 과제는 하나의 탐색방향에서 반대방향 탐색으로 바꾸는 조건이다. 양측에서 반복하여 탐색을 하면 편리하나, 가장 효율적이라고 할 수 없다[6].

두 번째 과제는 양측 방향에서의 탐색 중단 조건인데, 중단 조건 충족을 위한 과정이 결국 단일 방

향 탐색보다 비효율적으로 된다는 것이 밝혀졌다[6].

A* 알고리즘으로도 양방향 탐색법을 제시한 연구가 있으나, 네트워크에 따라서 중간에 만나지 못할 확률이 크게 되는 단점이 있으므로, 단방향 A* 알고리즘 보다 효율적이지 못한 것으로 판단할 수 있다[7].

2. 목적지 직접 탐색법(Goal-Directed Search)

목적지 직접 탐색법은 휴리스틱 탐색이며, 대표적인 것으로 A* 알고리즘을 들 수 있다. A* 알고리즘은 휴리스틱 추정치(Estimate)를 사용하는 평가함수(Guidance function)에 의해 최단경로를 탐색한다[8].

탐색노드 n에서의 평가함수는 식(1)과 같다.

$$f(n) = g(n) + h(n) \quad (1)$$

$g(n)$: 출발노드에서 n노드 까지의 부하량
$h(n)$: n노드에서 목적노드까지의 추정 부하량

$h(n)$은 실제 부하량보다 적어야 하므로, n노드에서 목적노드까지는 직선거리(Manhattan distance, Euclidean distance)를 많이 이용한다. 만약 $h(n)$이 상수(예로, "0")라면 A* 알고리즘은 Dijkstra 알고리즘과 같게 된다.

A* 알고리즘의 탐색노드는 평가함수 $f(n)$이 최소값을 가지는 노드를 선택하며, 목적노드가 검색될 때 까지 반복한다.

A* 알고리즘은 깊이 우선 탐색(Depth First Search)이므로 목적노드가 없는 경로에 깊게 빠질 우려가 있고, 목적노드에 이르는 경로가 다수인 경우에는 탐색된 경로가 최단경로가 아닐 확률이 높게 된다. 즉, 목적노드가 발견되면 검색하지 않은 타 노드가 있어도 최단경로 탐색을 종료하므로 최단경로 탐색을 실패할 확률이 높게 되는 단점이 있는 것이다.

Fu(1996)는 A* 알고리즘의 평가함수를 수정, 적용하여 탐색영역을 축소하고 탐색시간을 줄이는 방안을 (2)식과 같이 제안하였다[9].

$$g(n) + h(n) \le E_{(o,d)} \quad (2)$$

$E_{(o,d)}$: 출발노드에서 목적노드까지의 최소소요시간(부하량)추정 한계값

목적노드까지 추정부하량 $h(n)$은 식(3)과 같이 제안하였고, 출발노드에서 목적노드까지의 한계 값 $E_{(o,d)}$는 식(4)와 같이 제안하였다.

$$h_{(n)} = S_{(n)} / V \quad (3)$$

$S_{(n)}$: n 탐색노드에서 목적노드까지의 직선거리
V : 네트워크에서의 평균속도

$$E_{(o,d)} = K \bullet h(n) \quad (4)$$

K : 조정계수

식(4)에서 조정계수 K는 네트워크 사전 조사로부터 산정하거나, 목표노드 탐색 실패시 K값을 증가시키면서 조정해 가는 방법을 제안하였다.

3. 네트워크 계층 구분법(Hierarchical Methods)

네트워크 계층 구분법은 AI분야에서 많이 연구되어 왔다[10, 11]. 교통분야에서 이 방법은 기본적으로 운전자가 지도책을 보면서 경로를 찾는 것과 같은 방법이 된다[9]. 즉 예로 서울시청에서 부산시청의 경로를 생각할 때, 출발노드와 목적노드 부근에서는 복잡한 서울, 부산의 전체 가로 네트워크(하위계층)에서 탐색을 하지만 서울과 부산간은 단순한 고속도로(국도, 상위계층)만을 탐색하게 되는 것이다.

