디지털 로드맵 데이타베이스에서 효율적인 동적 경로 질의어 처리 방안

(An Efficient Dynamic Path Query Processing Method for Digital Road Map Databases)

정 성 원 †

(Sungwon Jung)

요 약 차량 항법 시스템에서 가장 중요한 기능중의 하나는 현재 위치로부터 목적지까지의 가능한 경로들 최단경로를 계산하는 일이다. 차량 항법 시스템의 중요한 어려움 중 하나는 대용량 디지털 로드맵상에서 최단경로를 구할 때 많은 계산 시간이 소요된다는 점이다. 차량 항법 시스템은 실시간 시스템이므로, 제한 시간을 만족하여 최소 비용 경로를 계산하는 것은 매우 중요하다. 본 논문에서는 대용량의 디지털 로드맵(Road Maps)에서 최단 경로 계산을 효율적이고 빠르게 하기 위해서 디지털 로드맵을 효율적으로 계층 구조화하는 *HiTi*(Hierarchical MulTi) 그래프 모델을 개발하였다. 제안된 *HiTi* 그래프 모델에 기반을 둔 *SPAH*라는 새로운 최단 경로 알고리즘을 제안하고, *SPAH*가 계산한 최단경로의 최적성(Optimality)을 증명하였다. *SPAH*의 성능분석을 통하여 *SPAH*가 기존의 다른 최단 경로 계산 알고리즘들 보다 훨씬 빠르게 최단경로를 계산한다는 것을 보여주었다. *HiTi* 그래프 방법은 여러 가지 측면에서 다른 유사한 방법들과 상세하게 성능실험을 통하여 장·단점이 비교 분석되어 졌다.

Abstract In navigation system, a primary task is to compute the minimum cost route from the current location to the destination. One of major problems for navigation systems is that a significant amount of computation time is required when the digital road map is large. Since navigation systems are real time systems, it is critical that the path be computed while satisfying a time constraint. In this paper, we have developed a *HiTi*(Hierarchical MulTi) graph model for hierarchically structuring large digital road maps to speedup the minimum cost path computation. We propose a new shortest path algorithm named *SPAH*, which utilizes *HiTi* graph model of a digital road map for its computation. We prove that ths shortest path computed by *SPAH* is the optimal. Our performance analysis of *SPAH* also showed that it significantly reduces the computation time over exiting methods. We present an in-depth experimental analysis of *HiTi* graph method by comparing it with other similar works.

1. 서 론

항법 시스템에서 가장 중요한 기능중의 하나는 현재 위치로부터 목적지까지의 가능한 경로들 중 최소 기대 비용을 갖는 최단경로를 계산하는 일이다. 이 목적을 위해서, 항법 시스템에서는 다음과 같은 재귀적 관계(*Recursive Relation*)로 표현된 디지털 로드맵을 사용한다.

topographical_road_map(source, destination, cost)

여기서, cost 속성은 예를 들면 출발 지점에서부터 목적지점까지의 순회에 대한 최소 기대 시간을 나타낸다. 적용 가능한 또 다른 cost의 예로는 두 지점간의 최단 거리를 들 수 있다. 항법 시스템의 어려운 문제 중 하나는 디지털 로드맵 데이타의 크기이다. 100feet 간격으로 이산된 100*mile*×100*mile*의 작은 디지털 로드맵을 저장하기 위해서는 약 2.4Gbyte가 필요하다[1, 2]. 그러므로, 큰 디지털 로드맵을 가정해 볼 때 이에 필요한 데이타의 크기는 매우 커진다. 이러함 대용량의 디지털 로드

· 본 연구는 서강대학교 산업기술연구소의 지원을 받아 수행되었음.
† 정 회 원 : 서강대학교 컴퓨터학과 교수
jungsung@ccs.sogang.ac.kr
논문접수 : 2001년 1월 31일
심사완료 : 2001년 6월 26일

맵 데이타를 가지고 최소 비용 경로를 계산하려면 상당히 많은 계산 시간이 요구된다. 차량 항법 시스템은 실시간 시스템이므로, 제한 시간을 만족하여 최소 비용 경로를 계산하는 것은 매우 중요하다.

지금까지 데이타베이스 분야에서 최단 경로 계산 문제를 다룬 많은 연구 논문들이 보고되어왔다. 하지만 이전에 제안된 이행 클로저(transitive closure) 또는 그래프 탐색 알고리즘 [3, 4, 5, 6, 7, 8, 9, 10, 11]은 **topographical_road_map(source, destination, cost)**에 직접 적용할 경우 탐색할 데이타의 매우 큰 용량 때문에 상당히 많은 최단경로 계산 시간을 필요로 한다. 그러므로, 최소 비용 경로 계산을 빠르게 하기 위해서 디지털 로드맵을 구조화하는데 필요한 효율적인 데이타베이스 구성이 요구된다. 이 점에 있어서, 두 가지의 접근 방안이 지금까지 연구되고 있다. 한가지 접근 방안은 신속히 준 최적(Suboptimal) 최소 비용 경로를 제공하는 데이타베이스 구조를 개발하는 것이다. 다른 접근 방안은 우선적으로 미리 계산된 최단 경로 정보를 이용하여 최적의 최단 경로를 제공하는 데이타베이스 구조를 개발하는 것이다.

준 최적 최단 경로를 생성하기 위해서, Ishikawa et. al., Shapiro et. al., Liu et. al., Huang et. al.은 최소 비용 경로 계산을 빠르게 하기 위한 도로 계층(road hierarchies)(예: 고속 도로, 간선 도로, 측면 도로)을 사용했다[12, 13, 14, 15, 16]. 그들은 불필요한 탐색 공간을 줄이기 위해서, 디지털 로드맵에 대한 다중 수준(multiple level)의 계층적 세부사항(hierarchical details)을 사용했다. Huang et. al.[12]은 도로 형태에 따라 에지들을 분류하는 계층적인 그래프 모델을 제안했다. Ishikawa et. al.[13]은 계층화된 디지털 로드맵에 Dijkstra 알고리즘을 적용했다. Shapiro et. al.[15]은 이론적으로 도로 계층을 모델링하는 *LGS(Level Graph Structure)*라는 새로운 그래프 구조를 제안했다. LGS를 기반으로, Shapiro et. al.은 신속히 대략적인 최단 경로를 생성하는 새로운 알고리즘을 제공하였다. 그들의 연구는 LGS에 의해 생성된 경로의 길이는 출발지와 목적지 노드 사이의 거리가 증가함에 따라, 급속하게 실제 최적 최소 비용 경로의 길이로 접근한다. Liu et. al.[14]은 최소 비용 경로 계산을 위해서, 인공지능 지식기반 방법과 사례 기반 추론(case-based reasoning)을 Dijkstra 알고리즘과 통합하는 연구를 했다.

최적의 최단 경로를 생성하기 위해서, Agrawal과 Jagadish은 최근에 경로 정보를 부분적으로 미리 계산하고 저장하는 데이타 구성 기술을 연구했다[1]. 그들은 최단 경로를 계산할 때, 탐색 공간의 필요 없는 부분을 제거하기 위해서 미리 계산된 부분 경로 정보를 사용한다. 그들의 접근 방안은 최단 경로의 계산 속도를 증가시키지만, 디지털 로드맵 항법(road map navigation)에 기반을 둔 최단 경로 질의어 처리를 최적화하지는 못한다. 그 이유는 그들의 접근 방법이 최소 비용 경로에 위치한 모든 중간 노드들의 탐색을 여전히 요구하고 있기 때문이다. 항법 시스템에서 운전자는 최단 경로에 위치한 모든 중간 노드들을 알 필요는 없다.

위의 문제는 이산된 간격이 매우 작은 대용량의 디지털 로드맵에서 두 종점 노드들간의 거리가 멀리 떨어져 있는 최단 경로를 계산하는 경우에 있어서 더욱더 심각해 질 것이다. 이 문제를 해결하기 위해서 Jing et. al.은 *HEPV(Hierarchical Encoded Path View)*을 제안하였다[17, 18]. 그들의 기본 발상은 하나의 큰 그래프를 더 작은 서브그래프들로 나누고, 경계 노드들을 밀어올림으로써 계층적인 방법으로 그 서브그래프들을 조직화하는 것이다. 그러나, *HEPV*는 미리 계산된 경로 정보의 방대한 용량을 유지하기 위한 지나친 저장 오버헤드를 포함한다. 이것은 *HEPV*가 각 서브그래프의 모든 멤버 노드들(경계 노드들 포함)간의 최단 경로[1]를 미리 계산하기 때문이다.

Goldman et. al.은 HEPV와 유사한 Hub Indexing 방법을 제안했다[19]. 두 방법의 주된 차이점은 각 서브그래프에 대한 미리 계산된 경로 정보의 용량이다. Hub Indexing 방법에서는 HEPV에서와 같이 각 서브그래프에 대한 모든 쌍의 최단 경로를 미리 계산하지 않고, 각 서브 그래프의 경계 노드들과 내부 노드들(예를 들어, 각 서브 그래프의 경계가 아닌 노드들) 사이의 최단 경로를 미리 계산한다. 이러한 HEPV와 Hub Indexing 방법은 대용량의 디지털 로드맵을 고려할 때 심각한 저장 오버헤드 문제를 야기할 것이다. 이 문제와 관련하여 Shekhar et. al. [20]은 미리 계산되어 저장된 경로정보를 이용한 최단거리 계산시간과 저장 공간 오버헤드사이의 장단점을 분석하였다. 그들의 최단 경로를 계산하는데 있어서 어느 정도로 최단경로 정보를 미리 계산하여 저장하여야 효과적인 저장공간 오버헤드를 갖는지에 대하여 분석하고 있다.

본 논문에서는 최적의 최소 비용 경로를 효율적으로 계산하기 위해서, 디지털 로드맵과 같은 매우 큰 재귀적

1) 여기서의 최단경로는 그래프 전체에 대한 최단경로(Globally optimal shortest path)가 아니라 각 서브 그래프만을 사용하여 계산된 지역적인 최단경로(Locally optimal shortest path) 이다.

관계를 모델링한 *HiTi* 그래프 모델을 개발하였다. *HiTi* 그래프 모델은 Shekhar et. al[20]에서 추천한 접근방법을 따른다. *HiTi* 그래프 모델의 기본개념은 큰 그래프를 작은 서브 그래프들로 나누고 각 서브 그래프의 경계노드들간의 최단 경로를 미리 계산하여 계층적인 방법으로 구조화 시켜 저장하는 것이다.

