모바일 컴퓨팅 환경에서의 디지털 로드맵 데이타베이스를 위한 근접 최단 경로 재계산 방법

(An Approximate Shortest Path Re-Computation Method for Digital Road Map Databases in Mobile Computing Environments)

김 재 훈 † 정 성 원 †† 박 성 용 ††
(Jae Hun Kim) (Sungwon Jung) (Sungyong Park)

요 약 모바일 컴퓨팅의 상업적인 응용분야로서, 지능형 교통정보시스템(ITS: Intelligent Transport Systems)의 한 분야인 첨단 여행자 정보시스템(ATIS: Advanced Traveler Information Systems)이 있다. ATIS에서 가장 중요한 모바일 컴퓨팅 태스크는 현재 위치에서 목적지까지의 최단 경로를 계산하는 일이다. 본 논문에서는 ATIS의 동적 경로 안내 시스템(DRGS: Dynamic Route Guidance System)에서 발생하는 최단 경로 재 계산 문제에 대해서 연구하였다. 이 문제는 동적인 교통상태에 따라 디지털 로드맵 상의 간선 비용이 빈번하게 갱신되기 때문에 발생한다. 기존의 방법들은 처음부터 최단 경로를 재 계산하거나, 또는 단지 비용의 변화가 일어난 간선 상에 있는 양 끝 노드 사이에 대해서만 최단 경로를 재 계산할 뿐이다. 이러한 방법은 앞서 계산된 최단 경로에 대한 정보를 이용하지 않는다는 점에서 모두 비효율적이다. 이에, 본 논문에서는 효율적인 동적 윈도우 기반의 근접 최단 경로 재 계산 방법(A Dynamic Window-Based Approximate Shortest Path Re-Computation Method)을 제안한다. 이 방법은 앞서 계산된 최단 경로의 정보를 이용하여 최적의 최단 경로에 상당히 근접한 경로를 매우 빠른 시간 안에 계산해 낸다. 우리는 제안한 방법을 이론적으로 분석한 다음 이를 격자 그래프 및 실제 디지털 로드맵 상에 구현하여 철저한 실험적인 성능 분석을 하였다.

키워드 : 모바일 컴퓨팅, 디지털 로드맵, 근접최단경로, 지능형교통정보시스템, 격자 그래프

Abstract One of commercial applications of mobile computing is ATIS(Advanced Traveler Information Systems) in ITS(Intelligent Transport Systems). In ATIS, a primary mobile computing task is to compute the shortest path from the current location to the destination. In this paper, we have studied the shortest path re-computation problem that arises in the DRGS(Dynamic Route Guidance System) in ATIS where the cost of topological digital road map is frequently updated as traffic condition changes dynamically. Previously suggested methods either re-compute the shortest path from scratch or re-compute the shortest path just between the two end nodes of the edge where the cost change occurs. However, these methods are trivial in that they do not intelligently utilize the previously computed shortest path information. In this paper, we propose an efficient approximate shortest path re-computation method based on the dynamic window scheme. The proposed method re-computes an approximate shortest path very quickly by utilizing the previously computed shortest path information. We first show the theoretical analysis of our methods and then present an in-depth experimental performance analysis by implementing it on grid graphs as well as a real digital road map.

Key words : Mobile Computing, Digital Road Maps, Approximate Shortest Path, ITS, Grid Graphs

· 본 논문은 2002년도 산업자원부 차세대신기술개발사업과 서강대학교 산업기술연구소의 지원을 받아 수행되었음.
† 비 회 원 : LG전자 연구원
freeso@mclab.sogang.ac.kr
†† 종신회원 : 서강대학교 컴퓨터학과 교수
jungsung@ccs.sogang.ac.kr
parksy@ccs.sogang.ac.kr
논문접수 : 2002년 3월 9일
심사완료 : 2003년 2월 19일

1. 서 론

무선통신 기술의 발달과 함께, 고성능 휴대용 컴퓨터의 등장은 모바일 컴퓨팅의 구현을 현실화시키고 있다. 이러한 모바일 컴퓨팅의 상업적인 응용분야로서 지능형 교통정보시스템(ITS: Intelligent Transport Systems)의 한 분야인 첨단 여행자 정보시스템(ATIS: Advanced Traveler Information Systems)이 있다. ATIS에서 가장 중요한 모바일 컴퓨팅 테스크는 현재 위치에서 목적지까지의 최단 경로를 계산하는 일이다. 여기서, 중요한 문제 중 하나는 매우 큰 디지털 로드맵 상에서 최단 경로를 찾는데 상당량의 계산 시간이 요구된다는 것이다. 예를 들어, 간격이 100 *feet*이고 100 *mi* × 100 *mi*인 작은 지도를 저장하는데 약 2.4 Gbyte가 필요하다[1,2]. ATIS는 실시간 이동 시스템이기 때문에, 어떤 최단 경로를 제한된 시간 안에 계산하는 일은 매우 중요하다.

디지털 로드 맵 데이타베이스 분야에서 최단 경로 계산 문제에 중점을 둔 두 가지 형태의 연구 결과가 문헌을 통해 알려져 있다. 그중 하나는 대략적인 최단 경로를 신속히 제공하는 데이타베이스 체계를 개발한 것이다[3,4,5,6,7]. 다른 하나는 최단 경로 계산을 빠르게 할 수 있는 디지털 로드 맵을 구성하기 위한 효율적인 데이타베이스 구성 방법을 개발한 것이다[1,8,9,10,11].

본 논문에서는 최단 경로 재 계산 문제에 대해서 연구하였다. 이 문제는 디지털 로드 맵 상의 간선 비용이 동적인 교통 상태에 따라 빈번하게 갱신되고 있는 ATIS의 동적 경로 안내 시스템(DRGS: Dynamic Route Guidance System)에서 발생한다. 교통 상태의 변화로는 예를 들어, "교통 장애의 발생", "교통 사고로 인한 도로 차단" 등이 있다. 결과적으로, 여행 시 처음 계획된 최단 경로는 원치 않은 경로 비용의 갱신으로 인해 최종적으로 얻어지는 최단 경로와 다를 수 있다.

그러므로, 효율적으로 최단 경로를 재 계산하는 일은 DRGS의 성공적인 수행을 위해 필수적이다. 현재 사용되고 있는 두 가지 방법은 처음부터 최단 경로를 재계산하거나 또는 비용의 변화가 일어난 두 정점 사이의 최단 경로만을 재 계산할 뿐이다. 일반적으로, 비용의 변화가 일어난 후 새롭게 재 계산된 최단 경로는 이전의 최단 경로와 크게 다르지 않다. 이러한 점에서, 위의 두 가지 방법들은 앞서 계산된 최단 경로의 정보를 재사용하지 않기 때문에 분명히 비효율적이다. 앞서 계산된 최단 경로의 정보를 이용하여 그 최단 경로를 부분적으로 바꾸는 방법은 새로운 현재 위치에서 목적지까지의 완전한 최단경로를 계산하는 것보다 더 빠를 것이다.

본 논문에서는 앞서 계산된 최단 경로에 대한 정보를 이용하는 효율적인 동적 윈도우 기반의 근접 최단 경로 재 계산 방법(A Dynamic Window-Based Approximate Shortest Path Re-Computation Method)을 제안한다. 이 방법은 앞서 계산된 최단 경로의 정보를 이용하여, 오직 간선 비용의 변화에 영향을 받은 부분만을 재 계산한다. 실제로, 이 방법은 최적의 최단 경로를 보증하지는 않지만, 최적에 상당히 가까운 경로를 제공한다. 제안한 방법의 기본적인 가정은 많은 실제 교통상황에서, 운전자는 경로 재 계산에 드는 매우 큰 시간 비용을 감안할 때, 최적으로 재 계산된 최단 경로를 제공받는 것에 관심이 없을 수 있다는 것이다. 오히려, 그들은 신속히 재 계산되는 대략적인 최단 경로를 얻고자 한다.

논문의 나머지 부분은 다음과 같이 구성된다. 2장에서는 경로 질의에서 사용되는 A* 알고리즘과 OTO(One-To-One) 알고리즘을 간략히 논의한다. 3장에서는 디지털 로드 맵의 전형적인 예인 2차원 격자 그래프에서 최단 경로 상의 하나의 간선에 대한 갱신(단일 장애) 영향을 실험적으로 분석한다. 4장에서는 하나의 간선 비용 갱신에 대한 동적 윈도우 기반의 근접 최단 경로 재 계산 방법을 제안한다. 그리고 제안한 방법의 효율성을 여러 경우에 대한 실험을 통하여 검증한다. 5장에서는 여러 간선 비용의 갱신(다중 장애)을 다루기 위해, 제안한 방법을 확장한다. 마지막으로, 6장에서 결론을 맺는다.

2. A*와 OTO 최단 경로 알고리즘

최단 경로 계산의 주된 수행 오버헤드(overhead)는 검색해야 할 탐색 공간(search space)의 크기에 의해서 좌우된다. 이런 이유에서, Dijkstra 알고리즘은 그 탐색 공간의 크기가 크기 때문에 디지털 로드 맵 상에서 SPSP(Single Pair Shortest Path)를 계산하는데 적합하지 않다. 이에 반해, A* 알고리즘은 탐색 공간을 줄이기 위해 노드 u와 d사이의 최단 경로 비용(유클리드 거리(Euclidean Distance))을 평가하는 평가 함수(estimator function) $f(u,d)$를 사용한다. V가 노드 집합, E가 간선 집합, L이 비용의 집합일 때, 방향 그래프 $G(V,E,L)$가 디지털 로드 맵을 나타낸다고 하자. 그림 1의 pseudo 코드는 $G(V,E,L)$상에서 노드 s와 노드 d사이의 SPSP를 찾는 A* 알고리즘을 나타낸 것이다.