이 방법은 탐색 노드를 줄일 수 있어 탐색 소요시간이 단축되는 장점을 가지나, 단순한 탐색영역 제약만이 행해진다고 볼 수 있으므로 최단경로 탐색에 실패할 확률도 가지게 된다.

그리고 이 방법의 또 다른 어려운 점은 상위계층 네트워크와 하위계층 네트워크를 구분하는 기준의 결정과 계층간 연결지점(노드)의 결정문제이다[9].

또한, 하위계층 네트워크에서 가질 수 있는 지름길 등 최단경로가 배제되는 문제가 있다.

결국, 계층 구분법은 출발노드와 목적노드가 원거리 일 경우에만 최단경로 탐색의 성공 확률이 높고, 가까울 경우에는 비효율적인 탐색방법이 된다.

Ⅲ. 최소 가로 부하량 원단위

최소 가로 부하량 원단위는 탐색 네트워크 내의 노드간 부하량을 해당 노드간 직선거리(Euclidean Distance)로 나눈 값들 중에서 최소 값이 되는데 탐색영역 축소를 위한 유영근(2013)의 연구에서 적용되었다. 이 연구에서는 최소 가로 부하량 원단위를 이용하여 노드 수를 최소로 하는 임시경로의 부하량 보다 적은 부하량을 가질 가능성이 있는 노드들을 찾는 방법을 개발한 것이다.

즉, 모든 노드가 출발노드와 목적노드로부터 직선으로 연결되어 있고 최소 부하량 원단위와 같은 부하량을 가지는 것으로 가정할 때, 임시경로에서의 부하량보다 적은 부하량을 가질 수 있는 직선거리(부하량 경계 값)의 경계를 산정할 수 있다.

최소 부하량 원단위(Min_Weight_Unit)를 A로 하고, A와 같은 임시경로 원단위를 B로 하면 (5)식이 성립되고 직선거리 경계 값(Weight_Bounds, X)은 (6)식으로 계산된다.

$$A\left(\frac{\text{부하량}}{\text{직선거리}}\right) = B\left(\frac{\text{임시경로 부하량}}{X(\text{직선거리 경계값})}\right) \quad (5)$$

$$Weight\_Bounds = TPRW / Min\_Weight\_Unit \quad (6)$$

$Weight\_Bounds$: 직선거리 경계 값
$TPRW$: 임시경로의 부하량
Min_Weight_Unit : 최소 부하량 원단위

하나의 노드에서 출발노드로 부터의 직선거리와 그 노드에서 목적노드까지의 직선거리 합이 직선거리 경계 값 보다 적을 경우(식 (7)), 그 노드는 임시경로 보다 적은 부하량을 가지는 최단경로에 포함될 가능성이 있는 노드가 되는 것이다.

$$(Ecu\_Dis[sn][n] + Ecu\_Dis[n][en]) \leq Weight\_Bounds \quad (7)$$

$Ecu\_Dis[i][j]$: 노드 i 에서 노드 j 까지의직선거리
sn : 출발노드
en : 도착노드
n : 탐색노드

직선거리 경계 값 보다 클 경우, 그 노드를 포함한 경로는 임시경로 보다 부하량이 적은 최단경로가 될 수 없기 때문에 그 노드에서의 최단경로 탐색은 최단경로 탐색시간의 손실만 가져온다[5].

Ⅳ. 최소 기대 부하량

최소 기대 부하량은 탐색노드에서 목적노드까지 대상 네트워크에서 가능한 최소 부하량의 예측치로써, 직선거리에 최소 가로 부하량 원단위를 적용하여 산출한다.

본 연구에서는 최소 기대 부하량을 이용하여 탐색영역 축소 알고리즘[5]에서와 같이 노드수를 최소로 하는 임시경로를 구축하고, 임시경로 부하량과의 비교로부터 탐색중인 노드에서 탐색 진행여부를 정확히 결정할 수 있게 한다.