다중의 지리적 경계선(예를 들어, 도시, 주, 국가)의 레벨은 *HiTi* 그래프 모델의 계층 구조로 쉽게 매핑될 수 있다. 또한, 여러 레벨의 계층적 추상화는 큰 디지털 로드맵의 효율적인 저장 관리를 위한 기반으로 활용될 수 있다. 그러므로, 이것은 저장 공간이 제한적인 항법 시스템(예: 차량 항법 시스템)에 적합한 보다 제어된 저장 관리를 개발하기 위한 기반를 제공한다. *HiTi* 그래프 방법은 여러 가지 측면에서 다른 유사한 방법들과 상세하게 성능실험을 통하여 장·단점이 비교 분석되어 졌다. *HiTi* 그래프 모델에 기반하여, 본 논문에서는 단일한 출발지점과 목적지점에 대한 최소 비용 경로를 계산하는 *SPAH*라는 새로운 단일쌍 최단(최소 비용) 경로 알고리즘을 제안한다. 또한 본 논문에서는 *SPAH*를 통해 계산된 최단 경로가 최적이라는 사실을 증명하고 *SPAH*가 탐색 공간을 매우 효과적으로 줄인다는 사실을 실험적으로 보여줄 것이다. 더 나아가, 에지 비용(edge cost) 분포와 *HiTi* 그래프의 계층 레벨의 정도를 다양화함으로써 SPAH를 분석하였다.

논문의 나머지 부분은 다음과 같이 구성된다. 2절에서는 *HiTi* 그래프의 기본 개념에 대해서, 3절은 *HiTi* 그래프의 형식적인 프레임 워크 및 정의에 대해 논의한다. 4절에서는 *HiTi* 그래프를 이용하는 새로운 단일쌍(SPSP: Single Pair Shortest Path) 최단 경로 알고리즘 *SPAH*를 제안한다. SPAH의 성능 분석은 5절에서 이루어진다. 6절에서는 항법 시스템(road navigation)이 동적인 교통 상태를 반영하기 위해 필수적인 *HiTi* 그래프에 대한 효과적인 갱신 알고리즘을 소개한다. 또한 이 절에서는 *HiTi* 그래프 갱신 알고리즘의 효율성을 보여준다. 7절에서는 본 논문의 *HiTi* 그래프 방법을 이론적뿐만 아니라 실험적으로 다른 유사한 연구와 상세히 비교한다. 마지막으로, 8절에서 결론을 내린다.

2. HiTi 그래프의 기본 개념 설명

방향성 그래프 $G(V,E)$로 표현된 디지털 로드맵을 고려해 보자. V의 노드들은 디지털 로드맵에서 맵 객체들을 나타내는 이산된 격자 점에 대응된다. E의 에지 $(x,y,cost)$는 V의 노드 x와 y사이의 $cost$(예: 거리)를 가지는 연결에 대응된다. 이때, 임의의 경계선(예: 정치적인 지역 경계선)들은 디지털 로드맵 $G(V,E)$를 CROM (Component ROad Maps)의 배타적 집합으로 나눈다. 각 CROM은 CROM의 경계를 명시하는 경계 노드들을 가진 서브 그래프 SG로 생각할 수 있다. CROM들간의 연결은 그것들의 경계 노드의 연결에 의해 묘사된다. 만약 한 CROM의 경계 노드가 다른 CROM의 경계 노드에 직접 연결되어 있다면, 이 두 CROM은 직접 연결되어 있는 것이다. 이러한 종류의 연결을 CROM들간의 *between* 연결이라고 부른다. 각 CROM에 대해서, 그 CROM의 경계 노드들의 각 쌍에 대한 최소 비용 경로를 미리 계산한다. 이 미리 계산된 최소 비용 경로를 CROM의 *within* 연결이라고 한다. *within* 연결은 G에 대한 전체의 최소 비용 경로를 얻는 것이 아니라, 단지 CROM에 대한 지역적인 최소 비용 경로를 얻는 것이다. *between* 연결과 *within* 연결을 이용해서, 전체의 디지털 로드맵을 나눌 수 있다. CROM들에 대한 *between* 연결과 *within* 연결의 예를 그림 1에 나타낸다.

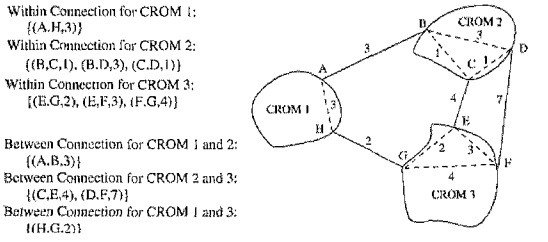


그림 1 CROM의 within과 between 연결 예

HiTi 그래프의 노드들은 CROM들의 경계 노드들로 되어 있고, edge는 CROM들에 대한 *within*과 *between* 연결로 되어있다. 하나의 CROM이 CROM들의 집합을 포함할 수 있기 때문에, 다중 수준(multi level) 계층이 만들어진다. 그러므로, 전체의 디지털 로드맵을 배타적으로 나누고 있는 최하위 level의 CROM 집합을 먼저 결정할 필요가 있다. 여기서 이러한 CROM들을 level 1 CROM이라고 부른다. 이때, $k \geq 2$에 대해서, level k CROM들의 집합은 지리적으로 인접한 level k-1 CROM들의 집합을 그룹화하여 순환적으로 구성될 수 있다. 이러한 level 1, 2, … ,k CROM들의 집합들은 균형 트리를 구성하며, 트리의 루트 노드는 전체의 디지털 로드맵이 된다.

3. HiTi 그래프의 형식적인 프레임 워크 및 정의

디지털 로드맵은 방향 그래프 $G(V,E)$로 볼 수 있다.

여기서, V의 각 노드는 디지털 로드맵의 맵 객체들(예: 도로의 교차로)을 나타낸다. E의 에지 (x,y,c)는 V에서 cost (예: 거리) c를 갖는 노드 x와 y사이의 연결에 대응된다. $G(V,E)$가 아래와 같은 조건을 만족시키며 서브 그래프 $SG_1^1(V_1^1,E_1^1), SG_2^1(V_2^1,E_2^1),\ldots,SG_{n_1}^1(V_{n_1}^1,E_{n_1}^1))$의 집합으로 분할된다고 가정하자. 여기서 $SG_i^1(V_i^1,E_i^1)$은 level 1인 *subgraph* i를 나타낸다 :

$$V_1^1\cup V_2^1\cup\cdots\cup V_{n_1}^1=V,\qquad E_1^1\cup E_2^1\cup\cdots E_{n_1}^1\subset E$$

$$V_i^1\cap V_j^1=\varnothing \text{ 그리고 } E_i^1\cap E_j^1=\varnothing,$$

$$\text{여기서 } 1\le i,j\le n_1 \text{ 그리고 } i\neq j$$

위의 모든 level 1 서브그래프들간의 연결은 PE^1가 나타내며 $E-(E_1^1\cup E_2^1\cup\cdots\cup E_{n_1}^1)$로 정의된다. 때, 각 $(x,y,c)\in PE^1$는 $i\neq j$에 대해서 $x\in v_i^1$와 $y\in V_j^1$라는 성질을 갖는다.

정의 3.1 N_i^1를 다른 서브 그래프 $SG_j^1(V_j^1,E_j^1)$의 노드와 직접 연결된 $SG_i^1(V_i^1,E_i^1)$에 속한 노드들의 집합이라고 하자. 여기서, $i\neq j$이고 $1\le i,j\le n_1$이다. N_i^1를 형식적으로 정의하면 $N_i^1=\{x\mid(x,y,c)\in PE^1\wedge x\in V_i^1\}\cup\{y\mid(x,y,c)\in PE^1\wedge y\in V_i^1\}$로 표시된다. N_i^1을 SG_i^1의 **level 1 경계 노드의 집합**이라고 한다.

정의 3.1에서 정의된 SG_i^1 V_i^1의 level 1 **경계** 노드들의 집합 N_i^1는 V_i^1에 속하지 않는 외부노드들에 직접 연결되어 있는의 노드들로 이루어져 있다. 예를 들어, 방향 그래프 G와 그것의 서브그래프 SG_1^1, SG_2^1, SG_3^1를 나타낸 그림 2를 고려해 보자. 서브그래프 SG_2^1의 집합 N_2^1은 $\{G,H,M,O\}$이다. 각 level 1인 서브그래프는 그것의 경계 노드들에 의해 묘사되고 식별된다. 그러므로, 그 경계 노드들은 하나의 level 1인 서브그래프에 배타적으로 속한다. SG_i^1의 경계 노드들에 기반을 두어, 정의 3.2는 SG_i^1의 level 1인 *between*과 *within* edge 집합 B_i^1과 W_i^1에 대한 형식적인 정의를 내린다.

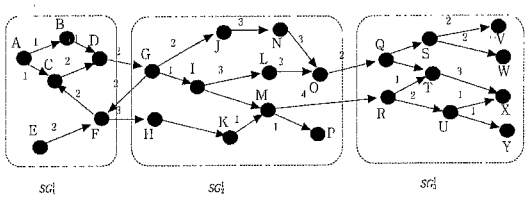


그림 2 방향 그래프 G와 그것의 level 1인 서브그래프 SG_1^1, SG_2^1, SG_3^1

정의 3.2 $\{SG_j^1(V_j^1,E_j^1)\mid j=1,n_1\}$을 그래프 $G(V,E)$에서 각각 대응되는 N_j^1를 가진 n_1개의 level 1인 서브그래프들의 집합이라고 하자. 이때, $1\le i\le n_1$에서 각 SG_i^1에 대해, $B_i^1=\{(x,y,c)\mid (x,y,c)\in PE^1\wedge x\in V_i^1\}$이고, $W_i^1=\{(x,y,f_c(x,y))\mid (x,y)\in(N_i^1\times N_i^1)\wedge x\xrightarrow{f_c(x,y)}y$ in $SG_i^1)\wedge x\neq y\}$이다. 함수 $f_c(x,y)$는 오직 SG_i^1안에서 계산된 최단 경로 비용(예: 노드 x에서 y까지의 최단경로)을 나타낸다.

표 1은 정의 3.2의 예로서 그림 2에서 보여준 세 개의 서브그래프에 대한 N_i^1, B_i^1, W_i^1에 대응되는 값을 보여준다.