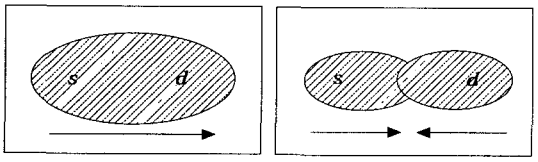
Mohr와 Pasche는 각각 출발지와 목적지를 루트

```
Procedure A*(V, E, s, d, f);
1. For each node u ∈ V, C(u) = ∞.
2. Let C(s) = 0, frontierSet = {s}, exlporedSet = ∅.
3. Select a node u in frontierSet for which C(u) + f(u,d) is minimum
4. frontierSet = fontierSet - {u},  exploredSet = exploredSet ∪ {u}
5. if (u==d) then C(u) is the shortest path cost and stop
6. for every edge (u,v) in E
        if C(u) > C(u) + L(u,v)  then C(u) = C(u) + L(u,v)
            frontierSet = frontierSet ∪ {v} if v ∉
                          (frontierSet ∪ exploredSet)
7. Go to step 3
```

그림 1 A* 알고리즘

(root)로 가지는 두 개의 트리를 선택적으로 확장해 나감으로써 최단 경로를 찾는 OTO(One-To-One) 알고리즘을 제안했다[12]. 여기서, OTO는 각 트리를 확장하는데 있어 A* 알고리즘을 사용한다. OTO 알고리즘에 대한 보다 자세한 설명은 [12]에 기술되어 있다.

OTO가 A*보다 더 뛰어나다는 사실을 보여주기 위해, 그림 2에서 두 알고리즘의 탐색된 공간을 도식적으로 표현하였다. 여기서, 출발지와 목적지는 각각 s와 d이며, 타원은 각 알고리즘에서 검색한 탐색 공간을 나타내고, 화살표는 탐색 공간을 검색하는 방향을 나타낸다. OTO 알고리즘이 양쪽 끝에서부터 두 개의 타원을 그리며 탐색 공간을 검색하기 때문에, 검색된 탐색 공간의 크기는 A*에서 검색된 탐색 공간의 크기보다 작다. 이 사실은 그림 2에서 명백히 입증된다.



(a) A*에서 검색되는 공간 (b) OTO에서 검색되는 공간

그림 2 A*와 OTO에 대한 검색된 탐색 공간의 상대적인 크기

3. 최단 경로상의 간선 비용 갱신에 대한 영향 분석

이 장에서는 최단 경로 상에 하나의 간선 비용이 갱신되었을 경우, 이 갱신이 최단 경로에 어떤 영향을 미치는지 분석한다. 이를 위해서, 2장에서의 OTO 알고리즘과 2차원 격자 그래프를 사용한다. 2차원 격자 그래프는 디지털 로드 맵의 전형적인 예라 할 수 있다[2,13].

3.1 최단 경로 재 계산에서 경로 중복도(Degree of Path Similarities) 측정

이 절에서는 최단 경로를 재 계산하는 과정에서 경로 중복도를 실험한다. 이 경로 중복도로부터 간선 비용의 갱신이 최단 경로에 얼마나 영향을 미치는지 분석할 수 있다. 먼저, 본 논문에서 사용될 다음 두 정의를 살펴본다.

정의 1. $P_s(s,d)$가 $G(V,E,L)$에서 출발지 s로부터 목적지 d까지의 최단 경로라고 하자. 이때, $P_s(s,d)$는 $z_0=s$, $z_m=d$인 노드 수열 $(z_0, z_1, z_2, \ldots, z_{m-1}, z_m)$와 같이 순차적으로 표현할 수 있다. 또, $0 \le i \le m-1$일 때, $P_s(s,d)$ 상의 어떤 간선 (z_i, z_{i+1})의 비용이 갱신된 후, s에서 d까지 재 계산된 새로운 최단 경로를 $P_n(s,d)$라고 하자. 마찬가지로, $P_n(s,d)$는 노드 수열 $(w_0, w_1, w_2, \ldots, w_{n-1}, w_n)$와 같이 순차적으로 표현할 수 있다. □

정의 2. $P_s(s,d)$와 $P_n(s,d)$에 대해서 다음 조건을 만족하는 서로 다른 부분 수열 $Q_0, Q_1, \ldots Q_t (0 \le t)$가 존재한다고 하자. : $0 \le k \le t$, $1 \le z$, $i+z < m$, $j+z < n$일 때, $Q_k = (x_i, x_{i+1}, \ldots x_{i+z}) = (y_j, x_{j+1}, \ldots x_{j+z})$이며 여기서, $x_{i-1} \neq y_{j-1}$와 $x_{i+z+1} \neq y_{j+z+1}$을 만족한다. 그리고, 부분 수열 Q_k를 구성하는 노드 개수가 $|Q_k|$일 때, $P_s(s,d)$와 $P_n(s,d)$사이의 경로 중복도(the Degree of Path Similarities) $DPS(P_s, P_n)$는 다음과 같다:

$$DPS(P_s, P_n) = \sum_{k=0}^{t} |Q_k| \times 100/m \ (\%). \quad \square$$

본 논문에서는 여러 종류의 격자 그래프와 최단 경로들을 생성하기 위해서, 다음과 같은 세 가지 파라미터들을 사용한다.

- Δ : 격자 그래프 $G(V,E,L)$에 대한 간선 비용의 다양성(variations of edge costs). 예를 들어, $\Delta=x\%$인 격자 그래프의 간선 비용은 $[100,100+x]$ 범위에서 임의적으로 할당된다.
- θ : 출발지 s와 목적지 d가 이루는 각도. 예를 들어, s=(10,10), d=(20,20)이면 θ=45.
- $f(s,d)$: 출발지 s와 목적지 d간의 유클리드 거리. 즉, $s=(x_1, y_1)$, $d=(x_2, y_2)$일 경우, $f(s,d)=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$이다.

이제, 실험에서 사용될 여러 격자 그래프와 최단 경로들을 생성한다. 먼저, 그래프 크기가 $|V|$=50×50, $|E|$=4×50×49로 모두 같으나, 각 그래프의 Δ값이 각각 20%, 40%, 80%, 160%, 240%, 320%, 480%인 7 종류의 격자 그래프를 가정해 본다. 이때, 각 격자 그래프에 대해서 간선 비용을 임의적으로 할당하는데 사용된 시드(seed)값을 서로 다르게 하여 100개의 격자 그래프를 생성한다. 그러면, 총 7×100개의 격자 그래프가 생성되었다. 그 다음으로,

각 7×100개의 격자 그래프 상에서 5개의 최단 경로를 선택한다. 즉, θ=0, 15, 30, 45, 90인 5개의 노드 쌍을 선택하였다. 여기서, 모든 노드 쌍에 대해서 $f(s,d)$=30이다. 이제, 각 격자 그래프 상의 5개의 노드 쌍에 대한 5개의 최단 경로를 계산한다. 그리고, 각 최단 경로의 5%, 10%, ..., 90%, 95% 위치에 해당되는 간선 비용을 한번에 하나씩만 갱신하면서(단일 장애), 각 위치의 간선 비용 갱신에 대한 $DPS(P_s, P_n)$을 측정한다. 여기서, 측정된 $DPS(P_s, P_n)$는 7 종류의 격자 그래프 별로 100개씩 평균화한다. 그 결과를 그림 3에서 보인다.

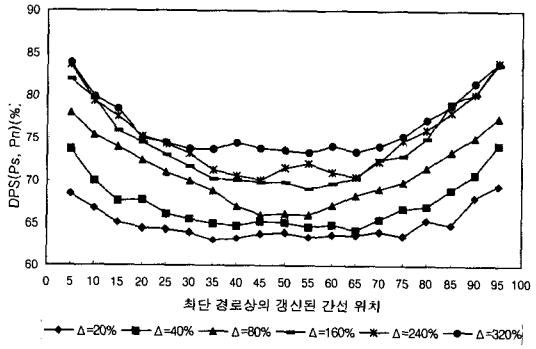


그림 3 최단 경로상의 각 위치에 대한 $DPS(P_s, P_n)$

위 결과로부터, 최단 경로상의 갱신 영향에 관해 다음의 세 가지 특성을 발견할 수 있다:

1. $DPS(P_s, P_n)$는 Δ값에 따라서 60%에서 90%까지 변한다.
2. Δ값이 20%에서 320%까지 증가함에 따라, $DPS(P_s, P_n)$도 역시 증가한다.
3. $DPS(P_s, P_n)$는 최단 경로의 양끝에서 최대이고, 중간으로 갈수록 점점 감소한다.

특성 1로부터, 앞서 계산된 최단 경로상의 대부분의 노드들이 간선 비용이 갱신된 후 새롭게 재 계산된 최단 경로에서 재 사용된다고 결론 내릴 수 있다. 특성 2와 특성 3은 각각 3.2절과 3.3절에서 논의한다.

3.2 탐색 공간의 크기와 경로 중복도 간의 상관 관계

이 절에서는 3.1절에서 언급한 특성 2의 원인을 분석한다. 먼저, 그래프의 크기가 모두 $|V|$=50×50, $|E|$=4× 50×49이고 Δ값이 각각 20%, 80%, 160%, 320%인 4 종류의 격자 그래프를 가정한다. 그리고 각 격자 그래프에 대해서 서로 다른 100개의 시드를 이용하여 총 4×100개의 격자 그래프를 생성한다. 그 다음 3.1절과 같은 5개의 최단 경로를 계산하여, 각 최단 경로의 5%, 10%, 15%, ..., 90%, 95%에 해당되는 위치의 간선 비용을 갱신하였다. 각 해당 위치의 간선 비용 갱신에 대해서 $DPS(P_s, P_n)$를 계산하였으며, 3.1절과는 달리 5개의 최단 경로 각각에 대해서 평균화한다. 그 결과를 그림 4에서 보인다.

