http://dx.doi.org/10.7236/JIWIT.2012.12.4.131
JIWIT 2012-4-16

주행시간 기반 실시간 점대점 최단경로 탐색 알고리즘

A Real-time Point-to-Point Shortest Path Search Algorithm Based on Traveling Time

이상운*

Sang-Un, Lee

요 약 네비게이션의 최단 경로 탐색 알고리즘은 일반적으로 Dijkstra 알고리즘에 기반을 두고 있으며, 가중치로 단지 길이 (거리) 만을 고려하고 있다. 거리 기반의 Dijkstra 알고리즘은 출발 노드부터 시작하여 그래프의 모든 노드에 대한 최단 경로를 결정하기 때문에 일반적으로 노드의 수 - 1회를 수행해야 하며, 알고리즘 수행에 많은 메모리가 요구된다. 또한, 거리에만 기반하기 때문에 전방에 차량사고로 인해 병목현상이 발생하였을 때 우회도로를 탐색하는 기능이 없어 항상 동일한 경로만을 탐색한다. 이러한 문제점을 해결하고자, 본 논문은 도로 등급 (고속도로, 국도, 지방도 등)을 고려하지 않고, 속도 기준 (원활, 지체 서행, 정체, 사고 통제 등)도 적용하지 않으며, 단지 도로별 주행시간 (주행속도 x 거리)을 고려한다. 이는 사고, 지체, 공사 등으로 인해 동일한 거리의 도로도 다른 시간이 소요되는 현실성을 반영하여 우회도로를 탐색할 수 있는 장점이 있다. 제안된 알고리즘은 특정 도로에서 사고가 발생하였다고 가정한 경우에도 도로의 통행속도를 실시간으로 반영함으로서 돌발지점을 우회하여 목적지 까지 최단시간 내에 도달 할 수 있음을 증명하였다.

Abstract The shortest path search algorithm of navigation is generally based on Dijkstra algorithm and considers only the distance using the weight. Dijkstra algorithm based on the distance mainly ought to perform the 'number of nodes 1' and requires a lot of memory, for it is to start from the starting node and to decide the shortest path for all the nodes. Also, it searches only the same identical path in case of any bottleneck due to an accident nearby, since it is based only on the distance, and hence does not have a system that searches the detour road. In order to solve this problem, this paper considers only the travelling time per road (travelling speed * distance), without applying speed criteria (smoothness, slow speed, stagnation and accident control) or road class (express road, national road and provincial road). This provides an advantage of searching the detour, considering the reality that there are differences in time take for the car to travel on different roads with same distance, due to any accident, stagnation, or repair construction. The suggested algorithm proves that it can help us to reach the destination within the shortest time, making a detour from any congested road (outbreak) on providing an information on traveling time continuously(real-time) even though there is an accident in a particular road.

Key Words : Shortest Path, Dijkstra's Algorithm, Point-to-Point Shortest Path Search, Length, Traveling Speed, Traveling Time

*정회원, 강릉원주대학교 과학기술대학 멀티미디어공학과
접수일자 : 2012년 4월 15일, 수정완료 : 2012년 6월 30일
게재확정일자 : 2012년 8월 10일

Received: 15 April 2012 / Revised: 30 June 2012
Accepted: 10 August 2012
*Corresponding Author: sulee@gwnu.ac.kr
Dept. of Multimedia Eng., Gangneung-Wonju National University, Korea

Ⅰ. 서 론

그래프 (Graph)는 정점 (Vertices, V)과 간선 (Edges, E)으로 구성되어 있으며, 정점들은 간선들로 연결되어 있고 (Connected), 간선들은 방향성 (Directed) 또는 무방향성 (Undirected)과 가중치가 있는 경우 (Weighted)와 없는 경우 (Unweighted)로 구분된다. 무방향 그래프 (Undirected Graph)는 $G=(V,E)$로 표기하며, 방향 그래프 (Digraph)는 $G=(N,A)$로 표기한다. 방향 그래프에서는 정점을 노드 (Nodes, N)로, 간선을 호 (Arcs, A) 또는 화살표 (Arrows)로 부른다.

방향 그래프는 최단 경로 (Shortest Path, SP)와 임계경로 (Critical Path, CP) 등을 구하는데 활용된다. 본 논문은 최단 경로에 초점을 맞춘다. 최단경로를 구하는 알고리즘은 최근 들어 GPS 차량용 네비게이션 (GPS Car Navigation System)에 일반적으로 적용되고 있다.[1] 이후부터는 “네비게이션”이라 부른다.

최단 경로를 찾는 알고리즘은 Dijkstra, Bellman Ford, 위상 순서 (Topological Ordering)와 A-Star (A*)가 있다.[1,2] 이들 알고리즘은 일반적으로 호의 가중치로 거리 (Length) 정보를 활용한다. Bellman Ford 알고리즘은 그래프 크기가 커짐에 따라 Dijkstra 알고리즘보다 많은 시간이 소요되지만 좋은 결과를 얻을 수 있다. 위상순서 알고리즘은 선형 시간대에 최단 경로를 찾을 수 있지만 최적의 경로를 찾지 못할 수 도 있다. A* 알고리즘은 Dijkstra 알고리즘과 같이 좋은 결과를 얻으며 메모리를 적게 요구하지만 알고리즘 구현이 보다 어렵다. 이와 같은 이유로 내비게이션에는 Dijkstra 알고리즘이 가장 널리 활용되고 있다.[1,6,7] Dijkstra 알고리즘은 양의 가중치를 갖는 호들로 구성된 방향 그래프에 대해 단일 출발 노드부터 다른 모든 노드까지의 최단 경로 (Single-Source Shortest Path)를 찾는 알고리즘이다[3-5].

기존에 상용화된 네비게이션의 문제점은 거리 기반의 최단경로를 탐색하기 때문에 실시간 교통흐름 정보를 반영하지 못해 이용자로부터 외면을 받고 있다. 이러한 문제점을 해결하고자, 최근 들어 실시간 교통정보 서비스를 이용하는 TPEG (Transport Protocol Expert Group) 네비게이션이 상용화되어 가급적 소통이 원활한 도로로 우회하는 방향으로 발전하고 있다. 그러나 TPEG을 탑재한 네비게이션의 경우에도 일반 네비게이션보다 월등한 성능을 자랑하지는 못하고 있다. 이는 실시간 교통정보의 이용 보다는 네비게이션 자체에 적용하는 속도 구분 (소통 원활, 지체 서행, 정체, 사고통제 등)과 도로 등급 (고속도로, 국도, 지방도로 등)에 따른 최단경로 탐색 알고리즘 문제에 기인한다고 할 수 있다.