표 1 그림 2에서의 SG_i^1에 대한 N_i^1, B_i^1, W_i^1

i	N_i^1	B_i^1	W_i^1
1	{D,F}	{(D,G,2), (F,H,3)}	{(F,D,4)}
2	{G,H,M,O}	{(O,Q,2), (M,R,4), (G,F,2)}	{(G,O,6), (G,M,3), (H,M,3)}
3	{Q,R}	∅	∅

지금까지, level 1인 서브그래프와 그것들과 관련된 내용 즉, level 1인 경계 노드 집합 그리고 *between*과 *within* edge 집합에 대해 소개를 하였다. 일반적으로, 이러한 level 1인 서브그래프와 그것들과 연관된 정보들에 기초하여, 어떠한 $k\ge2$에 대해서 level k인 서브그래프 SG_i^k와 이에 대응하는 N_i^k, B_i^k, W_i^k을 순환적으로 정의할 수 있다. 이 세부내용은 다음 정의에서 논의된다.

정의 3.3 $\Psi^{k-1}=\{SG_i^{k-1}(V_i^{k-1},E_i^{k-1})\mid i=1,n_{k-1}\}$을 $k\ge2$에서 그래프 $G(V,E)$의 n_k 개의 level k-1인 서브그래프들의 집합이라고 하자. 이때, level k인 서브그래프 $SG_i^k(V_i^k,E_i^k)$은 Φ 집합에 속한 모든 level 1인 서브그래프들에 의해 유도되는 서브그래프로 정의되며 여기서 Φ는 $\Phi\subseteq\Psi^{k-1}$과 $|\Phi|\ge2$인 조건을 만족하여야 한다.

Ψ^k에 포함된 모든 level k인 서브그래프들이 정의된 후에는 다음 세 가지 요구사항이 만족되어야 한다:

1. Ψ^{k-1}에 포함된 각각의 level $k-1$ 서브그래프는 Ψ^k에 포함된 임의의 level k 서브그래프의 서브그래프이다.
2. Ψ^k에 포함된 모든 level k 서브그래프들은 서로 겹쳐지지 않는다. 즉,

$$V_1^k\cup V_2^k\cup\cdots\cup V_{n_r}^k=V,\qquad E_1^k\cup E_2^k\cup\cdots E_{n_k}^k\subset E$$

$V_i^k \cap V_j^k = \emptyset$ 그리고 $E_i^k \cap E_j^k = \emptyset$, 여기서 $1 \le i, j \le n_k$ 그리고 $i \neq j$

3. level k인 *between* edge 집합들에 속하는 모든 edge들은 level $k-1$인 *between* edge 집합으로부터 제거된다. 그 결과 $(\cup_{j=1}^{n_{k-1}} B_j^{k-1} \cap \cup_{j=1}^{n_k} B_j^k) = \emptyset$ 이 성립하여야 한다.

앞의 처음 두 요구사항이 만족됨으로서, $\cup_{j=1}^{k+1} \Psi^j$에 속한 모든 서브그래프들은 서로간의 서브 그래프 포함 관계가 균형 트리(Balanced Tree)로 서로 연관되어 진다. 여기서 Ψ^{k+1}는 트리의 루트 노드를 나타내는 $G(V, E)$ 단지 하나 만을 포함하고 있다. 이 균형 트리는 *subgraph tree* (ST)라고 명명한다. ST의 루트 노드는 level $k+1$ 서브그래프라고 생각할 수 있고, 그것은 모든 level k 서브그래프들 자식 노드로 갖는다. 순환적으로, level k 서브그래프를 가리키는 각 노드는 level $k-1$ 서브그래프들을 자식노드로 갖는다. ST의 모든 말단 노드들은 level 1인 서브그래프들을 가리킨다. 이 서브그래프 트리의 루트 노드가 level $k+1$ 서브그래프(즉, $G(V, E)$)를 가리키면 level $k+1$ ST라고 명명된다. 다음 그림 3은 level 3인 서브그래프 트리의 예를 보여준다. 이 트리 구조는 level 2인 3개의 서브그래프들(즉, SG_1^2, SG_2^2, SG_3^2) 의 노드들에 의해 유도되는 level 3인 서브그래프 $SG_1^3 = G$를 보여준다. 또한 이 트리 구조는 level 1인 6개의 서브그래프 SG_1^1, SG_2^1, SG_3^1, SG_4^1, SG_5^1, SG_6^1의 노드들에 의해 유도되는 3개의 level 2 서브그래프 SG_1^2, SG_2^2, SG_3^2를 보여준다.

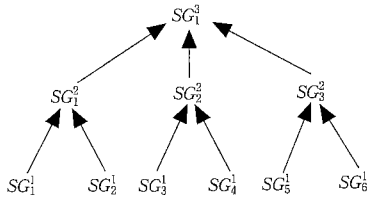


그림 3 level 3인 서브그래프 트리 ST의 예

$1 \le i \le n_k$에서 ST의 각 level k 서브그래프 SG_i^k에 대해서, 대응되는 level k *경계* 노드 집합(즉, N_i^k)과 level k *between*과 *within* edge 집합들(즉, B_i^k, W_i^k)이 존재한다. 그것들은 정의 3.4에서 형식적으로 정의된다.

정의 3.4 $PE^k = E - (E_1^k \cup E_2^k \cup \cdots \cup E_{n_k}^k)$라고 하자. 이 때, $\cup_{j=1}^{n_{k-1}} B_j^{k-1} = \cup_{j=1}^{n_{k-1}} B_j^{k-1} - PE^k$, $N_i^k = \{x \mid (x, y, c) \in PE^k \wedge x \in V_i^k\} \cup \{y \mid (x, y, c) \in PE^k \wedge y \in V_i^k\}$, $B_i^k = \{(x, y, c) \mid (x, y, c) \in PE^k \wedge x \in V_i^k\}$, 그리고 $W_i^k = \{(x, y, f_c(x, y)) \mid (x, y) \in (N_i^k \times N_i^k) \wedge x \xrightarrow{f_c(x,y)} y \text{ in } SG_i^1 \wedge x \neq y\}$ 이다. 함수 $f_c(x, y)$는 오직 SG_i^k에서 계산된 노드 x로부터 y까지의 최단 경로 비용을 나타낸다.

위 세 가지 정의를 사용해서 서로 다른 level의 경계 노드, *within*과 *between* edge 집합들에 대한 값을 얻을 수 있다. *within*과 *between* edge 집합들에 기반을 둔 *HiTi* 그래프의 형식적인 정의가 다음과 같이 주어진다:

정의 3.5 *HiTi* 그래프는 서브그래프 트리 ST의 모든 노드들에 연관되어 있는 *between*과 *within* edge들에 의해서 정의된 레이블된(labeled) 방향 그래프이다. level $k+1$ 서브그래프 트리 ST는 level k *HiTi* 그래프를 정의하며 $H^k(P^k, A^k)$로 표시된다. 여기서, $P^k = \cup_{i=1}^{k} \cup_{j=1}^{n_i} N_j^i$ 이고, $A^k = \cup_{i=1}^{k} \cup_{j=1}^{n_i} \{B_j^i \cup W_j^i\} \times \{i\}$이다.

이 절에서는 level k *HiTi* 그래프의 기본적인 개념과 형식적인 정의를 소개하였다. level k *HiTi* 그래프가 갖는 오버헤드는 *within* edge 집합들의 크기 즉, $|\cup_{i=1}^{k} \cup_{j=1}^{n_i} W_j^i|$ 에 의해 결정되는 사실은 쉽게 알 수 있다. 그러므로, *HiTi* 그래프 방법이 효율적이기 위해서는 $|\cup_{i=1}^{k} \cup_{j=1}^{n_i} W_j^i|$ 가 최소화되어야 한다. *within* edge 집합들의 크기는 생성되는 경계 노드들의 수에 의해 결정되므로, 각 level 서브그래프에 대한 경계 노드들의 수를 최소화함으로써 그래프 $G(V, E)$를 효율적으로 계층 분할하는 방법을 찾는 것은 중요한 일이다. 본 논문에서는 그래프 $G(V,E)$가 일정한 경계 노드수를 갖는 level 서브그래프로 이미 분할되어 있다고 가정하였다.

하지만 디지털 로드맵과 같은 평면 그래프를 경계노드수가 작은 서브그래프들로 효율적으로 분해하는 체계적인 방법들이 제안되어왔으며 이에 관한 연구결과가 보고되었다. McCormick et. al.은 지수 배의 복잡도를 갖는 최적의 그래프 분할 알고리즘을 제안했다[21]. Lipton과 Tarjan[22]이 처음으로 제시한 평면 그래프 분할은 Miller[23, 24]에 의한 평면 그래프의 계층적 분한을 산출하기 위해 이용되었다. Houstma et. al.은 효과적인 중심 노드들을 수동적으로 선택하는 것에 의존하는 center-based greedy 그래프 분해 알고리즘을 제안했다[25]. Huang et. al.은 그후에 그래프를 공간 근접성에 기초하여 분할하는 공간 분할이라고 불리는 훌륭한 평면 그래프 분할 알고리즘을 제안하였다. 그들의

실험적 분석은 공간 분할 방법이 디지털 로드맵을 서브그래프들로 분해하는데 있어서 각 서브그래프의 경계 노드들의 수를 최소화시키는데 가장 적절하다라는 사실을 보여주고 있으며 *HiTi* 그래프를 효과적으로 구축하는데 적합하다고 생각된다.

4. HiTi 그래프에 기반한 최적 최단 경로 계산

이 절에서는 $G(V,E)$상의 최단 경로를 계산하기 위하여 *HiTi* 그래프 사용방법에 대하여 논의한다. 먼저, 본 논문의 나머지 부분에서 이용될 기본적인 표기들을 소개한다.

정의 4.1 SG를 그래프 $G(V,E)$의 서브그래프라고 하자. 이때, $PC_{SG}(x,y)$와 $SPC_{SG}(x,y)$는 각각 SG에서의 노드 x에서 y로의 경로 비용과 최단 경로 비용을 나타낸다.

정의 4.2 집합 X가 서브그래프들로 이루어져 있다고 가정하자. 이때, $S_B(X)$와 $S_W(X)$는 각각 X에 포함된 모든 서브그래프들과 연관된 **between**과 **within** edge 집합들로 이루어져 있다. $S_{BW}(X)=S_B(X)\cup S_W(X)$이고, $S_N(X)$는 X에 속해있는 모든 서브그래프들과 연관된 **경계** 노드 집합들로 이루어져 있다.