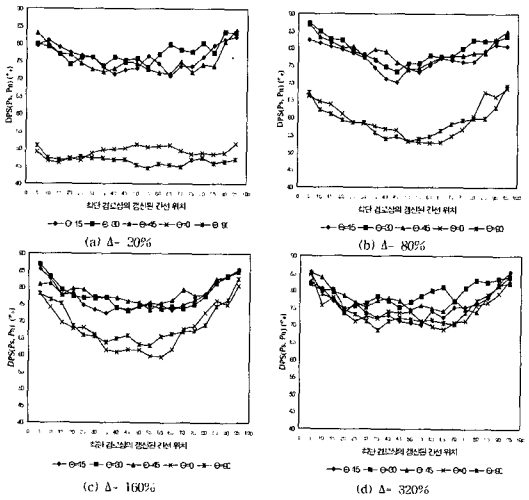


그림 4 Δ와 θ값에 따른 $DPS(P_s, P_n)$의 변화

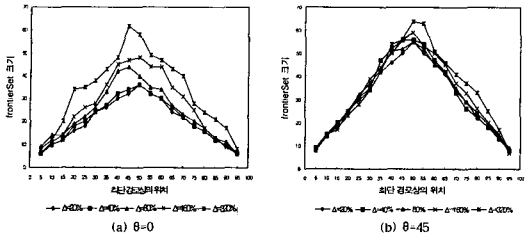
그림 4의 결과 그래프로부터 다음 두 가지 특성을 알 수 있다:

1. θ=0, 90[1]일 경우, $DPS(P_s, P_n)$는 Δ값에 따라 변한다.
2. θ=15, 30, 45일 경우, $DPS(P_s, P_n)$는 Δ값에 따라 변하지 않는다.

위의 두 가지 특성으로부터, $DPS(P_s, P_n)$는 격자 그래프의 Δ값과 최단 경로의 θ값이라는 두 파라미터에 의해 영향을 받는다고 결론을 내릴 수 있다. 본 논문에서는 이 사실의 원인을 보다 구체적으로 분석하기 위해 2장에서의 frontierSet의 변화를 실험해 보았다. 먼저, $|V|$=50×50, $|E|$=4×50×49이고 각각 20%, 40%, 80%, 160%, 320%의 Δ값을 가진 5개의 격자 그래프 $G(V,E,L)$를 생성하였다. 그 다음, $f(s,d)$=30이고 각각 θ=0, θ=45인 두 노드 쌍을 최단 경로로 선택하였다. 그리고, 각 격자 그래프 상에서 최단 경로를 계산하는 동안 frontierSet의 크기를 측정해 보았다. 그 결과를 그림 5에서 보인다.

그림 5로부터 θ와 Δ에 따라 *frontierSet*의 크기가 달라짐을 알 수 있다. *frontierSet* 크기는 그림 2에서의 타원[2] 바깥 둘레에 인접하고 있는 노드 개수를 나타낸

1) θ=0(또는 θ=90)일 경우, 출발지와 목적지는 같은 수평선(또는 수직선)상에 위치한다.



(a) θ=0 (b) θ=45

그림 5 Δ와 θ값에 따른 *frontierSet* 크기의 변화

수치이다. 따라서, *frontierSet* 크기는 최단 경로를 계산할 동안 검색된 탐색 공간의 크기를 나타낸다. 이제, Δ와 θ값에 의해서 탐색 공간의 크기가 결정된다는 사실을 정리 1을 통해 증명한다.

정리 1. $f(s,d)$가 일정할 때, 탐색 공간의 크기는 Δ와 θ에 의해 결정된다.

증명. Case 1 : Δ값이 일정할 때, θ값에 의해 탐색 공간의 크기가 결정됨을 보인다. 먼저, 그림 6을 본다. 그림 6의 격자 그래프 상에서 s에서 d까지의 모든 경로들 중 가장 적은 개수의 노드를 갖는 경로가 존재하는 영역은 s와 d를 두 꼭지점으로 갖는 직사각형(영역 S)이 된다. 때문에, $P_s(s,d)$를 찾기 위해서는 최소한 영역 S만큼은 검색해야 한다. 여기서, 영역 S의 크기에 따라 검색된 탐색 공간의 크기가 달라진다는 사실을 알아둔다.

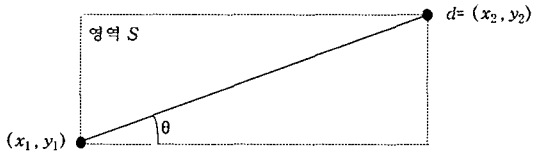


그림 6 격자 그래프 상의 출발지 s 와 목적지 d

이제, s와 d사이의 $f(s,d)$를 일정하게 하면서, θ값을 0에서 90까지 증가시킨다고 가정해 보자. 여기서, 90보다 큰 θ는 0에서 90사이의 θ로 나타낼 수 있기 때문에, 우리는 0에서 90사이의 θ값만을 고려한다. 결과적으로, 영역 S의 크기는 θ값이 0에서 45까지 증가할 동안 커지며, θ값이 45에서 90까지 증가할 동안에는 작아진다. 그 이유는 그림 7과 같이 반지름이 $r=f(s,d)$인 호 상에서 θ값을 0에서 90까지 증가시키며 영역 S의 크기 즉, $|x_2-x_1|\times|y_2-y_1|$의 값을 측정해 봄으로써 설명될 수 있다. 결과적으로, θ값에 의해서 영역 S의 크기가 결정되며, 따라서 검색된 탐색 공간의 크기는 θ값에 의해 결정된다.

Case 2 : 여기서 우리는 θ값이 일정할 때, Δ값에 따라 탐색 공간의 크기가 달라진다는 사실을 보인다. Δ값이 증가하면 격자 그래프 상의 모든 간선 비용간의 차이가 커지게 된다. 이 차이가 커질수록, 영역 S의 외부에 존재하는 노드들이 최단 경로에 포함될 가능성이 증가한다. 따라서, 최소한으로 검색해야 할 탐색 공간의 크기는 영역 S보다 커진다. 결과적으로, θ값이 일정할 때 Δ값에 의해 탐색 공간의 크기가 결정된다. □

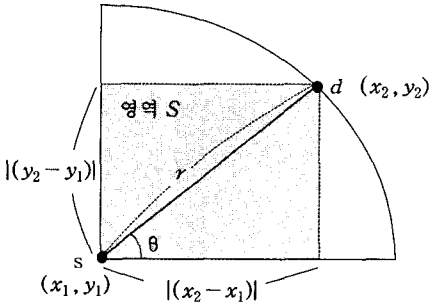


그림 7 θ에 S의 크기 변화

다음으로, 우리는 정리 2를 통해서 탐색 공간의 크기가 $DPS(P_s,P_n)$에 영향을 미친다는 사실을 보인다. 먼저, 정리 2에서 필요한 정의 3을 설명한 후, 정리 2를 증명한다.

정의 3. $P_s(s,d)$ 상의 어떤 간선 (p,q)의 비용이 갱신되었고, s~p까지의 노드 개수를 $|P_s(s,p)|$, q~d까지의 노드 개수를 $|P_s(q,d)|$라고 하자. 그리고, s~p까지의 노드들 중 x개의 노드들이, 그리고 q~d까지의 노드들 중 y개의 노드들이 $P_n(s,d)$에서 재 사용되었다고 하자. 이때, $P_s(s,d)$상의 갱신된 간선 (p,q)에 대해서 s방향으로의 장애 영향(Congestion Effect)을 $\frac{x}{|P_s(s,p)|}\times 100$, d방향으로의 장애 영향을 $\frac{y}{|P_s(q,d)|}\times 100$으로 나타낸다. □

정리 2. $DPS(P_s,P_n)$는 탐색 공간의 크기에 의해 결정된다.

증명. 그림 8과 같이 서로 다른 크기의 탐색 공간을 갖는 두 최단 경로를 가정해 보자. 여기서, 두 최단 경로에 대한 출발지와 목적지간의 $f(s,d)$는 서로 같다.

이때, 그림 8(a)에서 최단 경로 상의 c%(0<c<100)에

2) 타원은 검색된 탐색 공간을 나타낸다.

위치하는 간선 (p,q)의 비용이 갱신된 후, 새로운 최단 경로를 재 계산한 결과 노드 s에서 r사이에 있는 노드들이 재 사용되었다고 하자. 이 경우, 장애 구간이 s방향으로 미치는 장애 영향 α는 다음과 같다.

그림 8 탐색 공간의 크기에 따른 $DPS(P_s, P_n)$

$$\alpha(\%) = \frac{|P_s(r,p)|}{|P_s(s,p)|} \times 100.$$

여기서, $|P_s(r,p)|$와 $|P_s(s,p)|$는 각각 $P_s(r,p)$와 $P_s(s,p)$의 노드 개수이다. 결과적으로, 장애가 발생한 후 새로운 최단 경로를 얻기 위해서는 그림 8(a)의 빗금 친 영역만큼의 탐색 공간에 대해서 다시 재 계산해 볼 필요가 있다. 즉, 최단 경로가 될 수 있는 모든 가능한 대안 경로들(alternative shortest paths)이 빗금 친 탐색 공간으로부터 얻어 질 수 있다.