본 논문은 도로 등급과 속도 구분 기준을 적용하지 않으며 도로별 차량의 주행시간 (Traveling Time) 개념에 기반하여 실시간으로 최단 우회 경로를 탐색할 수 있는 알고리즘을 제안한다. 이 알고리즘의 특징은 기존의 길이 개념 대신 주행시간을 적용한다. 주행시간은 “거리/차량 주행 속도”로 계산한다. 또한, 목적지까지 경로 탐색에 있어 불필요한 노드와 호들을 가급적 많이 제거하여 알고리즘 수행 시간을 단축시킬 수 있다. 따라서 제안된 알고리즘은 실시간 교통상황을 입력 받아 도로별 주행시간을 계산하여 최단 시간 내에 목적지까지 우회할 수 있는 경로를 제공할 수 있다.

2장에서는 네비게이션에서 Dijkstra 알고리즘으로 최단 경로를 찾는 과정과 문제점을 고찰해 본다. 3장에서는 주행시간에 기반한 빠른 최단경로 탐색 알고리즘을 제안한다. 또한, 사고나 정체 등으로 인해 돌발 상황이 발생할 경우 정체구간을 우회하여 목적지 까지 최단 경로를 탐색할 수 있는지 검증한다.

Ⅱ. 관련 연구와 연구 배경

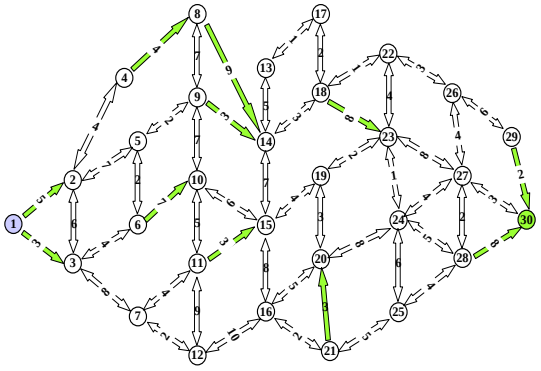
방향 그래프의 호 a는 방향성이므로 순서쌍 $w(x,y)$로 표기한다. 여기서, w는 가중치로 거리를 의미한다. 각 노드는 유출되는 호의 개수인 유출 차수 (Out- degree, d_o)와 유입되는 호의 개수인 유입 차수 (In- degree, d_i)를 갖는다. 임의의 노드 i의 d_i, d_o를 각각 $d_i(i)$와 $d_o(i)$로 표기한다. Dijkstra 알고리즘[5]은 그림 1과 같이 수행된다.

Dijkstra 알고리즘은 “특정 노드로부터 시작하여 최소 경로 길이를 갖는 노드를 한 번에 하나씩 선택하는 방식”으로, 특정 노드에 인접한 노드들과 이전에 계산된 노드들의 경로의 합이 최소가 되는 노드를 찾는다. 따라서 출발 노드로부터 시작하여 모든 노드들을 방문하는 최단경로를 찾기 때문에 알고리즘을 $n-1$번 수행한다.

방향 그래프 $D=(N,A)$와 가중치 함수 $w: A \rightarrow \mathbb{N}$에서, 출발 노드를 $s \in A$, $l(s)=0$로 설정하고 P에 저장
$x \neq s$인 모든 노드 $x \in N$에 대해 $l(x)=\infty$로 설정하고 Q에 저장
① Q에 있고, s에 인접한 모든 노드 $y \in A$ /* s의 유출 호
(1) $l(y)=min[l(y),\{l(s)+w(s,y)\}]$ 로 치환(단, 동일한 경로 길이인 경우 $l(s)$ 호 선택, $l(s)+w(s,y)$ 경로 길이가 적을 경우 (s,y) 선택)
(2) $l(n_1) \leq l(y)$인 노드 n_1 선택 (단, 최소 경로 길이 노드가 다수 존재시 하나만 선택)
(3) 호 (s,n_1)을 PSP (Partial Spanning Tree)에 저장, n_1을 Q에서 삭제하고 P에 저장
$n_i, i=1,2,\cdots$
② Q에 있고, n_i에 인접한 모든 노드 $y \in A$ /* n_i의 유출 호 중 P에 존재하면 무시함
(1) $l(y)=min[l(y),\{l(n_i)+w(n_i,y)\}]$ 로 치환(단, 동일한 경로 길이인 경우 $l(n_i)$ 호 선택, $l(n_i)+w(n_i,y)$ 경로 길이가 적을 경우 (n_i,y) 선택)
(2) $l(n_{i+1}) \leq l(y)$인 노드 n_{i+1} 선택 (단, 최소 경로 길이 노드가 다수 존재시 하나만 선택)
(3) 호 (n_i,n_{i+1})을 PSP (Partial Spanning Tree)에 저장, n_{i+1}을 Q에서 삭제하고 P에 저장
③ $i=i+1$, 그래프의 ST (Spanning Tree)를 얻을 때까지 ② 반복 수행
④ PSP에 저장된 호들에 대해 출발에서 목적지 노드까지의 경로 설정, $l(s,d)$를 SP로 결정

그림 1. Dijkstra 알고리즘
Fig. 1. Dijikstra's Algorithm

그림 2의 G_1 그래프에 대해 Dijkstra 알고리즘[5]을 적용하여 보자. G_1 그래프는 Lee[8]에서 인용되었으며, 이 도시는 30개의 노드 (교차로)와 41개의 양방향 도로, 11개의 일방통행로로 구성되어 통행 가능한 도로 (호)는 93개이다. 각 도로의 가중치는 거리 (단위 : Km)라 가정하자.



$|n|=30$, $|a|=93$

그림 2. G_1 그래프
Fig. 2. G_1 Graph

1 노드에서 출발하여 30 노드에 도착한다고 가정하여 보자. 출발 노드에서 목적지 노드까지의 최단거리를 갖는 경로를 Dijkstra 알고리즘으로 찾는 과정은 표 1에, 최단경로는 그림 3에 제시되어 있다. Dijkstra 알고리즘으로 찾은 1부터 30까지의 최단경로는 1-3-6-5-9-14-18-22-26-27-30으로 길이는 28 Km이다.