정의 4.3 SG_i^l와 SG_j^l은 level k인 서브그래프 트리 ST에서 정의된 2개의 구별된 level l인 서브그래프들이라고 하자. 이때, $LUB_{ST}(SG_i^l, SG_j^l)$은 SG_i^l와 SG_j^l의 최소 레벨된 공통의 조상(Least leveled common ancestor) 서브그래프이다.

정의 4.4 집합 X가 level k+1인 서브그래프 트리 ST에서 정의된 서브그래프들로 이루어져 있다고 가정하자. 이때, $S_A^t(X)=\{\ y \mid y$ is a level t ancestor subgraph of a subgraph in $X\}$, $S_A(X)=\bigcup_{t=1}^{k+1} S_A^t(X)$, $S_c(X)=\{\ y \mid y$ is a direct child node of a subgraph in $X\ \}$. 여기서, 서브그래프 트리 ST에서의 모든 level 1 서브그래프 SG_i^1에 대하여 $S_c(\{SG_i^1\})=\{SG_i^1\}$로 정의된다.

정의 4.2와 4.3에 대한 예는 그림 4에서 보여주는 level 4 서브그래프 트리를 통해 설명된다.

$X=\{SG_8^1, SG_{11}^1\}$라고 가정하자.

이때, $S_A^1(X)=\{SG_8^1, SG_{11}^1\}$,

$S_A^2(X)=\{SG_4^2, SG_5^2\}$,

$S_A(X)=\{SG_8^1, SG_4^2, SG_2^3, SG_1^4, SG_{11}^1, SG_5^2\}$,

$S_C(X)=\{SG_8^1, SG_{11}^1\}$,

$S_C(S_A^2(X))=\{SG_8^1, SG_9^1, SG_{10}^1, SG_{11}^1, SG_{12}^1\}$,

$S_B(X)=\{B_8^1, B_{11}^1\}$,

$S_W(X)=\{W_8^1, W_{11}^1\}$,

$S_N(X)=\{N_8^1, N_{11}^1\}$,

그리고 $LUB_{ST}(SG_8^1, SG_{11}^1)$는 SG_2^3이다.

4.1 HiTi 그래프 상에서 계산된 최단 경로의 최적성

HiTi 그래프 상에서 계산된 최단 경로 비용의 최적성을 증명하기 위해서, 다음 네 가지 정리를 제시한다. 처음 세 가지 정리는 마지막 정리를 증명하기 위해서 이용될 것이다. 마지막 정리는 *HiTi* 그래프 상의 최단 경로 계산의 최적성을 보여준다.

정리 4.1은 level l인 서브그래프의 경계 노드들의 어떤 쌍에 대한 최단 경로 비용이 level l인 서브그래프의 자식 서브그래프에서 level l-1인 *between*과 *within* edge 집합들에 대해서 계산된 것과 같다라는 사실을 보여준다. 더 나아가, 이 정리는 level l인 *within* edge 집합들을 효율적으로 계산하기 위한 기반을 제공한다. 즉, 정리 4.1을 적용하여, 오직 level l-1인 *within*과 *between* edge 집합들, 즉, $S_{BW}(S_C(SG_i^l))$을 이용해서 level l인 *within* edge 집합, W_i^l을 순환적으로 계산할 수 있다.

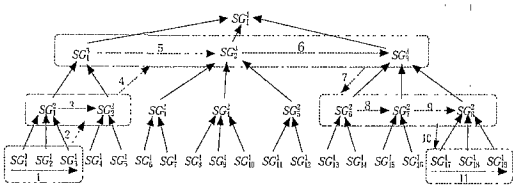


그림 4 level 4인 서브그래프 트리 ST의 예

정리 4.1 SG_i^l을 $k\geq 1$이고 $1\leq l\leq k+1$대해서 level $k+1$ 서브그래프 트리 ST에서의 level l서브그래프 노드라고 하자. 또, $S_C(SG_i^l)=\{SG_1^{l-1}, SG_2^{l-1}, \ldots, SG_p^{l-1}\}$라고 하자. $x, y\in S_N(S_C(SG_i^l))$인 어떤 경계 노드 쌍 x와 y대해서, $SPC_{SG_i^l}(x,y)=SPC_{S_{BW}(S_C(SG_i^l))}(x,y)$가 성립한다.

증명 : 증명은 부록에 작성되어 있다.

출발점과(source)와 목적지(destination) 노드가 SG_i^l의 같은 자식 서브그래프 SG_q^{l-1} 안쪽에 있고 양쪽 노드 모두 level l-1 경계 노드가 아닐 때, 정리 4.2는 level l 서브그래프 SG_i^l 상에서 최단 경로 비용을 계산하기 위한 효율적인 방법을 제공한다. 그 방법은, 계산을 위해 SG_i^l의 모든 edge들을 탐색하지 않고, SG_q^{l-1}의 edge들과 SG_i^l의 모든 자식 서브그래프들의 level l-1 *between*과

within edge 집합만을 탐색한다.

정리 4.2 SG_i^l을 $k\geq 1$이고 $1\leq l\leq k+1$대해서 level k +1 서브그래프 트리 ST에서의 level l 서브그래프 노드라고 하자. 또, $S_C(SG_i^l)=\{SG_1^{l-1}, SG_2^{l-1}, \ldots, SG_p^{l-1}\}$라고 하자. $1\leq q\leq p$인 모든 노드 쌍 $x, y\in SG_q^{l-1}$에 대해서, 다음이 성립한다:

$$SPC_{SG_i^l}(x, y)=SPC_{SG_u^{l-1}\cup S_{BW}(S_C(SG_i^l))}(x, y).$$

증명: 증명은 부록에 작성되어 있다.

출발점과(source)와 목적지(destination) 노드가 SG_i^l의 각기 다른 두 개의 자식 서브그래프 SG_u^{l-1}와 SG_v^{l-1}안에 각각 존재할 때, 정리 4.3은 level l인 서브그래프 SG_i^l 상에서 최단 경로 비용을 계산하기 위한 효율적인 방법을 제공한다. 그 방법은, 계산을 하기 위해서 SG_i^l의 모든 edge들을 탐색하지 않고, SG_u^{l-1}와 SG_v^{l-1}의 edge들과 SG_i^l의 모든 자식 서브그래프의 level l-1 *between*과 *within* edge 집합만을 탐색한다.

정리 4.3 SG_i^l을 $k\geq 1$이고 $1\leq l\leq k+1$대해서 level k +1 서브그래프 트리 ST에서의 level l인 서브그래프 노드라고 하자. 또, $S_C(SG_i^l)=\{SG_1^{l-1}, SG_2^{l-1}, \ldots, SG_p^{l-1}\}$라고 하자. $1\leq u,\ v\leq p$이고 $u\neq v$인 모든 노드 쌍 $x\in SG_u^{l-1}$, $y\in SG_v^{l-1}$에 대해서, 다음이 성립한다:

$$SPC_{SG_i^l}(x, y)=SPC_{SG_u^{l-1}\cup SG_i^{l-1}\cup S_{BW}(S_C(SG_i^l))}(x, y).$$

증명: 증명은 부록에 작성되어 있다.

정리 4.1, 4.2, 4.3에 기반을 둔 정리 4.4는 $G(V,E)$의 모든 노드 쌍 (x, y)의 최단 경로 비용은 level k *HiTi* 그래프의 선택된 부분과 반드시 구별되지 않아도 되는 2개의 level 1 서브그래프들로부터 계산한 것과 G(V,E)로부터 계산한 것과 동일하다는 사실을 증명한다.

정리 4.4 $k\geq 1$인 Level k+1 서브그래프 트리 ST를 $G(V, E)$로부터 구성하고 ST는 level k인 *HiTi* 그래프 $H^k(P^k, A^k)$를 정의한다고 하자. 어떤한 노드 쌍 $x\in SG_i^1$, $y\in SG_j^1$대해서도 $D=S_{BW}(S_C(S_A(\{SG_i^1, SG_j^1\})))$인 $SPC_G(x, y)=SPC_{SG_i^1\cup SG_j^1\cup D}(x, y)$가 성립한다.

증명: 증명은 부록에 작성되어 있다.

정리 4.4는 HiTi 그래프가 최단 경로 계산을 위해 필요한 탐색 공간을 상당히 줄이는 방법을 알기 쉽게 보여 주고 있다. 즉, HiTi 그래프를 사용하지 않으면, 탐색 공간은 $G(V, E)$전체가 된다. 게다가, 정리 4.4는 효율적인 저장 관리를 가능하게 하는데 이는 저장 공간이 제한된 항법 시스템(예를 들어, 자동차 항법 시스템)에 적합하다. 효율적인 저장관리가 가능한 이유는 최단 경로를 계산하기 위해 전체의 $G(V, E)$가 필요하지 않기 때문이다. 즉, 정리 4.4를 이용하면, 오직 level k HiTi 그래프의 선택된 부분과 level 1인 2개의 edge 집합들(예를 들어, source 노드에 대한 것 하나와 destination에 대한 것 하나)이 필요할 뿐이다. 정리 4.4에 대한 예를 보여주기 위해 그림 4의 level 4 서브그래프 트리를 예로 든다. SG_1^1의 출발점(Source) 노드 *START*로부터 SG_{19}^1의 목적지(destination) 노드 *DEST*까지의 최단 경로를 찾는다고 가정하자. 이때, 필요한 최대 탐색 공간은 그림 4에서 점선 사각형으로 둘러싸인 서브그래프들이 가지고 있는 *between*과 *within* edge 집합들과 $E_1^1\cup E_{19}^1$이다.

4.2 최단 경로 알고리즘 SPAH

이제, 정리 4.4에 근거하여 *SPAH*라는 최단 경로 알고리즘을 설명한다. *SPAH* 알고리즘은 그림 5에서 보여진다. *SPAH*는 두 단계로 수행되어 진다. 첫 번째 단계는 정리 4.4를 이용하여, $\{V_1^1, V_2^1, \ldots, V_{n_1}^1\}$에 속한 노드들의 부분 집합을 edge level 숫자로 표시를 하면서 필요한 탐색 공간을 결정한다. 각각의 표시된 edge level 숫자는 *SPAH*가 탐색 할 수 있는 edge의 최하 level 숫자를 나타내는데 이러한 edge들은 숫자로 표시된 노드들과 인접(incident)되어져 있다. 두 번째 단계에서 *SPAH*는 첫 번째 과정에서 얻어진 결정된 탐색 공간에 부분적으로 수정되어진 A^* 알고리즘을 적용한다.