다음으로 그림 8(b)를 보자. 여기서도 그림 8(a)와 같이 최단 경로 상의 c%에 위치하는 간선 (p,q)의 비용이 갱신되었다고 하자. 이때, s방향으로 미치는 장애 영향 β는 α보다 작다. 그 이유는 그림 8(b)의 탐색 공간이 그림 8(a)의 탐색 공간보다 크기 때문이다. 즉, 그림 8(b)의 빗금 친 영역은 그림 8(a)의 빗금 친 영역만큼의 모든 가능한 대안 경로들을 포함한다. 이와 같은 현상은 d방향으로의 장애 영향을 비교할 경우에도 똑같이 나타날 것이다. 따라서, 탐색 공간의 크기가 클수록 장애 영향은 작으며, 따라서 $DPS(P_s, P_n)$가 높아진다. □

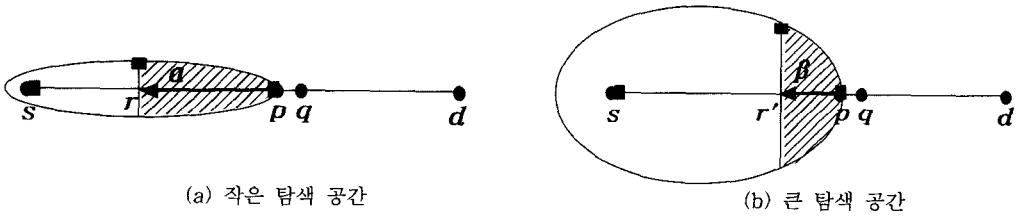
3.3 간선 비용이 갱신된 위치와 경로 중복도 간의 상관 관계

이 절에서는 3.1절의 특성 3을 설명한다. 먼저, 그림 9를 본다. 그림 9(a)는 $P_s(s,d)$의 50% 위치에 있는 간선이 갱신된 것이고, 그림 9(b)는 $P_s(s,d)$의 50% 위치보다 출발지 s에 더 근접해 있는 간선이 갱신된 것을 보여준다. 여기서, 갱신된 간선을 (p,q)로 나타내었다.

그림 9에서, Π_s^p은 s에서 p까지 검색된 탐색 공간을 나타내며, Π_d^q은 d에서 q까지 검색된 탐색 공간을 나타낸다. 또한, s방향으로의 장애 영향을 α%로, d방향으로의 장애 영향을 β%로 표시하였다. 이제, s에서 p까지의 노드 개수를 n, d에서 q까지의 노드 개수를 m이라고 하자. 그러면, 간선 (p,q)가 $P_s(s,d)$상에 미치는 장애 영향의 크기(eff 라고 하자.)는 다음과 같이 나타낼 수 있다.

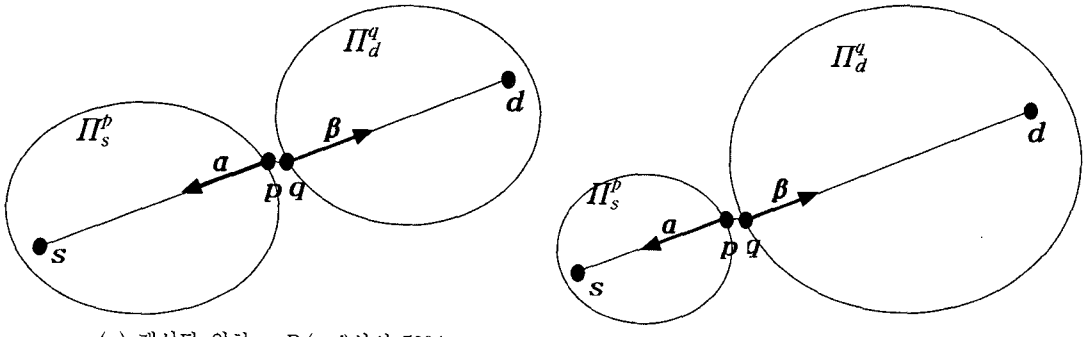
$$eff = \frac{\text{장애 영향을 받은 노드 개수}}{P_s(s,d) \text{ 상의 전체 노드 개수}} = \frac{n \cdot \alpha + m \cdot \beta}{n+m} \ (\%)$$

간선 (p,q)의 위치가 $P_s(s,d)$상의 중간에서 s로 근접할수록, m값은 n값에 비해 점점 커진다. 따라서, eff의 값은 β값에 점점 가까워진다. 또, (p,q)의 위치가 s로 근접할수록 Π_s^p은 작아지고 Π_d^q은 커지므로, 정리 2에 근거하여 장애 영향 α는 커지며, β는 작아진다. 결과적으



(a) 작은 탐색 공간 (b) 큰 탐색 공간

그림 8 탐색 공간의 크기에 따른 $DPS(P_s, P_n)$



(a) 갱신된 위치 = $P_s(s,d)$상의 50% (b) 갱신된 위치 < $P_s(s,d)$상의 50%

그림 9 갱신된 간선 위치에 따른 장애 영향의 크기

로, 간선 (p,q)의 위치가 $P_s(s,d)$상의 중간 위치에서 s로 근접할수록 장애 영향 eff의 값은 작아진다. 마찬가지로, 간선 (p,q)의 위치가 $P_s(s,d)$상의 중간에서 d로 근접할수록 장애 영향 eff의 값이 작아진다는 사실 또한 명백하다. 따라서, 특성 3이 증명되었다.

4. 동적 윈도우 기반의 근접 최단 경로 재 계산 방법

4.1 기본 개념

제안한 방법의 기본 개념은 간선 비용이 갱신된 후, 처음부터 다시 최적의 최단 경로를 재 계산하지 않고, 앞서 계산된 최단 경로 정보를 이용하여 근접 최단 경로를 산출하는 것이다. 제안한 방법을 설명하기 위해, 격자 그래프 $G(V,E,L)$ 상에서 그림 10과 같은 출발지 s에서 목적지 d까지의 최단 경로 $P_s(s,d)$가 있고, 이 최단 경로 상에서 간선 (p,q)의 비용이 갱신되었다고 가정해 보자[3].

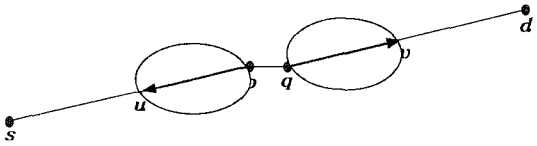


그림 10 출발지 s에서 목적지 d까지의 최단 경로 $P_s(s,d)$

간선(p,q)의 비용이 증가되었을 때, $DPS(P_s,P_n)$는 그림 3에서와 같이 평균적으로 60%에서 90%까지 이른다라는 사실을 알고 있다. 이 사실은 $P_s(s,d)$상의 어떤 노드 u와 v에 대해서, 간선 (p,q)의 비용 증가가 p에서 u까지 그리고 q에서 v까지의 구간에 영향을 미친다는 사실을 의미한다. 결과적으로, s에서 u까지 그리고 d에서 v까지의 두 경로가 s에서 d까지 재 계산된 최단 경로 $P_n(s,d)$에서 재 사용될 것이다. 여기서, 만약 노드 u와 v를 정확히 찾을 수 있다면, 다음 두 과정과 같이 $P_n(s,d)$을 산출할 수 있다:

1. 노드 u에서 v까지의 최단 경로 $P_s(u,v)$만을 A* 또는 OTO 알고리즘을 이용하여 계산한다.
2. 새로운 최단 경로 $P_n(s,d)$는 세 경로 $P_s(s,u)$, $P_s(u,v)$, $P_s(v,d)$들을 연결한 것이다. 즉, $P_n(s,d) = P_s(s,u) \| P_s(u,v) \| P_s(v,d)$.

3) 이 그림에서 노드 s에서 d까지의 최단 경로상의 모든 중간 노드들을 표시하지 않았다.

위의 방법으로 $P_s(s,d)$ 상에 하나의 장애가 발생했을 경우, $P_n(s,d)$을 처음부터 다시 재 계산할 필요 없이, 노드 u와 노드 v사이(장애로부터 영향을 받은 $P_s(s,d)$의 일부분)에 대해서만 최단 경로를 구하면 된다. 이 방법으로부터 최단 경로를 재 계산하는데 있어 부담해야 하는 많은 계산 시간을 줄일 수 있다.

그러나, 실제 최단 경로 $P_n(s,d)$에 근접하는 경로를 산출하기 위해서 노드 u와 v를 정확히 찾기란 매우 어렵다. 이에 본 논문에서는 적절한 노드 u와 v를 결정하여 $P_n(s,d)$에 근접한 경로를 구하는 동적 윈도우 기반의 근접 최단 경로 재 계산 방법을 제안한다. 여기서, 간선 비용의 갱신이 최단 경로에 미치는 영향 즉, 그림 10의 p에서 u까지 그리고 q에서 v까지의 구간을 "윈도우"라고 한다. 또, 여러 가지 파라미터를 고려하여(예를 들어, θ값과 Δ값 등) 윈도우 크기를 동적으로 결정한다라는 점에서 "동적"이라는 용어를 사용한다. 다음 4.2절에서는 주어진 상황에 따라 동적으로 윈도우 크기를 결정하는 방법에 대해서 설명한다.

4.2 동적 윈도우 결정 방법

제안한 방법의 목적은 $P_s(s,d)$상에 장애가 발생했을 경우, 새로운 최단 경로 $P_n(s,d)$에 근접하는 대략적인 최단 경로를 효율적으로 산출하는 것이다. 이를 위한 가장 중요한 작업은 적당한 윈도우 크기를 동적으로 결정하는 일이다. 4.1절에서 설명했듯이 윈도우 크기란 장애 영향의 크기를 의미하며, 이는 $DPS(P_s,P_n)$와 밀접한 관련이 있다. 따라서, 우리는 $DPS(P_s,P_n)$에 영향을 주는 파라미터들에 기반하여 적절한 윈도우 크기를 결정한다.

그림 10에서 알 수 있듯이, 최단 경로 상에 하나의 장애가 발생했을 경우 두 개의 윈도우가 발생하게 된다. 본 논문에서는 다음과 같은 두 가지 윈도우를 따로따로 계산한다:

- W_s : 출발지 s방향으로의 윈도우. 예를 들어, 그림 10에서 $W_s=[p,u]$.
- W_d : 목적지 d방향으로의 윈도우. 예를 들어, 그림 10에서 $W_d=[q,v]$.