최소 기대 부하량은 탐색노드로부터 최단경로 가능성을 최대로 확보하기 위하여 산정하는 값으로 탐색노드에서 목적노드까지 직선으로 연결되어 있고, 최소 부하량 원단위(A)와 동일한 값을 갖고 있다고 가정하여 산출하는 값이다.

$$A\left(\frac{\text{부하량}}{\text{직선거리}}\right) = B\left(\frac{X(\text{최소 기대 부하량})}{(\text{목적노드까지 직선거리})}\right) \quad (8)$$

결국, 식(9)과 같이 목적노드까지의 직선거리(Ecu_Dis[n][en])에 최소 가로 부하량 원단위(Min_Weight_Unit)를 곱하면, 탐색노드(n)에서 목적노드까지의 최소 기대 부하량(emw(n))이 산출된다.

즉, 해당 탐색노드에서 목적노드까지 최소 기대 부하량 보다 적은 부하량은 존재하지 않는 것이다.

$$emw(n) = Ecu\_Dis[n][en] \times Min\_Weight\_Unit \quad (9)$$

$emw(n)$: 탐색노드(n)에서 목적노드(en)까지 최소 기대 부하량
$Ecu\_Dis[n][en]$: 탐색노드(n)에서 목적노드(en)까지 직선거리

통행시간을 부하량으로 할 경우, 가로 구간별 통행시간을 가로 직선거리로 나눈 값 중 최소 값은 실제적이지 못한 값을 산출하므로 이를 다음 식(10)과 같이 수정 적용할 필요가 있다.

즉, 가로 통행시간은 속도와 관계가 있고 실제 속도는 직선거리가 아닌 실제거리에서 적용되어야 하므로 네트워크에서 실제로 가장 빠른 속도를 찾아 거리에 적용한다.

거리적용은 가로 실제거리를 가로 직선거리 값으로 나눈 값 중에서 최소인 값을 찾아 적용하면, 보다 실제적인 원단위가 되고, 보다 효율적인 탐색이 가능하게 된다.

$$Min\_Weights\_Unit = Min.\left(\frac{\text{가로 통행시간}}{\text{가로 직선거리}}\right) \Rightarrow$$

$$Min.\left(\frac{\text{가로 실제거리}}{\text{가로 직선거리}}\right) \times Min.\left(\frac{\text{가로 통행시간}}{\text{가로 실제거리}}\right) \quad (10)$$

직선거리 경계 값 또한 식(11)과 같이 변형하는 것이 효율적이다.

$$Weight\_Bounds = TPRW/Min.\left(\frac{\text{가로 통행시간}}{\text{가로 직선거리}}\right) \Rightarrow$$

$$TPRW/Min.\left(\frac{\text{가로 실제거리}}{\text{가로 직선거리}}\right)/Min.\left(\frac{\text{가로 통행시간}}{\text{가로 실제거리}}\right) \quad (11)$$

n 노드에서의 평가함수(Chk_Value) 값이 임시경로의 부하량(TPRW)보다 크다면 n 노드로부터의 계속된 탐색은 임시경로보다 더 적은 부하량을 가지는 경로를 구축할 가능성이 없기 때문에 불필요하게 된다.

그러므로 n 노드로 부터의 탐색을 중단하고, n 노드 탐색 이전 단계의 노드로 후퇴하여 탐색되지 않은 노드를 탐색한다.

$$Chk\_Value = g(n) + emw(n) \quad (12)$$

Chk_Value: 평가함수
$g(n)$: 출발노드에서 n노드까지의 부하량
$emw(n)$: 탐색노드(n)에서 목적노드(en)까지 최소 기대 부하량

If $Chk\_Value < TPRW$
n 노드에서 계속 탐색

Else
n 노드 이전 탐색단계로 후퇴

Ⅴ. 새로운 최단경로 탐색 알고리즘

새로운 최단경로 탐색 알고리즘은 최소 기대 부하량을 이용하여 빠르고 정확한 최단경로 구축을 목적으로 한다.