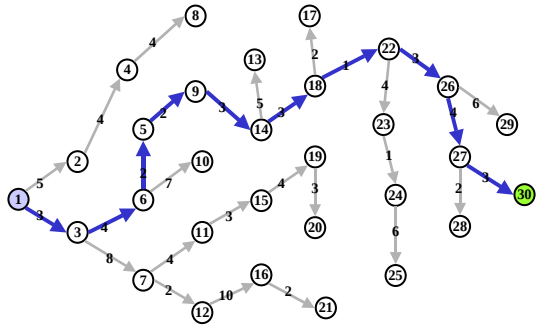
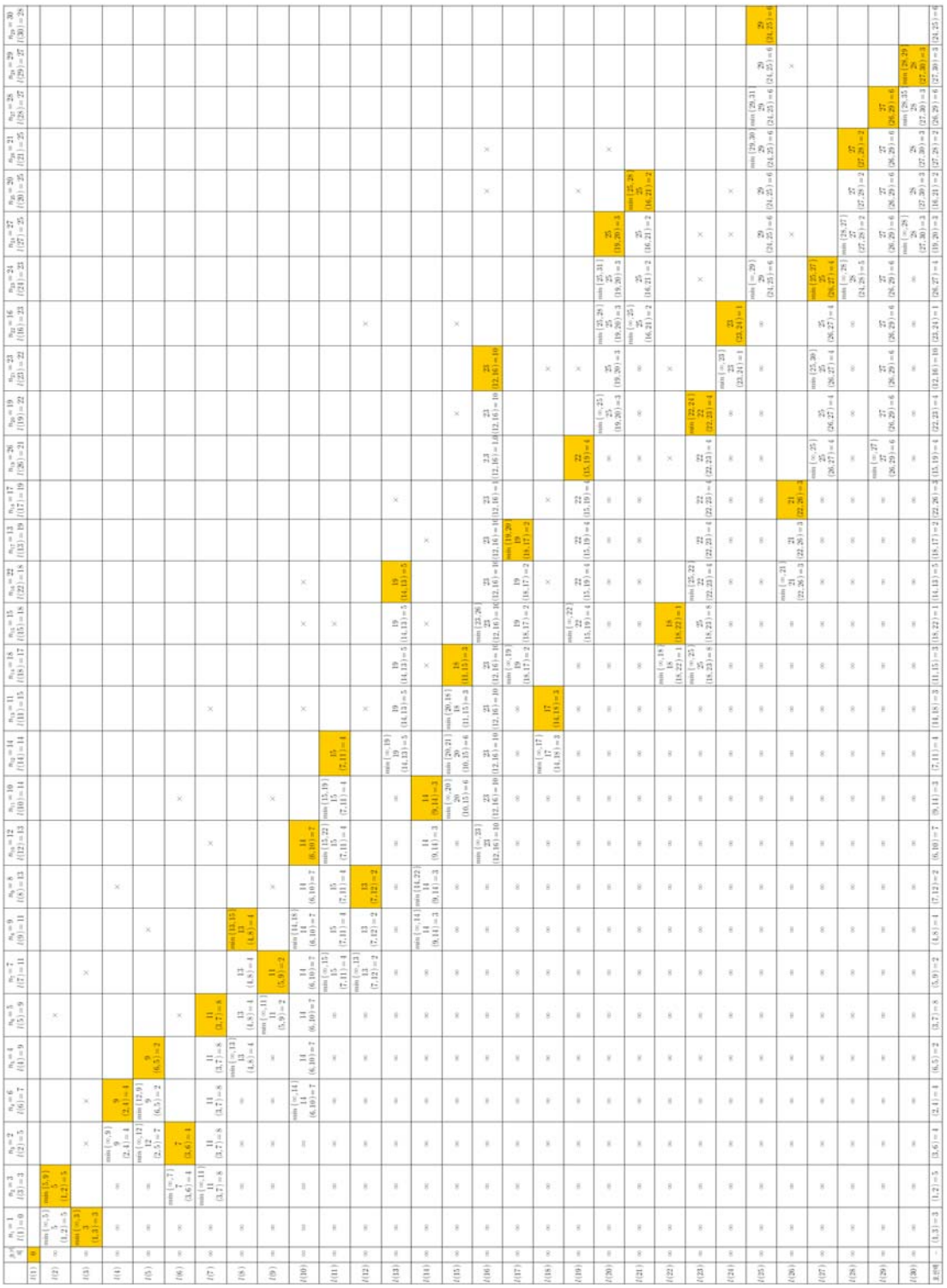


그림 3. G_1 그래프의 Dijkstra 알고리즘 최단경로
Fig. 3. The shortest path of Dijikstra's algorithm for G_1 graph

Dijkstra 알고리즘의 문제점은 첫 번째로, 알고리즘 수행에 많은 메모리를 요구하며 일반적으로 $n-1$개의 정점을 모두 방문해야 하기 때문에 그래프 크기가 클 경우 실시간으로 정보를 제공하기가 어렵다. 두 번째로, 네비게이션의 최단경로는 단일 출발점과 단일 목적지를 대상으로 하는 점대점 (Point-to-Point, P2P) 최단경로를 찾는 문제로 Dijkstra 알고리즘을 적용할 경우 최단 경로 상에 있지 않는 불필요한 노드들의 경로 길이를 계산하는 단점이 있다. 세 번째로, 기존의 모든 알고리즘은 호의 가중치로 거리를 적용하기 때문에 최단 경로 상에서 정체 또는 사고가 발생하여도 항상 동일한 경로를 제공하기 때문에 우회도로를 재탐색하지 못하여 이용자의 만족도를 높일 수 없다.

만약 네비게이션이 교통 흐름 상황 정보를 반영하여 목적지까지 최단 경로를 실시간으로 제공하고자 한다면, 호의 가중치로 기존의 거리 정보만을 활용할 수 없으며 거리를 속도로 나눈 주행시간 개념을 적용하여야만 한다. 또한, 빠른 시간 내에 목적지까지의 우회도로를 재탐색하기 위해 Dijkstra 보다 빠른 알고리즘 개발이 요구된다. 3장에서는 점대점 최단경로를 찾기 위해 호의 가중치로 주행시간 개념을 적용하며, 알고리즘 수행에 요구되는 메모리 크기도 줄이고, 수행속도를 월등히 향상시킬 수 있는 알고리즘을 제안한다.