먼저, 제안한 방법을 설명하는데 필요한 정리 3과 정리 4를 증명한다.

정리 3. $f(s,d)$가 일정할 때, 검색된 탐색 공간이 가장 작은 경우는 Δ=0%이고 θ=0(또는 θ=90)일 때이다.

증명. 정리 1로부터 정리 3은 명백하다. 또, Δ=0%이고 θ=0(또는 θ=90)일 때 계산된 최단 경로는 s와 d를

일직선으로 잇는 수평(또는 수직) 경로가 된다. □

정리 4. $f(s,d)$가 일정할 때, 검색된 탐색 공간이 가장 작을 경우의 $DPS(P_s,P_n)$는 평균적으로 50%이다.

증명. 정리 3으로부터, 탐색 공간이 가장 작을 때 $P_s(s,d)$는 수평(수직)인 경로이다. 이때, s에서 n개의 노드만큼 떨어진 간선 (p,q)의 비용이 갱신되었다고 하자(그림 11). 그러면, 탐색 공간의 크기가 최소이므로 frontierSet은 $P_s(s,d)$상의 각 노드에 인접하여 분포할 것이다. 또, frontierSet의 각 노드와 $P_s(s,d)$의 노드들을 연결하는 n개의 간선 $e_1,e_2,\ldots,e_n$을 표현하였다.

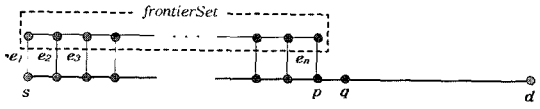


그림 11 탐색 공간이 가장 작은 경우 *frontierSet*의 분포

여기서, $P_n(s,d)$는 크게 두 가지 경우 즉, $P_s(s,d)$의 위 부분과 아래 부분을 지나는 경우 중 하나일 것이다. 먼저, $P_n(s,d)$이 $P_s(s,d)$의 위 부분을 지난다고 하자. 그러면, $P_n(s,d)$이 경유할 수 있는 간선의 범위는 $e_1,e_2,\ldots,e_n$이며, $P_n(s,d)$은 $e_1,e_2,\ldots,e_n$중 반드시 하나만을 경유한다. 그 이유는 다음과 같다.

$z_1=s$, $z_n=p$일 때, $(z_1,z_2,\ldots,z_n)$이 s에서 p까지의 최단 경로라고 하자. s에서 각 $z_i(2\le i\le n)$까지의 모든 최단 경로 $P_s(s,z_2)$, $P_s(s,z_3)$, $\ldots$, $P_s(s,z_n)$는 s와 z_i를 일직선으로 잇는 수평 경로이다. 만약, $P_n(s,d)$이 노드 z_a와 $z_b(1\le a<b\le n)$에 인접한 두 간선, e_a와 e_b를 경유한다고 하자. 이때, $P_n(s,d)=(s,..,z_a,..,z_b,..,d)$이다. 그러나, $P_s(s,z_b)$가 s에서 z_b까지 수평 경로이기 때문에, $P_s(s,z_b)$는 간선 e_a를 경유하지 않는다. 따라서, 가정은 모순이 되고, $P_n(s,d)$은 두 개 이상의 간선을 경유할 수 없다.

만약 $P_n(s,d)$이 e_1을 경유한다면, s에서 p사이의 모든 노드들이 $P_n(s,d)$에서 재 사용되지 않는다. 반면, $P_n(s,d)$이 e_n을 경유한다면 s에서 p사이의 모든 노드들이 $P_n(s,d)$에서 재 사용된다. 각 간선 $e_1,e_2,\ldots,e_n$에 대해서, $P_n(s,d)$가 경유할 가능성은 모두 같다. 따라서, s에서 p사이의 노드들 중 평균적으로 $n/2$개가 $P_n(s,d)$에서 재 사용된다고 볼 수 있다. 마찬가지로, q에서 d사이의 노드 개수가 m개라고 하면, q에서 d사이의 노드들 중 $m/2$개가 $P_n(s,d)$에서 재 사용된다. 결과적으로, $P_s(s,d)$의 전체 노드 개수가 $n+m$라고 할 때, 평균적으로 $(n+m)/2$개의 노드가 $P_n(s,d)$에서 재 사용된다. 그러므로, 정의 2에 의해 $DPS(P_s,P_n)=100\times(n+m)/2(n+m)=50\%$이다. 또, 이와 유사하게 $P_n(s,d)$이 $P_s(s,d)$의 아래 부분을 지나갈 경우, $DPS(P_s,P_n)$의 평균값이 50%이라는 사실을 유도할 수 있다. 따라서, 정리 4가 증명되었다. □

이제, 우리는 정리 4를 이용하여 적당한 윈도우 크기를 동적으로 결정하는 방법을 설명한다. 먼저, 출발지가 s이고 목적지가 d인 최단 경로 $P_s(s,d)$상의 어떤 간선 예를 들어, s에서 n개 노드만큼 떨어진 간선 (p,q)에 장애가 발생했다고 가정하자. 이때, s에서 장애 노드 p까지의 탐색 공간이 각각 최소일 경우와 최소보다 큰 경우를 그림 12에서 표현하였다. 여기서 점선으로 된 타원(π_s^p라고 하자.)이 최소 탐색 공간을, 실선으로 된 타원(Π_s^p라고 하자.)이 최소보다 큰 탐색 공간을 나타낸다. 그리고, 그림 8과 같이 s에서 p까지 생성된 탐색 공간이 π_s^p일 경우와 Π_s^p일 경우에 대한 장애 영향을 각각 α와 β로 표현하였다. 여기서, 목적지 d 방향으로의 장애 영향은 나중에 고려하기로 한다.

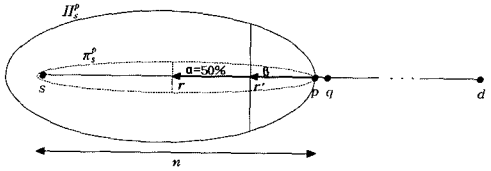


그림 12 탐색 공간의 크기에 따른 장애 영향의 변화

우리는 정리 4로부터 탐색 공간의 크기가 최소인 경우, 평균적으로 α=50%라는 것을 알 수 있다. 이때, 장애 노드 p까지 생성된 탐색 공간이 최소인 경우에 대해서 $W_s=[p,r]$이 된다. 여기서, 노드 r은 노드 p에서부터 $n\times\alpha/100=n/2$개의 노드만큼 떨어진 노드이다.

이제, 탐색 공간의 크기가 최소보다 큰 경우의 장애 영향 β를 구해야한다. 정리 2로부터, 탐색 공간의 크기가 커질수록 장애 영향은 작아진다는 것을 알 수 있으며, 따라서 β<α라는 사실은 명백하다. 그러면, β값이 α값보다 얼마나 작은지를 결정해야 한다. 우리는 정리 2에 근거하여 β의 값을 식 (1)과 같이 Π_s^p와 π_s^p의 크기 비율에 의해 결정한다.

$$\beta = \alpha \times \frac{Size\ of\ \pi_s^p}{Size\ of\ \Pi_s^p}\ (\%) \quad (1)$$

식 (1)은 π_s^p을 기준으로 하여, Π_s^p의 크기가 π_s^p의 크기보다 큰 정도만큼 α에서 줄여 β를 구한다는 의미이다. 탐색 공간의 크기는 frontierSet의 크기와 비례하므로, 식 (1)은 식 (2)로 다시 나타낼 수 있다.

$$\beta = \alpha \times \frac{|frontierSet_p^{\pi}|}{|frontierSet_p^{\Pi}|}\ (\%) \quad (2)$$

여기서, $|frontierSet_p^{\pi}|$는 탐색 공간이 π_s^p일 경우 s에서 p까지 생성된 frontierSet의 크기이고, $|frontierSet_p^{\Pi}|$는 탐색 공간이 Π_s^p일 경우 s에서 p까지 생성된 *frontierSet*의 크기이다. 탐색 공간이 π_s^p일 경우 *frontierSet*은 그림 11과 같이 최단 경로상의 각 노드들에 인접하여 분포할 것이다. 그러므로, 식 (2)는 식 (3)과 같이 간단하게 표현될 수 있다.

$$\beta = \alpha \times \frac{n \times 2 + 2}{|frontierSet_p^{\Pi}|}\ (\%) \quad (3)$$

식 (3)에서 α=50%이며, n과 $|frontierSet_p^{\Pi}|$의 값을 알기 때문에 β값을 구할 수 있다. 이제, β값으로부터 p에서부터 $n \times \beta/100$개만큼 뒤에 있는 노드 r'을 구할 수 있고, 결과적으로, 장애 노드 p까지 생성된 탐색 공간이 최소보다 큰 경우에 대해서 $W_s = [p, r']$이 된다.

이제, 위 방법과 유사하게 W_d도 구할 수 있다. 즉, q에서 d까지의 노드 개수가 m이라고 하면, 식 (4)와 같이 d 방향으로의 장애 영향을 구할 수 있으며, 결과적으로 W_d를 결정할 수 있다.

$$\beta = \alpha \times \frac{m \times 2 + 2}{|frontierSet_q^{\Pi}|}\ (\%) \quad (4)$$

그러나, 지금까지의 방법으로는 최단 경로 상의 모든 위치(간선 (p,q)의 위치)에서 W_s와 W_d를 구할 수가 없다. 왜냐하면, 식 (3)과 식 (4)의 $|frontierSet_p^{\Pi}|$와 $|frontierSet_q^{\Pi}|$가 각각 출발지 s와 목적지 d에서부터 확장된 탐색공간의 크기이기 때문이다. 실제로 OTO 알고리즘에서 탐색 공간은 그림 2의 (b)와 같이 확장되므로, 최단 경로 상의 각 노드들이 유지하고 있는 *frontierSet* 크기는 출발지에서 확장된 것과 목적지에서 확장된 것으로 나눠질 것이다. 그림 13은 이 사실을 도식적으로 표현한 것이다. 여기서, *frontierSet* 크기는 대략적으로 나타내었으며, 최단 경로 계산 알고리즘이 종료되는 중간 지점을 점선으로 표시하였다.