개발 알고리즘은 탐색 대상 네트워크에서 최소 가로 부하량 원단위(New_Min_Weight_Unit)를 우선하여 산정한다. 그리고 임시경로의 부하량을 임의의 큰 값으로 우선 설정해 둔다.

그리고 출발노드에서 연결된 노드 중 평가함수 값이 가장 적은 노드를 선택하여 탐색을 행하는데, 이 노드는 같은 단계(Step)에서의 검색 종료 노드가 아니며, 기존 탐색영역의 축소 조건을 만족하는 공간적 범위 내에 있어야 한다.

선택된 노드는 그 탐색단계의 Closed 배열에 넣어 같은 탐색단계에서는 탐색하지 않는다. 그러나 다른 경로로 그 노드가 연결되어 있다면 $g(n)$ 값이 다르기 때문에 다시 평가함수를 계산하고, 진행 또는 후퇴를 결정한다.

또한, 선택된 노드의 평가함수(Chk_Value) 값이 임시경로 부하량 보다 적다면 그 노드에서 계속해서 최단경로를 탐색한다. 목적노드까지 탐색되면, 그 경로가 임시경로가 되며, 부하량(TPRW)도 바꾼다.

다시 전 단계의 탐색노드로 후퇴하여 탐색하지 않은 노드들 중 최소 평가함수 값을 가지는 노드를 찾고, 그 노드에서의 진행여부를 판단한다. 이와 같은 과정으로 더 이상 검색할 노드가 없으면 종료한다. 종료 후 임시경로가 최단경로가 되고, 임시경로

의 부하량이 최단경로의 부하량이 된다.

개발하는 알고리즘은 A* 알고리즘과 같이 목적노드가 검색되면 종료하는 것이 아니라, 탐색영역 속에서 모든 노드를 탐색하는 것과 같은 효과를 가진다. 따라서 최단경로 탐색을 실패하지 않는다.

최소 기대 부하량을 이용한 최단경로 탐색 알고리즘을 단계(Step)별로 정리하면 다음과 같다.

```
< STEP 0 >
Min_Weight_Unit  /*  대상 네트워크에서 산정          */
i = 0            /*  반복(Recursion) 회수          */
p[i] = sn        /*  탐색 노드                     */
g[i] = 0         /*  출발 노드(sn)로부터 i번째
                     탐색노드까지의 부하량          */
Tem_Path[i] = p[i] /*  임시 최단경로의 i번째 노드   */
Shortest_Path[i] = p[i]  /* 최단경로의  I번째 노드   */
Closed[i] = {sn}   /*  i반복에서 탐색제외 노드집합 */
TPRW = MAX       /* 임시경로 부하량을 임의의
                    최대값으로 가정                 */
```

< STEP 1 >
i 를 1 증가시킨다.(i = i + 1)

< STEP 2 >
다음 조건을 만족하는 노드집합을 구축한다. 노드집합이 ϕ이면 STEP 6으로 간다.

- 노드 p[i-1]에 직접 연결.
- 탐색제외 노드집합 closed[i]에 비포함.
- 임시 최단경로(Tem_path[k], k = 0~i)에 비포함.
- 출발노드에서 그 노드까지의 직선거리와 도착노드까지의 직선거리 합이 직선거리 경계 값 (*Weight_Bounds*) 보다 적을 것.

< STEP 3 >
구축된 노드집합에서 각 노드의 평가함수(Chk_Value)를 계산하고, 최소값의 노드를 p[i]로 한다. p[i]를 탐색제외 노드집합 Closed[i]에 저장한다.
p[i]의 평가함수 값이 임시경로 부하량(TPRW)보다 크면 STEP 2로 간다.

< STEP 4 >
임시 최단경로 Tem_Path[i]에 p[i]를 대입한다. p[i]가 목적노드(en)가 아니면 STEP 1로 간다.