표 1. G_1 그래프의 Dijkstra 알고리즘 적용
Table 1. Dijikstra's algorithm for G_1 graph



Ⅲ. 주행시간 기반 최단경로 탐색 알고리즘

1. 알고리즘 제안

방향그래프의 점대점 최단거리를 빠르게 탐색하기 위해 먼저, 그래프의 노드들을 S, P와 D의 3 그룹으로 분류한다. 여기서 S는 출발 노드, D는 목적지 노드이며, P는 출발과 목적지 노드를 제외한 그래프의 모든 경로노드들이라 하자. 임의의 호 $w(i,j)$를 단순히 (i,j)로 표기한다. 초기 조건으로 $n \times n$ 가중치 호 정방행렬을 작성하였다고 가정한다. 호의 가중치로 차량 주행속도를 적용한다. 차량 주행 속도는 DMB 방송의 TPEG 속도 정보, 도로교통정보 수집 CCTV의 차량 속도 정보, 이들 정보를 활용할 수 없는 도로는 시간대별 주행속도를 활용할 수 있다.

알고리즘은 출발 노드부터 레벨을 작성하고, 3 단계에 걸쳐 그래프를 단순화시키면서 최단경로를 선택하는 기법으로 레벨 설정, 역-주행 방향 도로 (호) 삭제, 노드 최소 가중치 호 선택, 레벨단위 최소 주행시간 선택, 최소 주행시간 경로 선택 과정을 수행한다. 제안된 알고리즘을 "*TSP* (Traveling time Shortest Path) 알고리즘"이라 하자.

***L*. 그래프의 레벨을 설정한다.** 출발 노드를 레벨 1로 설정하고, 출발노드와 이웃하는 노드들을 레벨 2로, 레벨 2 노드들과 이웃하는 노드들을 레벨 3으로 설정한다. 목적지 노드에 도달할 때까지 레벨을 순차적으로 증가시킨다. 레벨은 정방행렬의 각 행 노드에 대한 호들을 확인하면 쉽게 분류할 수 있다.

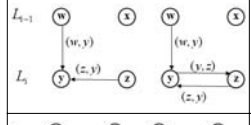
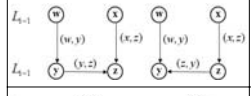
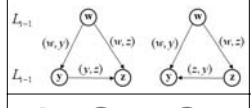
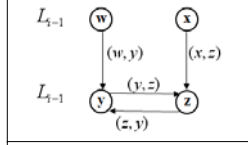
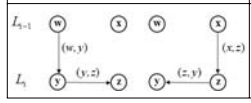
***P_1*. 역-주행 방향 호들을 삭제한다.** (1) 출발 노드에서 목적지 노드까지 주행함에 있어 역 방향 도로 (목적지에서 출발지로)는 주행하지 않으므로 역-레벨로 향하는 호를 삭제한다. (2) 그 결과, 유출 차수 d_o와 유입 차수 d_i가 0인 경로 노드가 존재하면 행과 열을 모두 삭제한다. $d_i = 0$인 노드는 결코 도달할 수 없는 노드이며, $d_o = 0$인 노드는 더 이상 다른 도로로 갈 수 없어 해당 노드의 모든 유입 호들을 삭제가 가능하다. 이 단계에서 그래프가 1차적으로 단순화된다.

***P_2*. 노드의 최소 가중치 호를 선택한다.** (1) 출발 (S) 노드의 행 (유출 호)과 목적지 노드 (D)의 열 (유입 호)에 대해 각각 첫 번째와 두 번째 최소 가중치 호(단, 동일 가중치 호 존재시 모두)를 선택한다. (2) 경로 노드 (P)들의 행 (유출 호)과 열 (유입 호)에 대해 최소 가중치 호를 1개씩 (단, 동일 가중치 호 다수 존재시 모두) 선택한다. 이와 같이 하는 이유는 경로 노드들의 큰 가중치 호는 최소 경로 길이에 기여를 하지 못하는 호들로 삭제가 가능하기 때문이다. 그러나 출발과 목적지 노드 연결 호들은 가중치가 최소가 되지 않더라도 경로 노드들에 의해 최단 경로로 선택 될 수 있으므로 최소 가중치 호를 2개 선택한다. (3) 유출 차수 d_o와 유입 차수 d_i가 0인 경로노드의 행과 열을 모두 삭제한다. 이 단계에서 그래프가 2차적으로 단순화된다.

***S_1*. 레벨 단위 최소 주행시간을 선택한다.** (1) L_2 레벨부터 시작하여 L_{i-1} 레벨 노드를 w, x, L_i 레벨 노드를 y, z라 하고 L_i 레벨의 제 노드 (Sibling) y, z간에 표 2의 관계가 있는 경우 해당 호를 삭제한다. (2) 유출 차수 d_o와 유입 차수 d_i가 0인 경로노드의 행과 열을 모두 삭제한다. 이 단계에서 그래프가 3차적으로 단순화된다.

***S_2*. 최소 주행시간 경로를 선택한다.** (1) 만약, 모든 노드들이 $d_i \le 1$와 $d_o \le 1$이면 출발노드부터 목적지 노드까지 최단경로가 결정된 상태이므로 알고리즘을 종료한다. (2) $d_o \ge 2$인 노드부터 $d_i \ge 2$인 노드까지 경로 중 최단경로를 선택하고, 선택되지 않은 최장 경로는 삭제한다. 이 과정을 모든 노드들이 $d_i \le 1$와 $d_o \le 1$가 될 때까지 반복적으로 수행한다.

표 2. 레벨 단위 최소 주행시간 선택
Table 2. Select of level-based minimum traveling time

경우	기준
	(x,z)는 없고 (w,y), (z,y)만 존재하는 경우 (z,y)호 삭제
	(w,y), (x,z), (y,z)가 존재하는 경우 만약, $(w,y) > (x,z)$이면 (y,z) 호 삭제
	(w,y), (w,z), (y,z)가 존재하는 경우 (y,z) 호 삭제
	(w,y), (x,z), (y,z), (z,y)가 존재하는 경우 $max\{(w,y)+(y,z), (x,z)+(z,y)\}$호 삭제
	(x,z)는 없고 (w,y), (y,z)만 존재하는 경우 (y,z)호 삭제 안함

TSP 알고리즘을 요약하여 정리한 결과는 그림 4에 제시하였다.