우리는 알고리즘이 종료되는 시점을 최단 경로 상의 중간 지점이라고 가정한다. 여기서, 최단 경로 상의 두

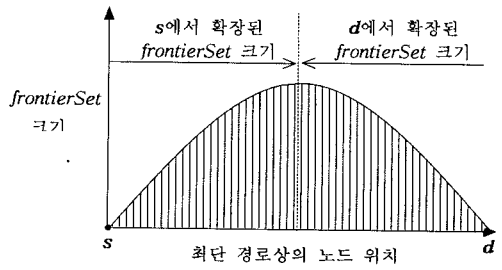


그림 13 s 또는 d에서 확장된 *frontierSet*

부분 [0%,50%]와 [50%,100%]을 각각 구간 1과 구간 2라고 하자. 이때, 구간 1은 s에서 확장된 *frontierSet*의 크기를, 구간 2는 d에서 확장된 *frontierSet*의 크기를 유지한다. 그러므로, 장애 구간 (p,q)가 구간 1에 존재할 경우의 W_d와 구간 2에 존재할 경우의 W_s를 결정하는 방법이 필요하다. 이 방법을 설명하기 위해 그림 14를 살펴본다.

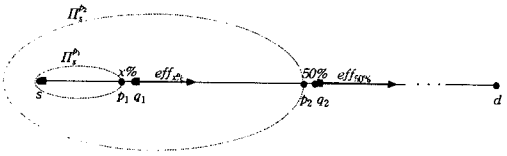


그림 14 장애 영향의 계산

그림 14에서 두 간선 (p_1, q_1)와 (p_2, q_2)는 각각 $P_s(s,d)$상의 50%와 $x\%(0 \le x \le 50)$위치에 있으며, $\Pi_s^{p_1}$와 $\Pi_s^{p_2}$는 s에서부터 각각 p_1과 p_2까지 확장된 탐색 공간을 나타낸다. 이때, 간선 (p_2, q_2)에 장애가 발생했을 경우 d 방향으로의 장애 영향을 $eff_{50\%}$, 간선 $(p_1,)$에 장애가 발생했을 경우의 장애 영향을 $eff_{x\%}$라고 하자. 우리는 식 (4)에 의해서 $eff_{50\%}$을 쉽게 구할 수 있다. 왜냐하면, 노드 q_2에서 측정된 $|frontierSet_{q_2}^{\Pi}|$는 목적지 d에서부터 확장된 것이기 때문이다. 그러나, 노드 q_1에서 측정된 $|frontierSet_{q_1}^{\Pi}|$는 s에서부터 확장된 크기이기 때문에, 식 (4)에 의해 $eff_{x\%}$을 구할 수 없다. 즉, W_d를 구할 수 없다.

이제, $eff_{x\%}$를 구하는 방법을 설명한다. 먼저, d에서부터 q_1까지 확장된 frontierSet 크기가 $|frontierSet_{q_2}^{\Pi}|$보다 크기 때문에, 정리 2에 의해서 $eff_{x\%} < eff_{50\%}$라는 사실을 알 수 있다. 그러면, $eff_{x\%}$의 크기가 $eff_{50\%}$의 크기보다 얼마나 작은지를 구해야 한다. 이를 위해서, 우리는 $\Pi_s^{p_1}$이 $\Pi_s^{p_2}$보다 작은 정도만큼 $eff_{50\%}$값에서 줄여

$eff_{x\%}$을 구한다. 여기서, $\Pi_s^{p_1}$와 $\Pi_s^{p_2}$의 크기 비율은 $|frontierSet_{p_1}^{\Pi}|$와 $|frontierSet_{p_2}^{\Pi}|$의 비율로 나타낼 수 있으므로, 다음 식 (5)와 같이 $eff_{x\%}$를 구할 수 있다.

$$eff_{x\%} = eff_{50\%} \times \frac{|frontierSet_{p_1}^{\Pi}|}{|frontierSet_{p_2}^{\Pi}|} \quad (5)$$

$|frontierSet_{p_2}^{\Pi}|$에 비해 $|frontierSet_{p_1}^{\Pi}|$가 작을수록, d에서 q_1까지 확장된 *frontierSet*의 크기가 $|frontierSet_{q_2}^{\Pi}|$비해 커지기 때문에, 정리 2에 의해 식 (5)를 유도할 수 있다. 이제, 식 (5)로부터 구간 1에 장애가 발생했을 경우 W_d를 구할 수 있다. 또, 식 (5)와 같은 방법으로 구간 2에 장애가 발생할 경우의 W_s도 결정할 수 있다.

지금까지 윈도우 결정 방법에 대해 설명하였다. 우리는 이 방법에서 *frontierSet* 크기간의 비율을 이용하였다. 이는 최단 경로 상의 구간 1과 구간 2 각각에서 *frontierSet*의 크기가 비례적으로 증가하고 감소한다라는 사실에 근거를 두고 있다. 본 논문에서 최단 경로를 찾기 위해 OTO 알고리즘을 이용했기 때문에, *frontierSet*의 크기는 그림 5와 같이 비례적으로 증가하고 감소한다. 따라서, 윈도우 크기를 결정하는 식에서 *frontierSet* 크기에 대한 비례식의 사용은 타당하다고 할 수 있다.

4.3 효율성 검증 I

이 절에서는 본 논문에서 제안한 방법의 효율성을 증명한다. 이를 위해 먼저, 제안한 방법에 의해 산출된 근접 최단 경로를 $P_{app}(s,d)$라고 하자. 이때, 처음부터 재계산된 최적의 최단 경로 $P_n(s,d)$와 $P_{app}(s,d)$를 비교함으로써 제안한 방법의 효율성을 보이기로 한다. 이 비교는 다음 세 가지 측면에서 수행된다.

- 노드 차이 : $P_{app}(s,d)$와 $P_n(s,d)$사이의 경로 중복도 $DSP(P_{app}, P_n)$를 평가한다.
- 비용 차이 : $P_{app}(s,d)$와 $P_n(s,d)$ 사이의 경로 비용(경로 길이)을 비교한다.

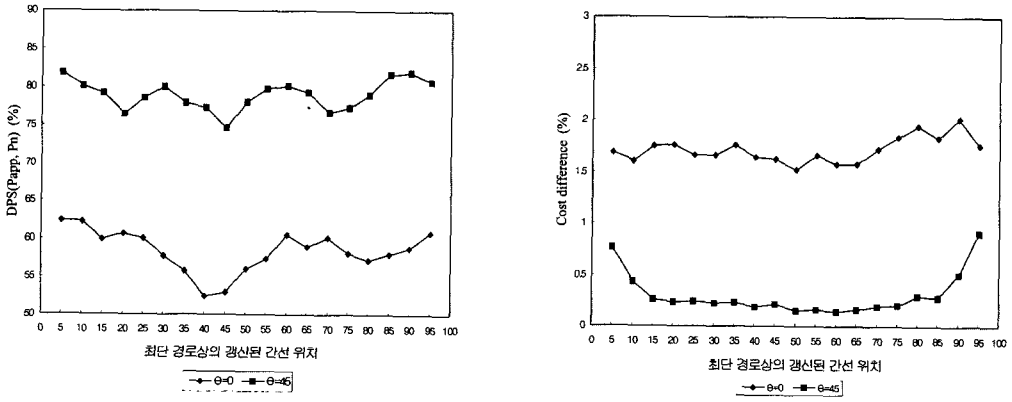
  즉, $\frac{P_{app}\text{의 } \cos t - P_n\text{의 } \cos t}{P_n\text{의 } \cos t} \times 100$을 측정한다.
- 계산 시간 차이 : $P_{app}(s,d)$와 $P_n(s,d)$사이의 계산 시간을 비교한다.

위의 실험은 표 1과 같은 환경에서 이루어진다.

표 1 실험 환경

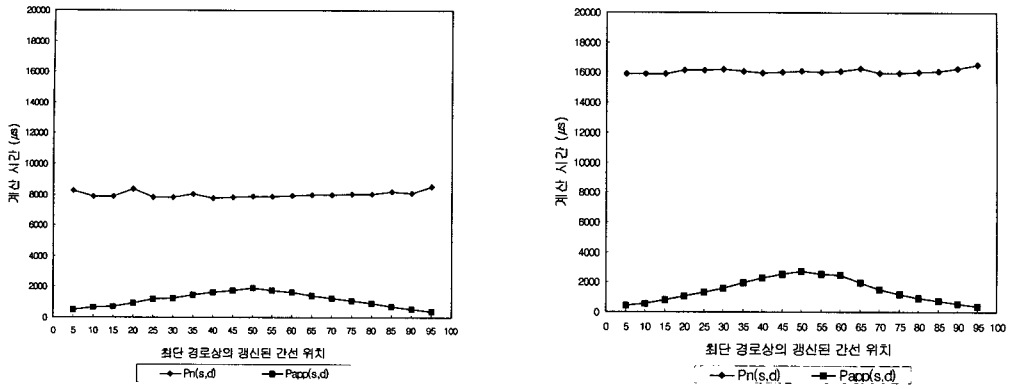
Machine	Processor	Memory
IBM Netfinity 4500R	Pentium III 1G zeon (Dual)	1G

먼저, 그래프의 크기가 $|V|$=50×50, $|E|$=4×50×49이고 Δ=80%인 격자 그래프 $G(V,E,L)$를 가정해 본다. 그리고, 이 격자 그래프에 대해서 서로 다른 100개의 시드값을 이용하여, 총 100개의 격자 그래프를 생성한다. 그 다음, 최단 경로 계산을 위해서 θ=0과 θ=45인 두 개의 노드 쌍을 선택한다. 여기서, $f(s,d)$는 30으로 모두 같다. 각 격자 그래프에 대해서, 두 개의 최단 경로를 계산하고, 각 최단 경로의 5%, 10%, 15%, ... , 90%, 95% 위치에 해당되는 간선 비용을 한번에 하나씩만 갱신해 가며 근접 최단 경로를 계산한다. 각 간선의 갱신에 대해서 노드 차이, 비용 차이, 계산 시간 차이를 측정하였다. 여기서 측정된 모든 결과값은 100개의 격자 그래프에 대해 평균화된다. 그림 15와 16은 이 실험에 대한



(a) 노드 차이 (b) 비용 차이

그림 15 $P_{app}(s,d)$와 $P_n(s,d)$사이의 노드와 비용 차이



(a) θ=0 (b) θ=45

그림 16 $P_{app}(s,d)$와 $P_n(s,d)$사이의 계산 시간 차이

결과를 나타낸 것이며, 이 결과는 제안한 방법의 효율성을 명백히 증명한다.