< STEP 5 >
Chk_Value값을 TPRW로 하고, i를 NN(number of node)으로 하며, Tem_Path[k] 를 Shortest_Path[k](k=0~NN)로 바꾼 후, STEP 2로 간다.

< STEP 6 >
탐색 제외 노드집합 Closed[i]를 공집합(Closed[i] = $\varnothing$)으로 하고, i 를 1감소시킨다. i가 "0"이 아니면 STEP 2로 간다.

< STEP 7 >
최단경로는 Shortest_Path[k](k=0~NN)가 되고 최단경로의 부하량은 TPRW이다.

개발한 알고리즘의 시간 혼잡도(time complexity)는 악조건에서 Dijkstra 알고리즘과 같이 $O(N^2)$이 되나, 모든 노드를 탐색해 가는 것이 아니라, 가능성 있는 노드 만을 탐색하는 차이가 있다. 즉, 광역 네트워크에서 출발노드와 목적노드가 가까울 경우 개발한 알고리즘은 큰 효과가 있을 것으로 판단한다.

Ⅵ. 개발 알고리즘의 효율성

1. 효율성 검증방법

본 연구에서 개발한 최단경로 탐색 알고리즘의 효율성 검증은 개발 방향이 유사하다고 판단되는 알고리즘들과 탐색 소요시간과 정확성 등의 항목을 비교하는 것으로 하였다.

비교 대상 알고리즘은 A* 알고리즘과 Fu(1996)의 제안을 대상으로 하였는데, 제안은 A* 알고리즘을 개선한 것으로 본 연구의 알고리즘 개발 방향과 유사하게 탐색영역을 축소하고 평가함수를 수정하여 적용하는 것이다.

Fu의 제안에서 평가함수는 A* 알고리즘에서 적용한 직선거리를 네트워크 평균속도로 나누어 적용하는 것을 제안하고 있다.

그리고 최단경로 예상 부하량 조정치인 K 값도 탐색시간 단축을 위해 우선 구축된 경로(임시경로) 값에서 추정되게 하였으며, 본 연구에서의 직선거리 경계 값(Weight_bounds)과 같은 기능을 수행하도록 하였다.

Fu의 제안, 즉 수정된 평가함수와 k 값을 A* 알고리즘에 바꾸어 적용 할 경우, A* 알고리즘의 부

정확성의 영향이 클 것으로 판단되므로, 본 연구에서 개발한 알고리즘에 적용하여 비교하였다.

2. 사례 네트워크와 부하량

개발한 최단경로 탐색 알고리즘의 효율성 검증을 위한 사례 네트워크는 탐색시간과 탐색 최단경로의 정확성 정도를 효과적으로 보기 위해서 여러 개의 도시권을 포함하는 보다 광역적 네트워크가 필요하다.

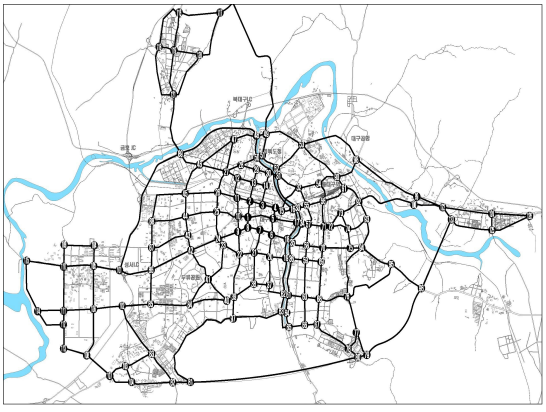
그러나, 현재 교통정보 여건에서 즉시적으로 변화하는 가로별 부하량(통행시간 등)을 이용하기 위해서는 하나의 도시권 영역으로 한정할 수밖에 없다. 그리고 사례연구에서 부하량은 자료이용이 객관적으로 가능한 가로 통행시간으로 하였다.

검증을 위한 네트워크는 대구시 간선가로 네트워크로 하였으며, 대구시 ATMS(Advanced Traffic Management System)로 부터 5분 단위로 바뀌는 간선가로 통행시간을 이용하였다.