방향 그래프 $G=(N,A)$
N : 노드 (Nodes, n), A : 호 (Arcs, a), (i,j)
호의 가중치 (주행시간) = 거리/주행속도
각 호들에 대해 $|n|\times|n|$ 가중치 정방행렬 작성

L. 출발 노드부터 목적지 노드까지 **그래프 레벨 생성**

P_1. **역-레벨 호 삭제**, 경로 노드 중 $d_i=0$와 $d_o=0$인 노드의 호들 모두 삭제

P_2. **노드 최소 가중치 호 선택**

노드	행 (유출 호)	열 (유입 호)
출발지 노드 (S)	최소 가중치 호 2개 선택 (단, 동일 가중치 호 존재시 모두 선택)	×
경로 (중간) 노드 (P)	최소 가중치 호 1개 선택 (단, 동일 가중치 호 존재시 모두 선택)	최소 가중치 호 1개 선택 (단, 동일 가중치 호 존재시 모두 선택)
목적지 노드 (D)	×	최소 가중치 호 2개 선택 (단, 동일 가중치 호 존재시 모두 선택)

경로 노드 중 $d_i=0$와 $d_o=0$인 노드의 호들 모두 삭제

S_1. **레벨 단위 최소 주행시간 선택**
$d_i \geq 2$인 L_i 레벨 경로 노드 y에 대해 L_{i-1} 레벨 노드 x와 L_i 레벨 형제노드 (Sibling) z의 L_{i-1} 레벨 노드 x간 비교
(1) (x,z)는 없고 (w,y), (z,y)만 존재하면 (z,y)호 삭제
(2) (w,y), (x,z), (y,z)가 존재하고, $(w,y) > (x,z)$이면 (y,z)호 삭제
(3) (w,y), (w,z), (y,z)가 존재하면 (y,z) 호 삭제
(4) (w,y), (x,z), (y,z), (z,y)가 존재하면 $max\{(w,y)+(y,z),\ (x,z)+(z,y)\}$호 삭제
(5) (x,z)는 없고 (w,y), (y,z)만 존재하면 (y,z)호 삭제 안함
경로 노드 중 $d_i=0$와 $d_o=0$인 노드의 호들 모두 삭제

S_2. **최소 주행시간 경로 선택**
IF 모든 노드들이 $d_o \leq 1$와 $d_i \leq 1$ THEN 알고리즘 종료.
ELSE IF $d_0 \geq 2$ 노드부터 $d_i \geq 2$ 노드까지 경로 최소 주행시간 선택 (만약, 동일 경로 시간은 노드 수가 적은 경로 선택)
$d_i \geq 2$인 노드가 없을 때까지 반복 수행

그림 4. *TSP* **알고리즘**
Fig. 4. TSP Algorithm

2. 정상 도로 상황에서의 적용성 평가

그림 2의 G_1 그래프에서 10,11,14,15,18,19,20,23, 24,27 노드에서 병목현상이 발생한다고 가정하여 보자. 이 도시의 도로는 제한속도 60 Km/h로 구성되어 있으며, 일부 지점의 병목현상으로 인해 주행속도는 정상소통 지점에서 병목지점으로의 차량 주행 속도는 30 Km/h, 병목지점 사이는 20 Km/h, 병목지점에서 정상소통 지점으로의 주행 속도는 40 Km/h, 정상소통 지점 간에는 50 Km/h라 가정하자. 이 결과를 반영하면 모든 도로의 주행속도는 그림 5와 같다.

그림 2의 G_1 그래프의 거리를 그림 5의 주행속도로 나눈 각 도로의 주행시간 (분:초)은 그림 6과 같다. 그림 6의 주행시간 데이터를 적용하여 *TSP* 알고리즘을 수행한 결과는 그림 7과 같다.

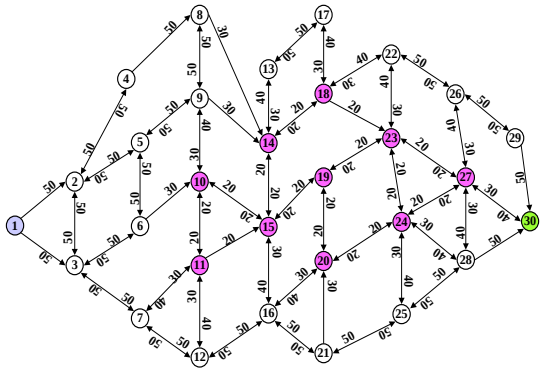


그림 5. 도로별 주행속도
Fig. 5. Traveling speed for each road

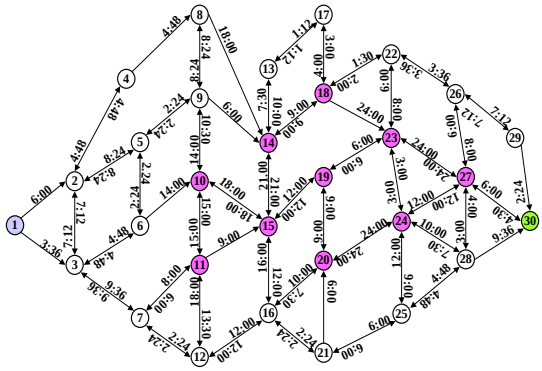
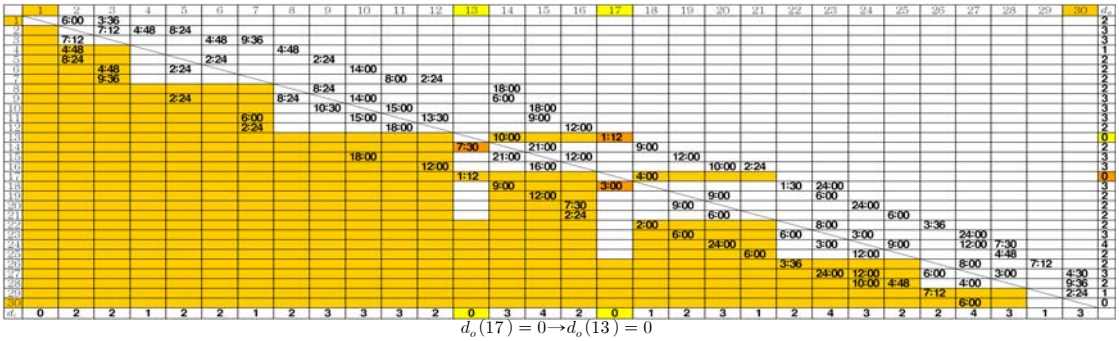


그림 6. 도로별 주행시간
Fig. 6. Traveling time for each road

그림 7에서 G_1 그래프의 레벨은 표 3과 같이 구해진다.