4.3 효율성 검증 II

이 절에서는 실제 지도상에서 제안한 방법을 적용하여 그 효율성을 실험해 본다. 이 실험은 노드수가 2956 에이지수가 4518개인 San Francisco Bay bridge를 모델링한 지도(그림 17)상에서 수행되며, Arcview GIS 3.2a라는 GIS Tool을 사용하여 성능 분석을 하였다.

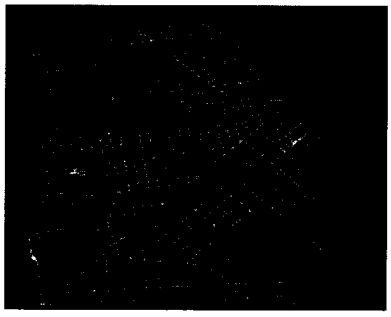
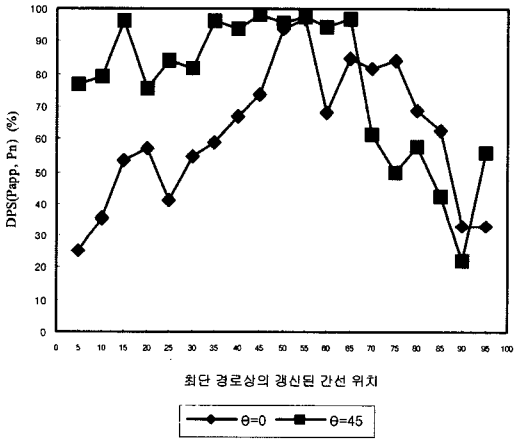
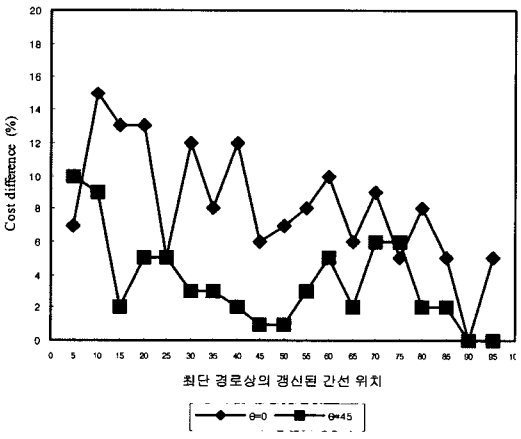


그림 17 San Francisco Bay bridge

먼저, 그림 17의 지도상에 최단 경로 계산을 위해서 θ=0과 θ=45인 두개의 노드 쌍을 선택한다. 여기서, $f(s,d)$는 약 40km으로 모두 같다. 그 다음, 두 개의 최단 경로를 계산하고 각 최단 경로의 5%, 10%, 15%, ... , 90%, 95% 위치에 해당되는 간선 비용을 한번에 하나씩만 갱신해가며 근접 최단 경로를 계산한다. 각 간선의 갱신에 대해서 4.3절과 같은 노드 차이, 비용 차이, 계산 시간 차이를 측정하였다. 그림 18과 19는 이 실험에 대한 결과를 나타낸 것이며, 4.3절과 비슷한 효율성을 보인다.

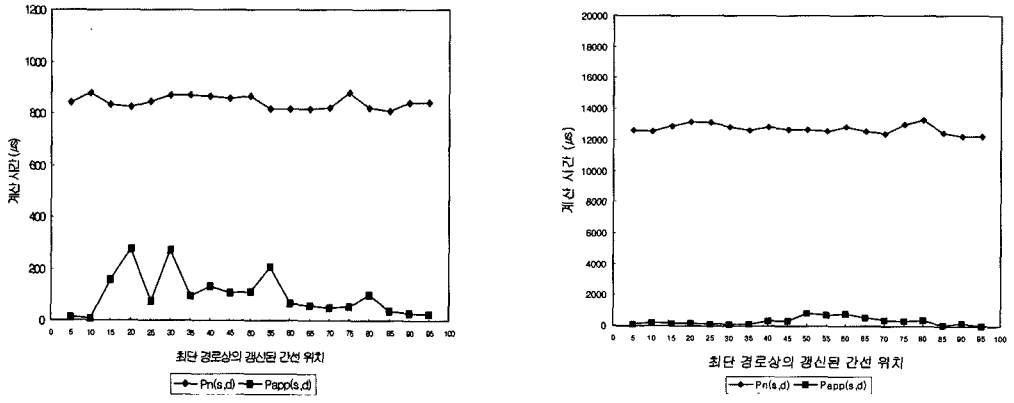


(a) 노드 차이



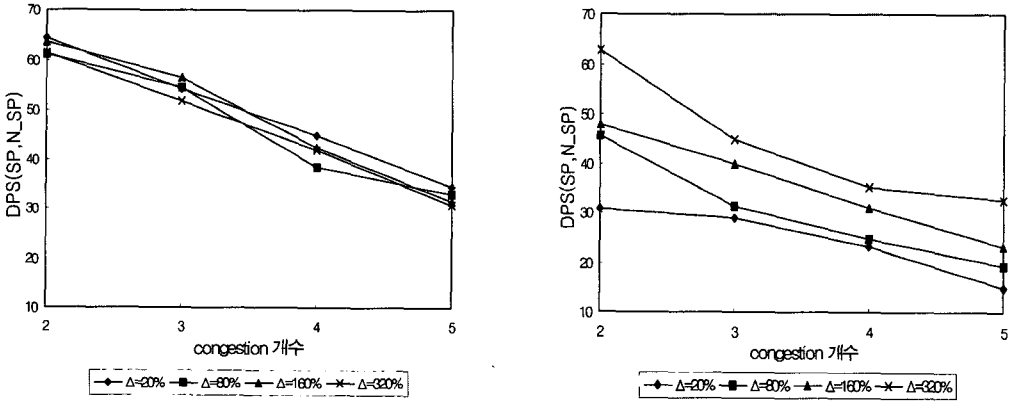
(b) 비용 차이

그림 18 $P_{app}(s,d)$와 $P_n(s,d)$사이의 노드와 비용 차이



(a) θ=45 (b) θ=0

그림 19 $P_{app}(s,d)$와 $P_n(s,d)$사이의 계산 시간 차이



(a) θ=45 (b) θ=0

그림 20 여러 간선 비용의 갱신에 대한 $DPS(P_s, P_n)$

5. 여러 간선 비용의 갱신에 대한 근접 최단 경로 재 계산 방법

실제 교통상황에서는 최단 경로 상에 두 개 이상의 간선 비용이 갱신될 수 있다. 따라서, 이 장에서는 $P_s(s,d)$ 상에 두 개 이상의 간선 비용이 갱신되었을 경우 근접 최단 경로를 산출하는 방법에 대해 설명한다. 먼저, 여러 간선 비용이 갱신된 후 $DPS(P_s, P_n)$를 측정해 볼 필요가 있다. 이 실험을 위해서 |V|=50×50, |E|=4×50×49이고 Δ값이 각각 20%, 80%, 160%, 320%인 4개의 격자 그래프를 가정해본다. 그리고, 서로 다른 시드값을 이용하여 4×100개의 격자 그래프를 생성하였다. 다음, 각 격자 그래프 상에 θ=0, 45인 2개의 최단 경로를 선택한다. 여기서, $f(s,d)$=40이다. 이제, 모든 최단 경로를 계산한 후, 각 최단 경로의 5%, 10%, ..., 90%, 95% 위치에 있는 간선들 중에서 임의로 2개의 간선 비용을 갱신하여 $DPS(P_s, P_n)$를 측정하였다. 또, 이와 같은 방법으로 3개, 4개, 5개의 간선 갱신에 대해서도 $DPS(P_s, P_n)$를 측정하였다. 그 결과를 100개씩 평균화하여 그림 20에 나타내었다.

위 그림으로부터, 2~3개의 간선 갱신에 대한 $DPS(P_s, P_n)$는 하나의 간선 갱신에 대한 $DPS(P_s, P_n)$와 큰 차이가 없다는 사실을 알 수 있다. 또, 이 사실은 2~3개의 간선 비용이 갱신되었을 경우에도 근접 최단 경로 재 계산 방법을 적용할 수 있는 가능성을 의미한다.

이제, 여러 간선 비용의 갱신에 대한 방법을 설명한다. 먼저, 최단 경로 $P_s(s,d)$상에 n개의 간선 c_1, c_2, ..., c_n의 비용이 갱신된 상황을 나타낸 그림 21을 살펴본다.