네트워크는 노드 수가 126개, 가로(링크)수는 213개이며, 가로가 규칙적인 격자형이 아니라 격자형과 방사·환상형이 결합한 복잡한 구조로 되어 있다.

통행시간 자료는 2013년 6월 13일 18시 15분에 나타난 가로 통행시간 값을 적용하였다.

대구시 네트워크는 <그림 1>에서 나타내었다.



〈그림 1〉 대구시 네트워크
〈Fig. 1〉 Daegu-city Network

3. 효율성 검증

사례 네트워크에서 효율성 검증을 위한 최단경로 탐색은 126개 전체 노드간(126×126－126 = 15,750개 경로)의 최단경로를 탐색하였다.

효율성 검증을 위한 알고리즘간의 탐색결과 비교항목은 최단경로 탐색 소요시간과 탐색된 경로의 정확성으로 하였고, 출발노드와 목적노드간의 직선거리를 2㎞단위로 구분하여 거리에 따른 결과 비교를 하였다.

1) 최단경로 탐색 소요시간

최단경로 탐색 소요시간은 출발노드에서 최단경로 탐색을 시작하여 목표노드를 찾아 종료할 때 까지 소요된 시간을 측정한 것으로, CPU 3.09GHz, RAM 2.99GB 성능의 PC에서 수행한 결과를 비교하였다.

전체 15,750개의 최단경로를 찾는 소요시간(Total)에서 A* 알고리즘은 2.00E-04ms, Fu의 제안안은 4.00E-06ms, 본 연구에서 개발한 알고리즘은 6.00E-06ms로 분석되었다.

하나의 경로를 탐색하는데 소요된 평균 탐색시간은 A* 알고리즘이 1.09E-08ms, Fu의 제안은 2.40E-06ms, 본 연구에서 개발한 알고리즘은 3.64E-06ms으로 분석되었다.

A* 알고리즘은 출발노드와 목적노드 간의 거리에 관계없이 극히 짧은 소요시간으로 탐색하는 반면, Fu의 제안과 본 연구에서 개발한 알고리즘은 출발노드와 목적노드가 길어질수록 탐색 노드의 증가에 따라 소요시간도 길어지는 것으로 나타났다.

탐색노드에서 목표노드까지의 직선거리를 평균속도로 나눈 값을 평가함수에 적용하는 Fu의 제안안이 최소 기대 부하량을 이용하는 본 연구의 알고리즘보다 소요시간이 짧은 것으로 분석되었다.

알고리즘별 최단경로 탐색 소요시간 분석결과는 <표 1>에서 나타내었다.

〈표 1〉 최단경로 탐색 소요시간 결과
〈Table 1〉 Minimum path running time results

Range of Straight Distance(km,X) (Number of Routes)	Algorithm		
	A* (Aver. ms)	Fu's Suggestion (Aver. ms)	New (Aver. ms)
X < 2 (1,430)	1.05E-08	3.5E-08	3.5E-08
2 ≤ X < 4 (3,198)	4.69E-09	6.88E-08	9.69E-08
4 ≤ X < 6 (3,214)	9.96E-09	3.05E-07	2.89E-07
6 ≤ X < 8 (2,570)	1.79E-08	9.22E-07	1.13E-06
8 ≤ X < 10 (2,124)	1.51E-08	2.22E-06	2.97E-06
10≤ X < 12 (1,626)	9.84E-09	4.75E-06	6.97E-06
12≤ X < 14 (898)	1.0E-30	9.44E-06	1.37E-05
14≤ X< 16 (412)	1.0E-35	1.73E-05	2.42E-05
16 ≤ X (278)	5.76E-08	2.22E-05	4.76E-05
Total	2.00E-04	4.00E-02	6.00E-02
Average	1.09E-08	2.40E-06	3.64E-06