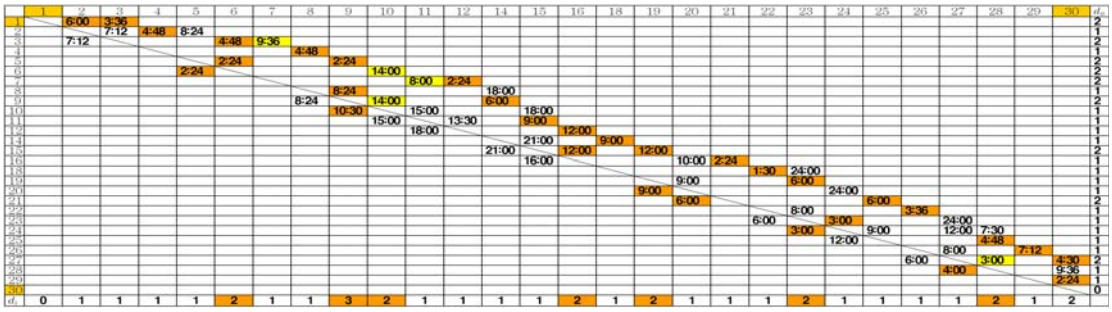
표 3. G_1 **그래프의 레벨**
Table 3. Levels of G_1 graph

레벨	노드	인접 노드
1	1	2, 3
2	2, 3	4, 5, 6, 7
3	4, 5, 6, 7	8, 9, 10, 11, 12
4	8, 9, 10, 11, 12	14, 15, 16
5	14, 15, 16	13, 18, 19, 20, 21
6	13, 18, 19, 20, 21	17, 22, 23, 24, 25
7	17, 22, 23, 24, 25	26, 27, 28
8	26, 27, 28	29, 30
9	29, 30	∅

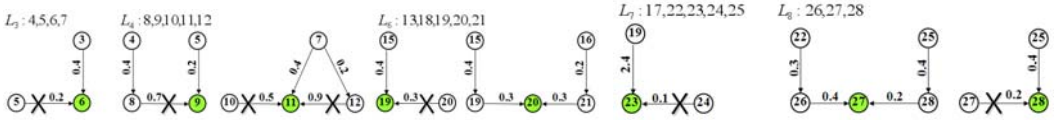


$d_o(17)=0 \rightarrow d_o(13)=0$

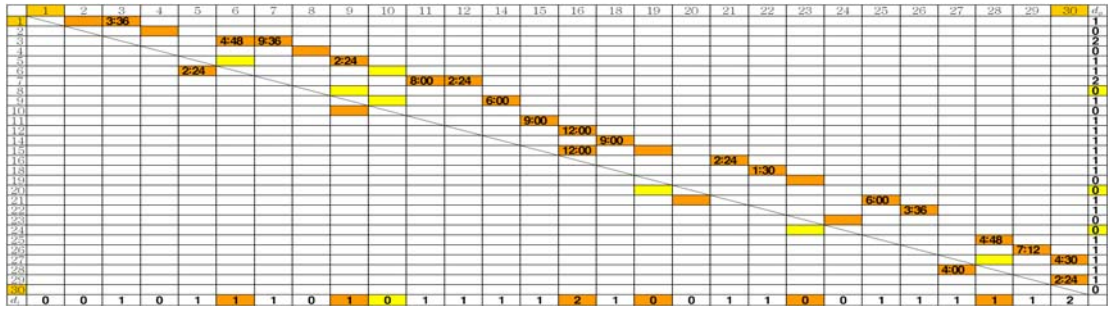
(a) P_1 단계 수행



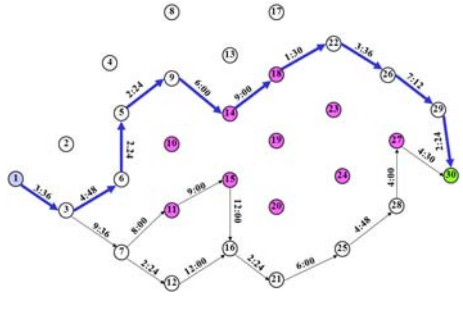
(b) P_2 단계 수행



$d_i(10)=0, d_o(8)=0 \rightarrow d_o(4)=0 \rightarrow d_o(2)=0, d_o(20)=0, d_o(24)=0 \rightarrow d_o(23)=0 \rightarrow d_o(19)=0$



(c) S_1 단계 수행



7-16 : $min\{7 \xrightarrow{8:00} 11 \xrightarrow{9:00} 15 \xrightarrow{12:00} 16 = 29:00, 7 \xrightarrow{2:24} 12 \xrightarrow{12:00} 16 = 14:24\} \Rightarrow (7,11), (11,15), (15,16)$ 삭제

3-30 : $min\{3 \xrightarrow{4:48} 6 \xrightarrow{2:24} 5 \xrightarrow{2:24} 9 \xrightarrow{6:00} 14 \xrightarrow{9:00} 18 \xrightarrow{1:30} 22 \xrightarrow{3:36} 26 \xrightarrow{7:12} 29 \xrightarrow{2:24} 30 = 39:18, 3 \xrightarrow{9:36} 7 \xrightarrow{2:24} 12 \xrightarrow{12:00} 16 \xrightarrow{2:24} 21 \xrightarrow{6:00} 25 \xrightarrow{4:48} 28 \xrightarrow{4:00} 27 \xrightarrow{4:30} 30 = 45:42\} \Rightarrow (3,7), (7,12), (12,16), (16,21), (21,25), (25,28), (28,27), (27,30)$ 삭제

(d) S_2 단계 수행

그림 7. G_1 그래프의 TSP 알고리즘 최단경로

Fig. 7. The shortest path for G_1 graph with TSP algorithm

Dijkstra 알고리즘은 1번 노드부터 나머지 29개 노드들에 대한 최단 거리를 계산하기 위해 29개 호 들을 선택한다. 반면에, *TSP* 알고리즘은 10개의 노드를 삭제하고 20개 노드들을 연결하는 21개 호 들을 대상으로 7-16과 3-30 경로만을 비교하여 최단 경로를 선택한다. 따라서 *TSP* 알고리즘은 목적지까지 최단 경로를 빠르게 계산할 수 있음을 알 수 있다.