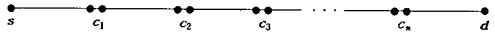


그림 21 최단 경로 $P_s(s,d)$사에서 갱신 n개의 간선

우리는 n개의 장애 구간에 대해서 n개의 윈도우 W_i ($1 \le i \le n$)를 구해야 한다. 여기서, W_i는 W_s와 W_d를 묶어 표현한 것이다. 이제, $P_s(s,d)$상에 c_1만이 갱신되었다고 가정하여 새롭게 재 계산한 최단 경로를 $P_{c1}(s,d)$이라고 하자. 또, $P_{c1}(s,d)$상에 c_2만이 갱신된 후, 새롭게 재 계산된 최단 경로를 $P_{c2}(s,d)$라고 하자. 결국 한 번에 하나의 간선만을 갱신하여 n번 재 계산한 최단 경로 $P_{cn}(s,d)$을 얻을 수 있다. 만약, n개의 간선이 한꺼번에 갱신된 후, 처음부터 다시 재 계산된 새로운 최단 경로를 $P_n(s,d)$라고 하면, 이때 $P_n(s,d) = P_{cn}(s,d)$이다. 그러므로, 동시에 n개의 간선 비용이 갱신되었다 하더라도, 한번에 하나의 간선 갱신만을 고려하는 순차적인 방법으로 새로운 최단 경로를 구할 수 있다. 이 사실은 갱신된 각 간선에 대한 장애 영향을 다른 것과는 독립적으로 계산할 수 있다는 것을 의미한다. 따라서, 우리는 n개의 간선(c_1, c_2, c_3, ..., c_n) 각각에 대한 갱신 영향을 4장에서 설명한 식 (3), (4), (5)를 이용하여 계산한다. 결과적으로, n개의 윈도우 모두를 결정할 수 있다.

그러면, 위의 방법이 2~3개의 간선 비용 갱신에 대하여 얼마나 효율적인지를 실험해 본다. 먼저, 크기가 |V|=50×50, |E|=4×50×49이고 Δ=80%인 격자 그래프를 가정해 본다. 그리고, 서로 다른 100개의 시드값을 이용하여, 100개의 격자 그래프를 생성한다. 그 다음, θ=0과 θ=45인 두 개의 노드 쌍을 최단 경로로 선택한다. 여기서, $f(s,d)$=30이다. 이제, 각 최단 경로의 5%, 10%, ..., 90%, 95% 위치 있는 간선들 중에서 임의로 2개의 간선 비용을 갱신하여 4장에서와 같은 세 가지 비교를 수행해 보았다. 또, 같은 방법으로 3개의 간선 갱신에 대해서도 실험을 하였다. 그 결과를 100개씩 평균화하여 표 2와 3에 나타내었다.

표 2 $P_{app}(s,d)$와 $P_n(s,d)$사이의 노드와 비용 차이 (%)

갱신된 간선의 개수	θ=45	θ=0
2	72.210854	66.285712
3	65.059429	51.028433

(a) 노드 차이

갱신된 간선의 개수	θ=45	θ=0
2	0.545533	1.902547
3	1.127255	3.388068

(b) 비용 차이

표 3 $P_{app}(s,d)$와 $P_n(s,d)$사이의 계산 시간 차이 (μs)

갱신된 간선의 개수	$P_n(s,d)$	$P_{app}(s,d)$
2	18446	2829
3	13938	4810

(a) θ=45

갱신된 간선의 개수	$P_n(s,d)$	$P_{app}(s,d)$
2	7182	2063
3	7559	4512

(b) θ=0

6. 결 론

본 논문에서는 앞서 계산된 최단 경로 정보를 이용하는 효율적인 동적 윈도우 기반의 근접 최단 경로 재 계산 방법을 제안하였다. 이 방법은 어떤 최단 경로 상에 간선 비용이 갱신되었을 경우, 처음부터 다시 최적의 최단 경로를 재 계산하지 않고, 오직 간선 비용의 변화에 영향을 받은 부분만을 재 계산하여 최적의 경로에 매우 가까운 근접 최단 경로를 빠르게 재 계산해 낸다.

제안한 방법에서 가장 중요한 작업은 간선 비용의 갱신으로부터 영향을 받은 부분 즉, 윈도우 크기를 적절하게 결정하는 일이다. 이를 위해 본 논문에서는 최단 경로 상의 갱신 영향에 대한 이론적이고 실험적인 분석을 하였으며, 이 분석 내용에 기반을 두어 윈도우 크기를 동적으로 결정하는 효율적인 방법을 제안하였다. 또한, 제안한 방법을 실제로 구현하고 수많은 격자 그래프와 최단 경로 상에 이를 적용하여 면밀한 성능 분석을 하였다. 실험 결과로부터 제안한 방법이 최적의 경로에 근접하는 경로를 매우 빠른 시간 내에 산출한다는 사실을 증명하였다.

본 논문에서 제안한 방법은 최단 경로의 길이와 갱신된 간선의 개수에 관련하여 그 효율성이 좌우된다. 예를 들어, 길이가 짧은 최단 경로 상에 많은 간선 비용이 갱신되었을 경우, 제안한 방법을 이용하는 것보다 처음부터 다시 최단 경로를 재 계산하는 것이 더 효율적일 것이다. 결국, 갱신된 간선의 개수가 많을수록 최적의 최단 경로에 근접하는 경로를 산출하기가 어려워지며, 최단 경로의 길이가 짧을수록 제안한 방법으로부터 얻어지는 계산 시간 이득은 적어진다. 이와는 또 다른 문제로, 제안한 방법에 의해 산출된 근접 최단 경로 상에 간선 비용이 갱신되어 계속적으로 제안한 방법을 적용한다면, 산출되는 경로의 효율성은 점점 떨어질 것이다. 이에, 제안한 방법이 효율적으로 적용될 수 있는 범위에 대한 연구가 필요하다.

참 고 문 헌

[1] R. Agrawal and H. Jagadish, "Algorithms for Searching Massive Graphs", In IEEE Transactions on Knowledge and Data Engineering, Vol. 6, No.2, pp. 225-238, April 1994.

[2] R. Kung, E. Hanson, Y. Ioannnidis, T. Sellis, L. Shapiro, and M. Stonebraker, "Heuristic Search in Data Base System", In Proc. 1st Int. Workshop Expert Database Systems, pp. 96-107, Oct. 1984.

[3] Y. Huang, N. Jing, and E. Rundensteiner, "Hierarchical Path Views: A Model Based on Fragmentation and Transportation Road Types", In Proc. of the 3rd ACM Workshop on Geographic Information Systems, pp. 93-100, 1995.

[4] K. Ishikawa, M. Ogawa, S. Azume, and T. Ito, "Map Navigation Software of the Electro Multivision of the '91 Toyota Soarer", In Int. Conf. on Vehicle Navigation and Information Systems(VNIS IVHS), IEEE, (1991) pp. 463-473.

[5] B. Liu, S. Choo, S. Lok, S. Leong, S. Lee, F. Poon, and H. Tan, "Integrating Case-Based Reasoning, Knowledge-Based Approach and Dijkstra Algorithm for Route Finding", Proc. Tenth Conf. Artificial Intelligence for Applications (CAIA '94), pp. 149-155, 1994.

[6] J. Shapiro, J. Waxman, and D. Nir, "Level Graphs and Approximate Shortest Path Algorithms", In Networks, Vol. 22, pp. 691-717, 1992.

[7] T. Yang, S. Shekhar, B. Hamidzadeh, and P. Hancock, "Path Planning and Evaluation in IVHS Databases", IEEE Int'l Conf. on Vehicle Navigation and Information Systems(VNIS IVHS), pp. 283-290, 1991.

[8] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H. Gracia-Molina, "Proximity Search in Databases", In Proceedings of the 24th VLDB Conference, pp. 26-37, 1998.

[9] N. Jing, Y. Huang, and E. Rundensteiner, "Hierarchical Optimization of Optimal Path Finding for Transportation Applications", In Proc. of 5th Int'l Conf. on Information and Knowledge Management, pp. 261-268, 1996.

[10] N. Jing, Y. Huang, and E. Rundensteiner, "Hierarchical Encoded Path Views for Path Query Processing: An Optimal Model and Its Performance Evaluation", In IEEE Transactions on Knowledge and Data Engineering, Vol. 10, No. 3, pp. 409-432, May/June 1998.

[11] S. Jung and S. Pramanik, "HiTi Graph Model of Topographical Road Maps in Navigation Systems", Proceedings of the 12th IEEE International Conference on Data Engineering, pp. 76-84, New Orleans, Louisiana, Feb. 1996.

[12] T. Mohr and C. Pasche, "A Parallel Shortest Path Algorithm", In Computing, Vol. 40, pp. 281-292, 1988.

[13] S. Shekhar, A. Kohli, and M. Coyle, "Path Computation Algorithms for advanced Traveler Information System (ATIS)", In Proc. IEEE 9th Int'l Conf. Data engineering, pp. 31-39, 1993.

김 재 훈

2000년 2월 한국항공대학교 항공통신정보학과 학사. 2002년 8월 서강대학교 대학원 컴퓨터학과 석사. 2002년 9월(현재 : LG전자 연구원). 관심분야는 이동통신, 이동DB, GSM

정 성 원

1988년 서강대학교 전자계산학 학사. 1990년 M.S. in Computer Science at Michigan State Univ. 1995년 Ph.D. in Computer Science at Michigan State Univ. 1997년~2000년 한국전산원 선임연구원. 2000년~현재 서강대학교 컴퓨터학과 조교수. 관심분야는 Mobile Computing Systems, Mobile Databases, ITS/GIS, Spatial DB, Distributed Databases

박 성 용

1987년 서강대학교 전자계산학과 졸업. 1994년 Syracuse 대학 Computer Science 석사. 1998년 Syracuse 대학 Computer Science 박사. 1998년~1999년 Bellcore 연구소 연구원. 1999년~현재 서강대학교 컴퓨터학과 조교수. 관심분야는 High Performance Cluster Computing and System. High Performance Communication and Network I/O. Networked Storage Technologies (SAN and NAS). Meta-Computing (Grid Computing)