〈표 2〉 최단경로 탐색 정확성 비율 결과
〈Table 2〉 Minimum path Searching success rate results

Range of Straight Distance(km,X) (Number of Routes)	Algorithm		
	A* (%)	Fu's Suggestion (%)	New (%)
X < 2 (1,430)	59.23	93.99	100.00
2 ≤ X < 4 (3,198)	27.33	75.58	100.00
4 ≤ X < 6 (3,214)	16.46	60.11	100.00
6 ≤ X < 8 (2,570)	9.61	53.19	100.00
8 ≤ X < 10 (2,124)	5.79	46.94	100.00
10≤ X < 12 (1,626)	1.48	38.31	100.00
12≤ X < 14 (898)	1.22	31.51	100.00
14≤ X< 16 (412)	0.97	30.10	100.00
16 ≤ X (278)	0.36	24.46	100.00
Average	16.88	58.13	100.00

2) 정확성

최단경로 탐색의 정확성은 출발노드에서 도착노드까지 최단경로를 정확히 찾아내는 비율(성공비율)로 나타내었다. 경로탐색에 실패하거나, 최단경로가 아닌 경로를 구축한 경우는 실패로 하였다.

알고리즘별 정확성의 분석결과는 <표 2>에서 나타내었는데, 전체 15,750개의 경로탐색에서 A* 알고리즘은 성공률이 16.88%에 지나지 않고, Fu의 제안은 58.13%로 분석되었다. 또한 출발노드와 목적노드간의 거리가 길어질수록 성공확률이 급격히 감소하는 것으로 나타났다.

그러나 본 연구에서 개발한 새로운 알고리즘은 출발노드와 목적노드간의 거리에 관계없이 최단경로를 정확히 찾는 것으로 분석되었다.

결과적으로, A* 알고리즘은 탐색을 빨리하나, 실패확률이 높다는 것이 확인되었다.

Fu가 제안한 방법은 평균속도를 적용시킴에 따라 최소 부하량 원단위(최고속도)를 적용하는 새로운 알고리즘에 비해 직선거리를 적용한 가로 통행시간이 더 늘어나게 된다. 따라서 탐색노드가 줄게 됨으로 탐색속도는 빠르게 된다.

그러나, 최단경로 탐색 정확도는 상당히 떨어지는 것으로 나타났는데, 이는 결국 평균속도와 최고속도 사이에서 포함될 수 있는 목적노드가 검색에서 제외되었기 때문이다.

본 연구에서 개발한 알고리즘은 철저하게 불필요한 노드에게 만 제약이 가해지고 약간의 가능성이 있어도 검색을 행하기 때문에 네트워크 구조에 따라 탐색속도를 크게 줄이기에는 무리가 있을 수 있다. 그러나 최단경로 탐색의 정확도는 100%이기 때문에 효율성이 있는 것으로 판단되며, 네트워크

가 상당히 큰 경우에는 더욱 높은 효율을 나타낼 것으로 판단한다.

Ⅶ. 결론

가로상의 빠른 부하량(소요시간 등) 변화와 함께 점차 광역화 되어가고 있는 네트워크에서 RGS (In-Vehicle Route Guidance Systems)는 보다 효율적인 최단경로 탐색 알고리즘을 필요로 한다.

본 연구에서는 광역 네트워크에서 기존 Dijkstra 알고리즘의 단점과 A* 알고리즘의 단점을 극복하기 위한 알고리즘 개발을 목적으로 하였다.

본 연구는 가로 부하량 최소 원단위를 이용하여 탐색영역을 축소한 기존 연구[5]의 확장으로, 탐색영역의 축소 후, 같은 원단위를 이용하여 목적노드 탐색을 정확히 하는 알고리즘을 개발한 것이다.

개발한 알고리즘은 탐색노드에서 목적노드까지의 직선거리에 가로 부하량 최소 원단위를 곱하여 최소 기대 부하량를 산출하고, 산출된 최소 기대 부하량을 이용하여 그 탐색노드에서 진행 여부를 미리 판단할 수 있도록 한 과정을 핵심으로 한다.