3. 특정 도로 사고, 지체시 우회 탐색 적용성 평가

그림 2의 G_1 그래프에서 현재 (1,3) 도로를 운행 중에 (14,18) 도로에서 사고가 발생하여 (14,18) 도로가 폐쇄 (삭제)되고, 이로 인해 (18,14)과 14와 18 노드로 유입되는 (8,14),(9,14), (13,14),(15,14), (17,18)과 (22,18) 도로에 정체 현상이 발생하여 주행속도는 5 Km/h라 가정하여 보자. 이 경우 각 도로별 주행 시간은 그림 8과 같다.

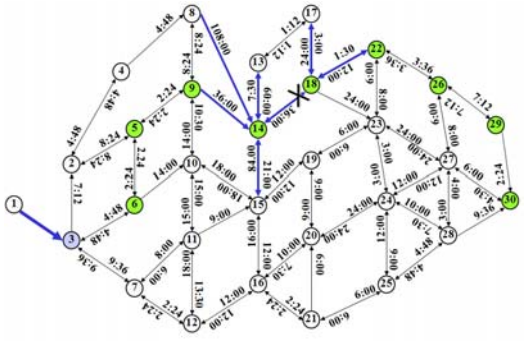


그림 8. 돌발상황 발생 각 도로의 주행시간
Fig. 8. Traveling time for each road with Emergency situation

현재 (1,3) 도로를 운행 중에 있으므로 출발 노드는 ③ 노드가 된다. 따라서 출발 노드로 유입되는 (2,3)은 삭제된다.

30번 노드까지의 최단 경로를 운행 중에 최단 경로 상에 있는 (14,18) 도로에서 사고가 발생하였음에도 불구하고 Dijkstra 알고리즘의 거리기반 최단경로 탐색 알고리즘은 도로 통행 속도 개념을 적용하지 않아 계속 (14,18) 도로를 지나는 경로 정보를 제공한다. 결국, (14,18) 도로를 통과하지 못하여 목적지에 도달하지 못하는 상황이 발생한다. 반면에, 주행시간 기반의 *TSP* 알고리즘을 적용할 경우 (14,18) 도로를 우회할 수 있는지 고찰해보자.

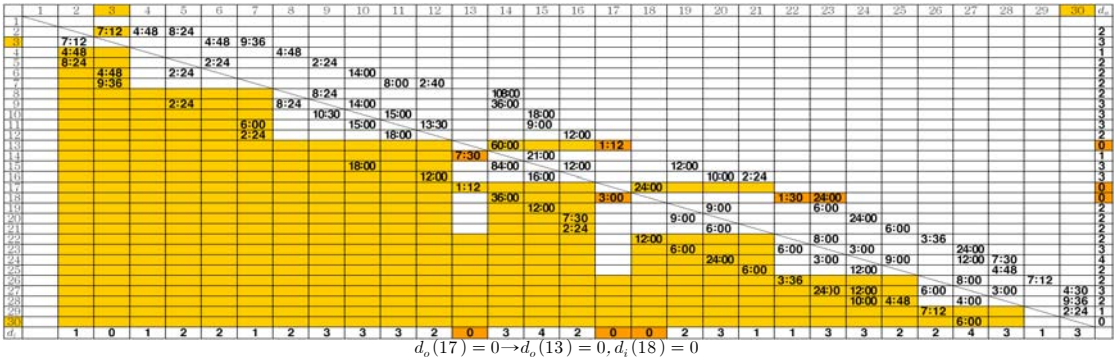
그림 8의 주행시간에 대해 *TSP* 알고리즘을 적용한 결과는 그림 9와 같다. *TSP* 알고리즘은 돌발생황 발생 시 정체 구간을 우회하여 목적지 까지 최단 경로를 재설정할 수 있음을 알 수 있다. 또한, 이 경우 16개 노드와 19개 호 만을 대상으로 7-16 경로에 대한 최소 경로를 결정한 결과 목적지까지의 최단 경로를 빠르게 선택할 수 있었다.

Ⅳ. 결론 및 향후 연구과제

본 논문에서는 네비게이션에 일반적으로 적용되고 있는 Dijkstra 알고리즘의 문제점을 고찰해보고 새로운 최단 경로 탐색 알고리즘을 제안하였다. Dijkstra 알고리즘은 출발 노드부터 시작하여 모든 노드들을 한번에 하나씩 방문하는 방법으로 알고리즘 수행속도가 느린 단점이 있다. 또한 거리정보만을 대상으로 하기 때문에 복잡한 도시의 지체나 사고 등으로 정체구간이 발생할 경우 우회할 수 있는 탐색 기능을 제공하지 못한다. 이러한 문제점으로 인해 최근 들어 TPEG 네비게이션이 상용화되고 있지만 실시간 교통정보를 반영하여 우회도로를 탐색하여 제공하는데 한계가 있다.

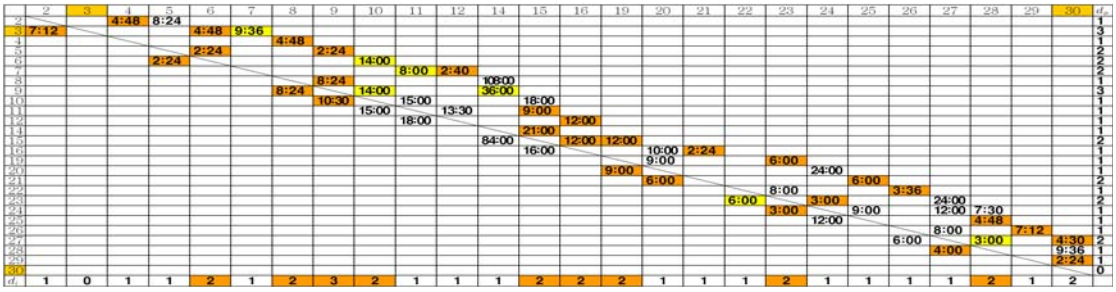
본 논문은 TPEG의 기능을 최대한 발휘하도록 빠른 시간 내에 최단경로를 탐색하는 알고리즘을 제안하였다. 제안된 알고리즘은 그래프의 출발 노드부터 목적지 노드까지의 레벨을 설정하고, 불필요한 노드와 호들을 3단계로 제거하여 최소한의 노드와 호들만을 대상으로 최단경로를 선택함으로서 실시간으로 탐색 정보를 제공할 수 있는 장점을 갖고 있다.