A^*는 함수 $f(u, DEST)$를 사용하는데 이 함수는 노드 u에서부터 *DEST*까지의 최단 경로 비용을 추정하는데 사용된다. 디지털 로드 맵에서, 함수 $f(u, DEST)$는 노드 u와 *DEST*사이의 유클리드의(Euclidean) 거리를 계산한다. 이것은 디지털 로드맵상의 모든 노드들이 위도와 경도의 좌표로 표시되기 때문에 가능하다. $(u.x, u.y)$와 $(DEST.x, DEST.y)$를 노드 u와 *DEST*의 대응되는 좌표라고 가정하면, $f(u,DEST)$는 $\sqrt{(DEST.x-u.x)^2+(DEST.y-u.y)^2}$를 계산한다. 노드 u와 *DEST*사이의 계산된 유클리드의 거리는 실제 최단 경로보다 항상 과소로 계산 추정되기 때문에, A^*는 최적의 최단 경로를 찾는다. A^* 알고리즘의 최적성에 대한 자세한 증명은 [26]에 나와있다.

*SPAH*는 최적의 최단 경로 비용을 찾기 위해서, SG_i^1의 START 노드로부터 SG_j^1의 DEST 노드까지의 표시된 노드들과 인접되어 있는 edge들을 탐색한다. 이 계산을 위해서, *SPAH*는 *within* edge 집합들에 저장된 미리 계산된 최단 경로 비용을 이용한다. 미리 계산된 최단

```
begin
/* Assume we have a level k HiTi graph defined on a level k+1 ST; */
/* {SG_1^1, SG_2^1, ..., SG_{n_1}^1} and H*(P*, A*) are maintained as adjacency lists; */
/* All the edges in SG_t^1 for 1 ≤ t ≤ n_1 are assumed to be level 0 edges; */
/* All the edges in A* have their corresponding edge level */

Step 1:
  For 1 ≤ i, j ≤ n_1, find SG_i^1 and SG_j^1 where START ∈ SG_i^1 and DEST ∈ SG_j^1;
  Let l be the level number of LUB_{ST}(SG_i^1, SG_j^1);
  for (p = k+1; p > l; p--)    /* S*_A(SG_i^1) = S*_A(SG_j^1) */
    Mark all nodes in S_N(S_G(S*_A(SG_i^1))) with level p-1;
  if (l = 1)    /* SG_i^1 = SG_j^1 */
    Mark all nodes in SG_i^1 with 0;
  else {
    for (p = l; p > 1; p--) {
      Mark all nodes in S_N(S_G(S*_A(SG_i^1))) with level p-1;
      Mark all nodes in S_N(S_G(S*_A(SG_j^1))) with level p-1; }
    Mark all nodes in SG_i^1 and SG_j^1 with 0; }

Step 2:
  Let λ(x) = ∞ for all the marked nodes x and λ(START) = 0;
  FSet = { START }; ESet = ∅;
  while (FSet ≠ ∅) {
    Select u from FSet with minimum λ(u) + f(u, DEST);
    /* the function f(u, DEST) estimates the Euclidean distance
       from node u to DEST */
    FSet = FSet - { u }; ESet = ESet ∪ { u };
    if (u = DEST) stop;
    Let r be the highest level number of the edges incident on node u;
    if (u is marked) {
      Let β be the marked level number on node u;
      for each edge u --z--> v with the levels from β to r {
        if ((v ∉ FSet) and (v ∉ ESet)) {
          λ(v) = λ(u) + z;
          FSet = FSet ∪ { v }; }
        else {
          if (λ(v) > λ(u) + z) λ(v) = λ(u) + z; } } } }
end
```

그림 5 SPAH: START에서 DEST 노드까지의 최단 경로 비용 계산

경로 비용이 반드시 $G(V,E)$상의 전체 최단 경로가 아니지만, *SPAH*는 정리 4.4에서 증명한 바와 같이 서브그래프 트리를 사용하여 정확한 전체 최단 경로를 제공한다. 서브그래프 트리는 전체 최단 경로의 계산에서 탐색이 불필요한 서브그래프들에 관한 정보를 제공한다. *SPAH*의 edge 탐색은 두 단계 즉, 상승(ascending) 단계와 하강(descending) 단계로 이루어져 있다. *SPAH*는 상승 단계에서는 비감소(non-decreasing) edge level 순서로 하강 단계에서는 비증가(non-increasing) edge level 순서로 edge들을 탐색한다. 여기서 주의하여야 할 점은 각각의 고려된 경로들은 자신들만의 프로세싱 쓰레드(thread)를 갖는다. 다시 말하면, 어떤 경로들은 여전히 상승 단계에 있는데 반해, 다른 나머지 경로들은 하강 단계에 있을 수 있다.

앞의 그림 4는 또한 출발점 노드와 목적지 노드들이 각각 SG_1^1와 SG_{19}^1에 있을 경우, *SPAH*에서 탐색이 진행되는 과정에 대한 예를 보여준다. 그림에서 5개의 점선 사각형은 서브그래프들을 포함하고 있는데, 이 서브그래프들의 *within*과 *between* edge 집합들은 *SPAH*의 단계 1에서 선택되어지는 필요한 탐색 공간이다. 단계 2에서 *SPAH*는 먼저 서브그래프 $SG_1^1(V_1^1, E_1^1)$의 edge들을 SG_1^1의 경계 노드에 도착할 때까지 탐색한다. 경계노드에 도착한 후 SPAH는 상승 단계로 진입하며 이 단계에서는 비감소 level 숫자를 갖는 *between*과 *within* edge들이 탐색된다. 상승 단계에서 edge 탐색의 순서는 1, 2, 3, 4, 5, 6의 숫자가 표시된 점선 화살표로 보여주고 있다. 이 숫자는 탐색 진행의 순서를 나타낸다. *SPAH*가 SG_9^3의 경계 노드를 만날 때 하강단계로 진입하며 이 단계에서는 비 증가 level 숫자를 가진 *between*과 *within* edge들이 탐색된다. 하강 단계에서 edge 탐색의 순서는 7, 8, 9, 10, 11의 숫자가 표시된 점선 화살표로 보여주고 있다. *SPAH*가 SG_{19}^1의 경계 노드들에 도착한 후에는 목적지 노드에 도착할 때까지 $SG_{19}^1(V_{19}^1, E_{19}^1)$에 속해있는 edge들을 탐색한다.

*SPAH*로부터 최단 경로 비용을 구한 후, 운전자는 최단 경로 상에서 0보다 큰 level 숫자를 갖는 상위 level edge들에 대한 더 상세한 경로 정보를 필요로 할 수 있다. 이것은 상위 level edge를 세분화함으로써 이루어진다. 상위 level *within* edge들은 하위 level edge들로 나타냄으로써 세분화된다. 이를 위해서, 모든 *within* edge들에 대한 실제 최단 경로 정보를 저장한다. 상위 level *between* edge는 lower level edge들에 의해 세분화되지 않는다. 그 이유는 $\cup_{i=1}^{k}\cup_{j=1}^{n_i}B_j^i$에서의 모든 *between* edge들이 $G(V,E)$의 edge들이기 때문이다.

5. 성능 분석

SPAH 알고리즘의 분석을 위해서, 4개의 인접한 노드를 가진 2차원 격자 그래프 $G(V,E)$를 생성한다. 2차원 격자 그래프는 디지털 로드 맵의 전형적인 모델로 고려되어진다[16, 31]. 격자 그래프 G에서, V와 E는 각각 800×800 노드들과 4×800×799 방향 edge들을 가지고 있다. $G(V,E)$로부터, 각 level 1 서브그래프 $SG_i^1(V_i^1, E_i^1)$가 $|V_i^1| = 100\times100$와 $|E_i^1| = 4\times100\times99$를 가지는 level 4 서브그래프 트리 ST를 생성한다. 그러므로, level 4 서브그래프 트리 ST는 그림 6과 같이, 64개의 level 1, 16개의 level 2, 4개의 level 3, 1개의 level 4인 서브그래프들로 이루어져 있다. 이렇게 만들어진 level 4 서브그래프 트리 ST는 다음절에서 *SPAH*를 분석하기 위해서 필요한 level 3 *HiTi* 그래프를 생성하기 위하여 사용될 것이다.

5.1 SPAH와 A* 알고리즘의 성능 비교

*SPAH*가 ([31]에서 기술된) 전통적인 A* 알고리즘보다 탐색 공간을 많이 줄인다는 사실을 보여주기 위해서,

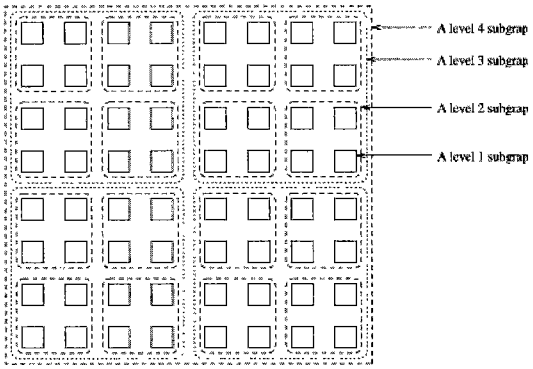


그림 6 level 4 서브그래프 트리에 의하여 분할된 800×800 격자 그래프

$10 \le |N_i^1| \le 20$이고 edge cost가 5개의 서로 다른 seed를 가진 균일 분포(Uniform Distribution) [100, 120]로부터 생성된 5개의 level 3인 *HiTi* 그래프를 만든다. 여기서 $|N_i^1|$은 level 1 서브그래프 SG_i^1상에서 정의된 모든 경계 노드들의 총 개수를 나타낸다. Level 3인 각 *HiTi* 그래프와 각 평면 격자 그래프에 대해서, 임의로 고정된 출발점(source)과 목적지(destination) 노드들의 쌍에 대한 20개의 서로 다른 최단 경로를 계산한다. H_n과 A_n을 각각 *SPAH*와 A*에 의해 방문된 edge들의 총 개수라고 하자.

본 논문에서는 $f(t, DEST)$의 추정 값을 edge cost의 값을 고려하여 표준화(normalization)하기 위해서, 이 추정값에 100을 곱한다. 다시 말해, 숫자 100이 위의 표준화에 사용되는 이유는 두 인접한 노드 좌표 사이에 대한 유클리드 거리가 1이고 두 노드사이의 edge cost가 적어도 100이기 때문이다. *SPAH*와 A*는 비율 A_n/H_n를 관찰함으로서 비교해 본다. 이때, 이 값은 같은 source와 destination 노드를 가진 5개의 서로 다른 level 3인 *HiTi* 그래프와 평면 격자 그래프 상에서 평균을 낸 값이다. 그 값은 그림 7에 보여진다. 이 그림에서, x 축의 1에서 20까지 숫자는 <*source, destination*>를 갖는 20개의 최단 경로 비용들이 증가하는 것을 나타낸다.