개발한 알고리즘의 효율성 검증은 126개 노드를 가진 대구시 간선 네트워크을 대상으로 하여 전체 노드간의 최단경로 탐색결과를 타 알고리즘의 적용결과와 탐색 소요시간, 최단경로의 정확도를 비교하는 것으로 하였다.

비교하는 알고리즘은 탐색노드에서 목적노드까지 직선거리를 이용하여 탐색하는 A* 알고리즘, 그리고 직선거리에 네트워크 평균속도를 나누어 적용할 것을 제안한 Fu(1996)의 연구로 하였다.

검증결과, 탐색 소요시간은 A* 알고리즘이 가장 빠른 것으로 나타났고, 본 연구에서 개발한 알고리즘이 가장 늦은 것으로 나타났다. 그러나, 개발한 알고리즘의 정확도는 전혀 실패가 없는 것으로 나타났다.

효율성 검토 자료의 범위 한계에 의해 전국 또는 수개 도시를 합한 광역권에서의 검토는 행할 수 없었으나 가로상의 실시간 교통량 또는 속도자료가 광역권에서 이용 가능하게 될 경우, 개발한 알고리즘의 정확도만이 아니라 탐색속도 측면에서의 효과를 입증할 수 있을 것으로 기대한다.

차후, RGS에서 보다 실용적인 최단경로를 구축할 수 있도록 하기 위하여, 소요시간, 통행요금, 사고 위험도 등 다양한 가로 부하량을 조합하여 적용하는 방법의 개발을 수행할 필요가 있다.

참고문헌

[1] M. G. H. Bell, “Hyperstar: A multi-path Astar algorithm for risk averse vehicle navigation”, *Transportation Research Part B 43*, pp.97-107, 2009.

[2] E. W. DIJKSTRA, “A Note on Two Problems in Connexion with Graphs”, *Numerische Mathematik1*, pp.269-271, 1959.

[3] L. Fu and D. Sun and L. R. Rilett, “Heuristic shortest path algorithms for transportation applications: State of the art”, *Computers & Research Volume 33*, pp.3324-3343, 2006.

[4] W. Thomas and P. Weil, “Speed-Up Techniques for Shortest-Path Computations”, *STACS 2007, LNCS 4393*, pp.22-36, 2007.

[5] Y. G. Ryu, “Development of shortest path searching network reduction algorithm”, *The journal of The Korea Institute of Intelligent Transport Systems*, vol. 12, no. 2, pp.12-21, 2013.

[6] E. S. Dreyfus, “An appraisal of some dhortest path algorithms.”, *Operations Research*, vol. 17, pp.395-412, 1969.

[7] I. Pohl., “Bi-directional search.”, *Machine Intelligence*, vol. 6, pp.127-140, 1971.

[8] P. E. Hart and N. J. Nilsson and B. Raphael, ″A Formal Basis for the Heuristic Determination of Minimum Cost Paths″, *Transactions on Systems Science and Cybernetics SSC4 (2)*, pp.100 - 107, 1968.

[9] L. Fu, “Real-time vehicle routing and scheduling in dynamic and stochastic traffic networks.” *Ph.D.*

Dissertation, University of Albera, Edmonton, Alberta, 1996.

[10] E. D. Sacerdoti, "Planning in a hierarchy of abstraction spaces", *Artificial Intelligence*, vol. 5, pp.115-135, 1974.

[11] R. E. Korf, "Planning as search: a quantitative approach", *Artificial Intelligence*, vol. 33(1), pp.65-68, 1987.

저자소개

유 영 근 (Ryu, Yeong-Geun)
2011년 2월 ~ 현 재 : 영남교통정책연구원 원장
2001년 3월 ~ 2011년 2월 : 영남대학교 겸임교수
1997년 2월: 영남대학교 도시공학과 박사
1987년 2월: 영남대학교 도시공학과 석사
1985년 2월: 영남대학교 도시공학과 학사
E-mail : ygryu@chol.com