제안된 알고리즘은 기존의 거리 정보 대신 주행 시간 개념을 도입함으로서 실시간 교통정보를 활용할 수 있는 효과를 얻었다. 이는 정상 상태의 교통 흐름과 돌발 상황 발생을 가정한 2개 경우에 대해 검증하였다. 제안된 알고리즘 검증 결과 최단 시간 내에 목적지 까지 도달 가능한 경로를 탐색하는데 성공하였다. 제안된 알고리즘을 추후 실제 TPEG 네비게이션에 도입하여 고객의 만족도를 얼마나 높일 수 있는지 검증할 계획이다.

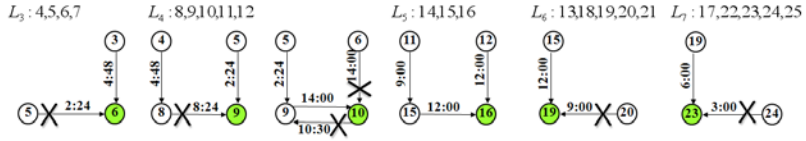


$d_o(17)=0 \rightarrow d_o(13)=0, d_i(18)=0$

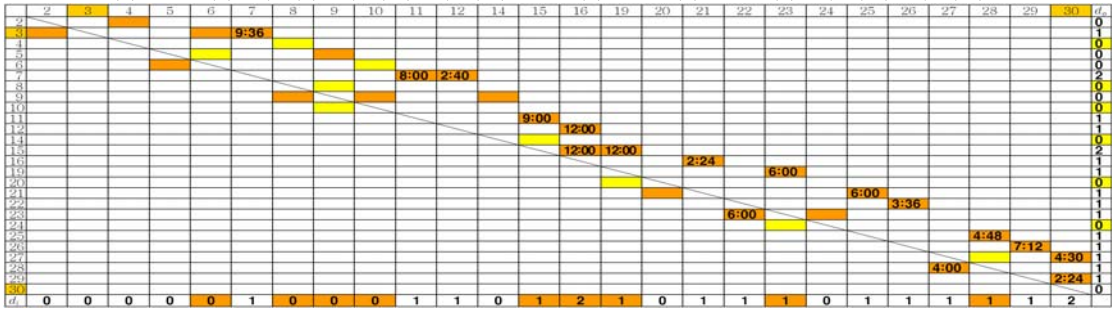
(a) P_1 단계 수행



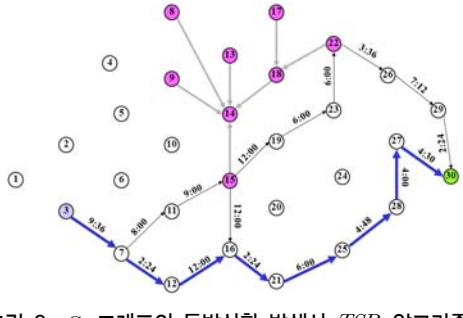
(b) P_2 단계 수행



$d_o(4)=0 \rightarrow d_o(2)=0, d_o(8)=0, d_o(10)=0, d_o(14)=0 \rightarrow d_o(9)=0 \rightarrow d_o(5)=0 \rightarrow d_o(6)=0, d_o(20)=0, d_o(24)=0$



(c) S_1 단계 수행



$7-16: min\{7 \xrightarrow{8:00} 11 \xrightarrow{9:00} 15 \xrightarrow{12:00} 16 = 29:00, 7 \xrightarrow{2:24} 12 \xrightarrow{12:00}$

$16 = 14:24\} \Rightarrow (7,11), (11,15), (15,16)$ 삭제

$(15,19), (19,23), (23,22), (22,26), (26,29), (29,30)$ 삭제

(d) S_2 단계 수행

그림 9. G_1 그래프의 돌발상황 발생시 TSP 알고리즘 최단경로

Fig. 9. The shortest path of TSP algorithm for G_1 graph with emergency situation

참 고 문 헌

[1] M. Abboud, L. Mariya, A. Jaoude, and Z. Kerbage, "Real Time GPS Navigation System," 3rd FEA Student Conference, Department of Electrical and Computer Engineering, American University of Beirut, 2004.

[2] T. H. Cormen, C. E. Leserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms," 2nd Edition, MIT Press and McGraw-Hill, 2001.

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

[4] Wikipedia, "Dijkstra's Algorithm," http://en.wikipedia.org/wiki/Dijkstra_algorithm, Wikimedia Foundation Inc., 2012.

[5] J. Misra, "A Walk Over the Shortest Path: Dijkstra's Algorithm Viewed as Fixed-Point Computation," Department of Computer Science, University of Texas at Austin, USA, 2000.

[6] Y. T. Lim and H. M. Kim, "A Shortest Path Algorithm for Real Road Network Based on Path Overlap," Department of Civil Engineering, Institute of Transportation Studies, University of California, Irvine, USA, http://www.its.uci.edu/~hyunmyuk/library/(2005)20EAST(k-path). pdf, 2005.

[7] F. B. Zhan, "Three Fastest Shortest Path Algorithms on Real Road Networks: Data Structures and Procedures," Journal of Geographic Information and Decision Analysis, Vol. 1, No. 1, pp. 69-82, 1997.

[8] K. S. Lee, "Shortest Path Algorithm," http://www.geocities.com/leekinseng1/

저자 소개

이 상 운

- 1987년: 한국항공대학교 항공전자공학과 (학사)
- 1997년: 경상대학교 컴퓨터과학과 (석사)
- 2001년 : 경상대학교 컴퓨터과학과 (박사)
- 2003년 : 강원도립대학 컴퓨터응용과 전임강사
- 2004년 ~ 현재 : 강릉원주대학교 멀티미디어공학과 부교수

<관심분야 :소프트웨어 프로젝트 관리, 개발 방법론, 소프트웨어 척도, 분석과 설계 방법론, 시험 및 품질보증, 소프트웨어 신뢰성, 신경망, 뉴로-퍼지, 그래프 알고리즘>

- e-mail : sulee@gwnu.ac.kr