그림 7은 *SPAH* 알고리즘이 전통적인 A* 알고리즘보다 탐색 공간을 얼마나 효율적으로 줄이고 있는지를 명백하게 보여준다. source에서 destination까지의 최단 경로 비용이 증가함에 따라, 비율 A_n/H_n이 급격히 증가한다는 것은 흥미로운 사실이다. 이것은 A* 알고리즘이 탐색하는데 필요한 탐색 공간이 지수 적으로 증가하는 반면, *SPAH*의 탐색 공간은 *HiTi* 그래프의 계층적인 구조로 인해 매우 느리게 증가하기 때문이다.

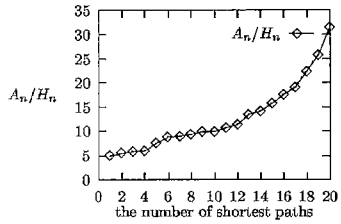


그림 7 A*와 SPAH간의 탐색공간 비교

5.2 Edge 비용 분포에 의한 영향 분석

이 절에서는 여러 형태의 edge 비용 분포가 *SPAH*의 성능에 미치는 영향을 연구한다. 이를 위해, $10 \le |N_i^1| \le 20$를 만족하는 4×5(즉, 5개의 seed를 가진 4개의 균일 분포)개의 level 3 *HiTi* 그래프를 생성한다. 4개의 균일 분포(Uniform Distribution) [100,120], [100, 200], [100,300], [100,500]은 각각 20%, 100%, 200%, 400%에 해당하는 edge 비용 편차에 대응된다. 위에서 언급한 각각의 level 3 HiTi 그래프에 대하여 무작위로 생성된 50개의 서로 다른 < *source, destination* > 쌍의 최단 경로 비용을 *SPAH*를 적용하여 계산한 후 그 계산에 드는 비용의 평균을 구한다. 평가함수 $f(u, DEST)$가 유클리드 거리추정과 0(즉, 거리추정이 없음)을 제공했을 때의 *SPAH*를 각각 *MA*와 *MD*가 나타낸다고 하자. 그림 8은 *SPAH*의 성능에 대한 edge 비용 분포의 영향을 H_n의 관점에서 보여주고 있다. 여기서 주의할 점은 H_n의 값은 5개 seed에 의해 생성된 5개의 level 3 HiTi 그래프로부터 얻어진 평균값이다.

$f(u, DEST)$가 유클리드 거리를 추정했을 때, *SPAH*의 성능은 edge 비용의 편차가 증가함에 따라 저하된다. 그 주된 원인은 편차의 증가가 최단 경로에 대한 유클리드 거리 추정에 대한 질을 낮추기 때문이다. 이러한 유클리드 거리 추정 정확도 감소 현상은 *MA*의 초기

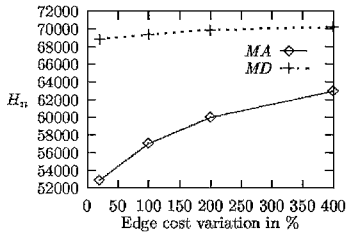


그림 8 MA와 MD 성능에 대한 Edge 비용의 영향

성능(즉, 20%에서 200%사이) 상에서 그 나머지 (즉, 200%에서 400%사이)보다 더 심각한 영향을 받는다. *MA*와 달리, *MD*는 다양한 edge 비용 분포에 대하여 매우 안정된 성능을 제공한다. 위의 분석과 유클리드 거리 추정이 주는 계산적인 오버헤드를 고려할 때, *MD*는 *MA*보다 큰 edge 비용 편차를 가진 디지털 로드 맵에 더 적합하다.

5.3 계층 level 수의 영향

이 절에서는 여러 가지 level의 *HiTi* 그래프가 *SPAH* (즉, *MA*와 *MD*)의 성능에 대하여 어떠한 영향을 주는지를 분석하여 보았다. 이를 위하여 우선 level 1 서브그래프들의 동일한 집합으로부터 서로 다른 level의 *HiTi* 그래프를 생성한다. 다시 말하면, 2(즉, $4\le|N_i^1|\le8$ 그리고 $10\le|N_i^1|\le20$) × 5(즉, 5개의 seed를 가진 edge 비용 분포 [100,200])개의 level 3 *HiTi* 그래프들을 생성한다. 비슷하게, 2×5 level 1과 level 2인 *HiTi* 그래프를 생성한다. 이때, 5.2절에서 했던 방법으로 *MA*와 *MD*의 평균 H_n을 측정한다. 이 결과는 다음의 표 2와 3에서 보여 주고 있다.

표 2 $10\le|N_i^1|\le20$일 때, *HiTi* 그래프 level이 H_n에 미치는 영향

	level 1인 *HiTi* 그래프	level 2인 *HiTi* 그래프	level 3인 *HiTi* 그래프
*MA*에 대한 H_n	57539	57421	57161
*MD*에 대한 H_n	72972	72756	70375

표 3 $4\le|N_i^1|\le8$일 때, *HiTi* 그래프 level이 H_n에 미치는 영향

	level 1인 *HiTi* 그래프	level 2인 *HiTi* 그래프	level 3인 *HiTi* 그래프
*MA*에 대한 H_n	51348	51014	51498
*MD*에 대한 H_n	60173	60132	60714

표 2와 3에서 알 수 있듯이, 높은 level의 *HiTi* 그래프는 낮은 level의 *HiTi* 그래프보다 더 나은 *SPAH*의 성능을 보장하지 않는다. *SPAH* 성능은 그것의 평균 탐색 공간에 따라 좌우된다. Y^k로 표시되는 level k *HiTi* 그래프에 대한 *SPAH*의 평균 탐색 공간은 다음과 같이 공식화된다 :

$$Y^k=2\cdot|E^1|+\sum_{i=1}^{n_k}|W_i^k|+\frac{2}{n_k}\sum_{i=1}^{n_{k-1}}|W_i^{k-1}|+\frac{2}{n_{k-1}}\sum_{i=1}^{n_{k-2}}|W_i^{k-2}|+\cdots+\frac{2}{n_3}\sum_{i=1}^{n_2}|W_i^2|+\frac{2}{n_2}\sum_{i=1}^{n_1}|W_i^1|$$

$|E^1|$은 level 1인 서브그래프의 edge 총 개수를 나타내고, n_l은 $1\le l\le k$일 때, level l인 서브그래프의 총 개수를 나타낸다. 만약 $p>q$일 때 $Y^p<Y^q$라면, *SPAH*은 하위 level q보다 상위 level p인 *HiTi* 그래프 상에서 보다 나은 성능을 보일 것이다. 만약 그렇지 않다면, 높은 level의 *HiTi* 그래프는 낮은 level의 *HiTi* 그래프에 비해 우월한 성능을 제공하지 못할 것이다. 표 4는 표 2와 3에서 사용된 level 1, 2, 3의 *HiTi* 그래프에 대응하는 Y^1, Y^2, Y^3의 값을 보여준다.

표 4 *HiTi* 그래프 level의 Y^k에 미치는 영향

	Y^1	Y^2	Y^3
$10\le|N_i^1|\le20$	99000	98737	98487
$4\le|N_i^1|\le8$	82272	83364	83332

표 4는 각기 다른 level *HiTi* 그래프들 상에서 *SPAH*의 성능에 관하여 본 논문에서 추측했던 것을 입증한다. 표 2와 3에서 나타난 흥미 있는 사실은 탐색 공간의 크기(즉, H_n)가 level 1 *HiTi* 그래프보다 상위 *level HiTi* 그래프에서는 두드러진 변화가 없다는 것이다. 이것은 Y^i와 Y^{i+1}사이의 차이가 i가 증가함에 따라 크게 변화하지 않는 성질을 가진 격자그래프로 실험을 했기 때문이다. 이러한 현상은 대부분의 디지털 로드맵에서 나타난다고 여겨진다. 결과적으로, 상위 *level HiTi* 그래프가 반드시 최단 경로 계산 시간을 단축하지는 않는 것을 알 수 있다. 위의 실험으로부터, Y^i와 Y^{i+1}간의 차이가 작은 디지털 로드맵의 응용분야에서는 level i *HiTi* 그래프가 충분하다고 결론 내릴 수 있다.

5.4 메모리 요구

A* 최단 경로 알고리즘은 데이타베이스가 메인 메모리에 로드될 수 있을 때 *breadth-first search single pair* 최단 경로 알고리즘보다 더 효율적이라고 알려져 왔다[2, 32]. 이것은 알고리즘 *SPAH*에 있어서도 해당되는 일이다. 즉, *SPAH*는 전체 그래프를 탐색하지 않고 작은 서브그래프들의 *within*과 *between* edge 집합들에 속한 edge들을 탐색하기 때문에, 매번의 평균 메인 메모리 요구 량은 작을 것이라고 기대되어 진다.

위의 주장에 대한 좀 더 자세한 분석을 위해서, [18]에서 제안된 단순화된 분석 모델을 사용한다. 이 모델에서, 격자 그래프 $G(V,E)$는 $|V|=v=m\times m$이고 $n_1=$

$f \times f$인 n_1개의 level 1 서브그래프들의 집합으로 배타적으로 분할된다. 경계 노드들의 총 개수는 $2m(f-1)$, $\approx 2\sqrt{vn_1}$이다. 평균적으로, level 1인 각각의 서브그래프는 v/n_1개의 노드들을 갖는다. Level 1 서브그래프의 각 측면에 많아야 $\sqrt{v/n_1}$개의 경계 노드가 있으므로, level 1서브그래프는 $4\sqrt{v/n_1}$개의 경계 노드까지 가질 수 있다. 이 분석에서, v와 n_1으로 표현된 다음 4개의 파라미터를 사용한다.

- n_1 : 배타적으로 격자 그래프 $G(V,E)$를 분할하는 level 1 서브그래프들의 개수
- $n_j = n_1/4^{j-1}$: level j인 서브그래프들의 개수
- $b_j = 4\times(2^{j-1}\sqrt{v/n_j})$: 각 level j 서브그래프에 대한 경계 노드들의 평균 개수
- $w_j = b_j\times(b_j-1)$: level j인 *within edge* 집합의 평균 크기

최악의 경우(즉, $LUB_{ST}(SG_i^1, SG_j^1)$가 k+1인 경우)에 *SPAH*가 메인 메모리로 로드하는 level k *HiTi* 그래프의 부분에 대한 크기를 MR^k라고 하자. 정리 4.4와 위의 단순화된 분석 모델에 근거하여, MR^k을 다음과 같이 공식화할 수 있다:

$$MR^k = n_k w_k/n_{k+1} + 2n_{k-1}w_{k-1}/n_k + 2n_{k-2}w_{k-2}/n_{k-1} + \cdots + 2n_1 w_1/n_2$$

아래 그림 9는 level 1, 2, 3인 *HiTi* 그래프가 사용될 때, *SPAH*의 메모리 요구량을 보여주고 있다. 여기서, level 1 서브그래프(즉, n_1)의 개수를 64로 고정한 후, G에 대한 노드들의 개수를 100×100에서 800×800까지 변화시키며 MR^1, MR^2, MR^3을 측정한다.

그림 9는 노드 개수가 증가함에 따라, *level* 3의 *HiTi* 그래프에 대한 메모리 요구량이 매우 급격하게 증가한다는 사실을 보여준다. 그러나, *level* 1인 *HiTi* 그래프에 대해서는 메모리 요구량이 매우 느리게 증가하고 있다. 예를 들어, 그래프 *G(V,E)*가 640,000개의 노드를 갖을 때 *level* 1의 *HiTi* 그래프에 대한 메모리 요구량은 10 Mbyte이며, 이것은 메인 메모리에 로드될 수 있는 충분히 작은 크기이다. *HiTi* 그래프가 필요로 하는 메모리 크기는 그림 9에서 보여준 것처럼 많지 않지만, 어떤 경우에서는 메인 메모리가 충분하지 않을 수 있다. 그런 경우에, *SPAH*는 *I/O* 비용을 가능한 한 많이 줄임으로써 disk-based HiTi 그래프상의 단일쌍 최단경로 비용을 계산한다. 이를 위해, 디지털 로드맵과 같은 네트워크 데이타 상의 경로 계산 질의어를 위한 효율적인 공간 접근 방법(spatial access methods)이 필요하다. 네트워크 데이타는 지리적인 근접(geographic proximity)보다는 노드들의 연결(connectivity of nodes)에 기초하여 효율적으로 클러스터 되어야 한다.

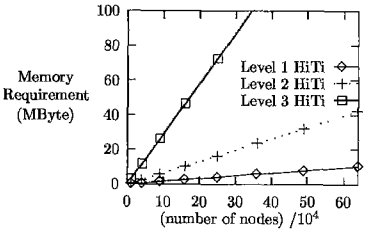


그림 9 Level 1, 2, 3 *HiTi* 그래프의 메모리 요구량

연결에 기반을 둔(connectivity-based) 접근 방법에 관한 많은 연구가 있어왔다[7, 25, 27, 28, 29]. 그러나, 여기서 제안된 방법들은 로드맵을 포함하는 일반적인 네트워크 상에서 경로 평가와 같은 집합(aggregate) 질의어 처리에는 적합하지 않다. *Shekhar*와 *Liu*는 디지털 로드맵과 같은 일반적인 네트워크 상에서 집합(aggregate) 질의어를 처리하기 위한 효율적인 접근 방법인 *CCAM* (*Connectivity Clustered Access Method*)을 제안하였다[30]. Minneapolis 디지털 로드맵 상에서 경로 계산에 대한 그들의 실험은 *CCAM* 방법의 효율성을 보여주고 있다. 이러한 것을 고려하여 볼 때 *CCAM* 방법은 단일쌍 최단경로(Single Pair Shortest Path)계산에 드는 *I/O* 비용을 줄이기 위해 디스크에 *HiTi* 그래프를 효율적으로 저장하는데 적합하다고 생각한다.

6. HiTi 그래프 updating

$G(V,E)$의 edge들의 비용이 갱신될 때마다, *HiTi* 그래프를 수정하는 것이 필요하다. 예를 들어, 디지털 로드맵은 교통 상태가 변할 때마다 갱신될 필요가 있다. 그러므로, *HiTi* 그래프를 위한 효율적인 갱신 알고리즘은 항법 시스템과 같은 응용 분야에 필수적이다. 그림 10은 level k인 *HiTi* 그래프에 대한 효율적인 갱신 알고리즘을 보여주고 있다.

그림 10에서 소개한 알고리즘의 갱신 비용은 level k +1 서브그래프 트리 ST에 속한 서브그래프들의 개수에 비례해 선형적이다. 즉, 하나의 edge 비용 변화의 영향은 level 1 서브그래프의 조상(ancestor) 서브그래프들의 *within edge* 집합들로 제한되고, 그 밖의 다른 모든 서브그래프들의 *within edge* 집합들에는 영향을 미치지 않는다. 이것은 $1 \le l \le k$ 이고 $1 \le i \le n_l$인 각각의 SG_i^l의 모든 경계 노드들 사이에서 최단 경로가 단지 SG_i^l 내에서

/* Assume we have a level k *HiTi* graph defined on a level $k+1$ subgraph tree ST; */

1. Let U be a set of edges of $G(V,E)$ whose costs are updated.
2. U can be represented as $U_L^0 \cup U_B^1 \cup U_B^2 \cup \cdots \cup U_B^k$ where
   U_L^0 contains the updated edges in $\cup_{i=1}^{n_1} SG_i^1$,
   U_B^1 contains the updated edges in $\cup_{i=1}^{n_1} B_i^1$,
   U_B^2 contains the updated edges in $\cup_{i=1}^{n_2} B_i^2, \ldots$,
   U_B^k contains the updated edges in $\cup_{i=1}^{n_k} B_i^k$.
3. Identify the level 1 subgraphs whose edges are in U_L^0 and let $l = 1$.
4. Recompute the level l *within edge sets* of those identified level l subgraphs.
5. Let X contain the level l subgraphs whose *within edge sets* are updated by the recomputation.
6. Let Y contain the level l subgraphs whose *between edge sets* include the edges in U_B^l.
7. For each level $l+1$ subgraph SG_i^{l+1} in $S_A^{l+1}(X \cup Y)$, recompute the level $l+1$ *within edge set* $S_W(SG_i^{l+1})$ by using $S_{BW}(S_C(SG_i^{l+1}))$.
8. Let $l = l+1$. If $l < k$ then goto step 4. Otherwise stop.
   /* When $l = k$, the recomputation of level $k+1$ *within edge sets* is not necessary since the *within* and *between edge sets* of a level $k+1$ subgraph (i.e., the root node of ST) are always empty. */

그림 10 Level k HiTi 그래프를 위한 갱신 알고리즘

만 지역적(locally)으로 미리 계산되기 때문에 가능하다. 그러므로, edge 비용이 변할 때, 갱신 비용 *UC*는 그림 10의 step 7에서 보이는 것과 같이 재계산이 필요한 *within edge* 집합들의 크기이다. *UC*에 대한 좀 더 자세한 분석을 위해서, 5.4절에서 언급한 단순화된 분석 모델을 사용한다. Level k *HiTi* 그래프에서 *UC*는 $\sum_{i=1}^{k} W_i$로 표현된다. 갱신된 level 1 서브그래프들의 개수가 변화할 때, *UC*가 어떻게 되는지 보기 위해서, 64개의 level 1, 16개의 level 2, 4개의 level 3인 서브그래프들이 10000개의 노드들을 가진 격자 그래프 G상에 정의되었다고 가정하자. 다음 그림 11은 level 1, 2, 3인 *HiTi* 그래프들에 대한 갱신 비용을 보여준다.

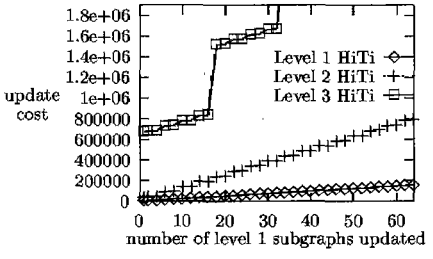


그림 11 Level 1, 2, 3 HiTi 그래프들의 갱신 비용

이 그림에서 *level* 1과 2인 *HiTi* 그래프의 갱신 비용은 변화된 level 1인 서브그래프들의 개수 증가와 함께 거의 선형적으로 증가하고 있음을 알 수 있다. 그러나, level 3 *HiTi* 그래프의 갱신 비용은 1에서 16개의 서브그래프까지는 선형적으로 증가하다가, 17개의 서브그래프에서 훨씬 높은 cost로 급격히 증가한다. 이것은 갱신된 하위 level의 *within edge* 집합들이 재계산이 필요한 동일한 다음 상위 *level*의 *within edge* 집합에 속해 있기 때문이다. 결과적으로, 갱신 비용의 급격한 증가는 갱신된 하위 level의 *within edge* 집합들이 재계산이 필요한 다른 다음 상위 level *within edge* 집합들을 가질 때 일어난다. 위와 같은 경향은 그림에서 *level 2 HiTi* 그래프에서도 보여지지만 *level 3 HiTi* 그래프에서의 갱신 비용의 급격한 증가와 비교해 볼 때 상대적으로 작다. 이러한 분석으로 볼 때 *HiTi* 그래프는 상위 *level within edge* 집합의 *edge*들이 낮은 *level*의 *edge*들 갱신에 덜 민감하게끔 하는 효율적인 구조를 제공한다는 것을 알 수 있다.

7. 다른 유사한 연구와의 비교

이 절에서는 본 논문의 *HiTi* 그래프 모델과 매우 관련 있는 다른 두 가지 연구 결과들을 비교 분석한다. HEPV[17, 18]와 Hub Indexing[19] 방법이 바로 그것이다. HEPV 방법은 먼저 각 서브그래프에 속한 모든 노드 쌍에 대하여 그 서브그래프 안에서의 최단 경로[2]를 미리 계산하여 각 서브그래프에 대한 FPV(Flat Path View)를 생성한다. 모든 FPV들에 있는 경로 정보를 사용하여, 모든 서브그래프에 대한 하나의 *supergraph*를 생성한다. 이 supergraph는 각 서브그래프의 모든 경계 노드들간의 경로 정보를 가지고 있다. HEPV는 supergraph에 대하여 하나의 FPV를 생성한다. 이 FPV는 모든 서브그래프들의 모든 경계노드들 간의 최단 경로[3]를 가지고 있다. 이러한 FPV를 여기서는 SFPV (Flat Path View for Supergraph)라고 명명한다.

Hub Indexing 방법은 모든 서브그래프들에 대한 하나의 distance table을 생성하는데 이 테이블은 각 서브그래프의 경계 노드들과 그 서브그래프의 내부 노드들(즉, 각 서브그래프의 비 경계 노드들) 사이의 미리 계산된 최단 경로[4]를 저장하고 있다. 여기서 각 서브그래프에 대응하는 distance table의 부분을 PDT(Partial Distance Table)로 표시한다. 또한, Hub Indexing 방법은 모든 서브그래프들의 모든 경계 노드 쌍에 대한 최단 경로[5]를 미리 계산하여 저장한 하나의 Hub Set을 생성한다.

HiTi 그래프 방법과 같이, 이 두 가지 방법도 edge cost 갱신의 영향을 그 edge가 속한 서브그래프에 해당

2) 여기서 계산된 최단 경로는 반드시 전체 그래프 G에 대한 최적 최단 경로는 아니다.
3) 여기서의 최단경로는 전체 그래프 G에 대한 최단 경로이다.
4) 여기서 계산된 최단경로는 반드시 전체 그래프 G에 대하여 최적이 아니다.
5) 여기서의 최단경로는 전체 그래프 G에 대한 최단 경로이다.

하는 FPV와 PDT으로 제한하며 다른 모든 서브그래프들에 해당하는 FPV와 PDT들은 영향을 받지 않는다. 그러나, SFPV와 Hub Set에서 저장된 모든 최단 경로는 그들이 전체적인 최적성을 고려하기 때문에 갱신에 의해 영향을 받게 된다. 이 두 가지 방법의 특성을 고려해 볼 때, 이 방법들은 큰 디지털 로드맵 상에서는 더 심각한 저장 오버헤드와 갱신 유지 문제를 갖는다고 생각한다.

HEPV와 Hub Indexing 방법과는 달리, HiTi 그래프 모델은 미리 계산된 많은 경로 정보를 유지하는데 대한 과도한 저장 오버헤드를 갖지 않는다. 그 이유는 HiTi 그래프 모델에서는 오직 각 서브그래프의 경계 노드들 간의 최단 경로만을 미리 계산하기 때문이다. 또한, HiTi 그래프 모델에서의 저장 오버헤드 감소로 인하여 HiTi 그래프 모델은 HEPV와 Hub Indexing 방법에 비해서 훨씬 적은 갱신 비용을 제공한다. 왜냐하면, HiTi 그래프를 갱신하기 위해 필요한 최단 경로 재 계산 량은 HEPV와 Hub Indexing 방법에 비해서 훨씬 적기 때문이다.

더 자세한 분석을 위해, 5.4절에서 언급한 단순화된 분석 모델을 다시 사용한다. Hub Indexing 방법이 오직 level 1 서브그래프들만을 다루기 때문에, 위 세 가지 방법에 대한 공정한 비교를 위해서 오직 level 1 서브그래프들만을 고려한다. 그러므로, 여기서부터는 달리 언급하지 않으면, level 1 서브그래프는 그냥 서브그래프와 똑 같은 것으로 취급한다. 아래의 5개의 파라미터들은 v와 n_1에 의하여 표시된다.

- v: 격자 그래프 $G(V,E)$에서 노드들의 총 개수
- n_1 : 그래프 $G(V,E)$를 배타적으로 분할하는 level 1 서브그래프들의 총 개수
- $t=2\sqrt{vn_1}$: G에 속한 모든 서브그래프들에 대한 경계 노드들의 총 개수
- $b_1=4\sqrt{v/n_1}$: 각 level 1 서브그래프에 대한 경계 노드들의 평균 개수
- $i_1=v/n_1-b_1$: 각 level 1 서브그래프에 대한 내부 노드들의 평균 개수

위의 5개의 파라미터를 이용함으로써, 갱신 비용, 저장 오버헤드, 단일쌍 최단경로(SPSP: Single Pair Shortest Path) 계산 시간의 3가지 관점에서 각각의 방법을 분석해 본다. 갱신 비용은 그래프 G의 edge 비용 갱신 때문에 재 계산이 필요한 미리 계산된 최단 경로들의 개수로서 나타낸다. 저장 오버헤드는 미리 계산된 최단 경로들의 총 개수로 나타낸다. SPSP 계산 시간은 각 방법의 계산 복잡도(Computational Complexity)로 나타낸다.

먼저, edge 비용이 변할 때의 갱신 비용을 논의한다. HEPV 방법에서 갱신 비용은 FPV갱신 비용(즉, $(b_1+i_1)(b_1+i_1-1)$)과 supergraph 갱신 비용(즉, $t(t-1)$)으로 이루어진다. Hub Indexing 방법에서 갱신 비용은 Hub Set를 갱신 비용(즉, $t(t-1)$)와 PDT 갱신 비용(즉, b_1i_1)으로 이루어진다. HiTi 그래프에서 갱신비용은 하나의 within edge 집합의 갱신 비용(즉, $b_1(b_1-1)$) 이다.

다음으로 각 방법에 대한 저장 오버헤드를 논의한다. HEPV에서, 저장 오버헤드는 모든 FPV들의 크기(즉, $(b_1+i_1)(b_1+i_1-1)n_1$)와 SFPV의 크기(즉, $t(t-1)$)로 이루어진다. Hub Indexing에서, 저장 오버헤드는 distance table의 크기(즉, $b_1i_1k_1$)와 Hub Set의 크기(즉, $t(t-1)$)로 이루어진다. HiTi 그래프 방법에서, 저장 오버헤드는 모든 within edge 집합들의 크기(즉, $b_1(b_1-1)n_1$)이다.

마지막으로, 각 방법에 대해 SPSP 계산 시간을 논의한다. HEPV와 Hub Indexing 방법에서 제시한 자세한 SPSP 계산 알고리즘들은[17, 18, 19]에 기술되어 있다. HEPV 방법에서 제시한 SPSP 계산 알고리즘의 시간 복잡도(time complexity)는 SFPV중의 필요한 부분에 대한 탐색(즉, b_1^2)에 의해 결정된다. Hub Indexing 방법에서 제시한 SPSP 계산 알고리즘의 시간 복잡도는 주로 PDT 탐색 비용(즉, 서브 그래프 탐색 비용, i_1^2)과 Hub Set의 일부분에 대한 탐색 비용(즉, b_1^2)에 의해 결정된다. HiTi 그래프 방법의 SPSP 계산 알고리즘의 시간 복잡도는 level 1인 2개의 서브그래프들에 대한 탐색 비용(즉, $2i_1^2$)과 level 1인 HiTi 그래프에 대한 탐색 비용(즉, t^2)으로 이루어진다. 다음 표 5는 지금까지 언급한 세 가지 방법들에 대한 성능 비교를 요약하여 보여주고 있다.

표 5에서 주어진 결과를 이용하여, HEPV, Hub Indexing, HiTi 그래프 방법들의 성능을 분석한다. 그림 12에서, 먼저 10,000개의 노드들을 가진 격자 그래프 상에 정의된 서브그래프들의 개수(즉, n_1)를 변화시키면서, 각 방법에 대한 갱신 비용, 저장 오버헤드, SPSP 계산 시간을 측정한다. 그림 12 a)는 HiTi 그래프의 갱신 비용이 HEPV와 Hub Indexing 방법들보다 훨씬 적다라는 사실을 보여준다. 그 이유는 HiTi 그래프 방법에서 edge 비용 변화의 영향은 오직 연결된 대응되는 서브그

표 5 HEPV, Hub Indexing, HiTi 그래프 방법들의 성능 비교

Methods	Update cost	Storage overhead	SPSP Computation time
HEPV	$(b_1+i_1)(b_1+i_1-1)+t(t-1)$ $=O(v^2/n_1^2+vn_1)$	$(b_1+i_1)(b_1+i_1-1)n_1+t(t-1)$ $=O(v^2/n_1+vn_1)$	b_1^2 $=O(v/n_1)$
Hub Indexing	$b_1i_1+t(t-1)=O(vn_1)$	$b_1i_1n_1+t(t-1)=O(vn_1)$	$i_1^2+b_1^2=O(v^2/n_1^2)$
HiTi Graph	$b_1(b_1-1)=O(v/n_1)$	$b_1(b_1-1)n_1=O(v)$	$2(b_1+i_1)^2+t^2=O(v^2/n_1^2+vn_1)$

래프의 *within edge* 집합으로만 제한되기 때문이다. 그러나, HEPV와 Hub Indexing 방법에서 갱신의 영향은 대응되는 서브그래프에 대한 FPV와 PDT 뿐만 아니라, SFPV와 Hub Set에도 미친다. Hub Indexing 방법에서의 갱신 비용은 HEPV 보다 더 작은데 이는 PDT의 크기(b_1i_1)가 FPV의 크기($(b_1+i_1)(b_1+i_1-1)$) 보다 훨씬 더 작기 때문이다.

HiTi 그래프 방법에서는 서브그래프들의 개수가 증가함에 따라 갱신 비용은 감소함을 알 수 있다[6]. 그러나, HEPV와 Hub Indexing 방법의 갱신 비용은 각각 25와 16개의 서브그래프 지점까지 감소하고 있으며 그 이후부터는 그 둘의 차이를 좁히면서 증가하기 시작한다. 이것은 다음 두 가지 이유 때문에 일어난다:

- HEPV와 Hun Indexing 방법에서 서브그래프들의 개수의 증가는 각각 25와 16개의 서브그래프까지 갱신 비용 감소에 긍정적인 영향을 미친다.
- 그 후, supergraph의 SFPV 크기(즉, $t(t-1)$)와 Hub Set 크기(즉, $t(t-1)$)가 갱신 비용에 증가에 대한 결정적인 요인으로 작용한다.

이 사실로부터, HEPV와 Hub Indexing 방법의 갱신 비용(update cost)은 많은 서브그래프들을 가지는 격자 형태의 그래프에 대해서 거의 같을 것이라고 생각된다.

그림 12 b)는 HiTi 그래프 방법의 저장 오버헤드가 HEPV와 Hub Indexing 방법과는 달리, 서브그래프들의 개수가 변해도 큰 영향을 받지 않는다는 것을 보여준다. HEPV와 Hub Indexing 방법의 저장 오버헤드의 경향은 갱신 비용에서와 같이 비슷한 특성을 갖는다. 이 사실에 대한 정당성은 위에서 주어졌던 두 가지 이유로 설명된다. 단일쌍 최단 경로 계산 비용(SPSP computation cost)에 대해서, 그림 12 c)는 HEPV와 Hub Indexing 방법이 HiTi 그래프 방법보다 훨씬 적은 계산 시간을 제공한다는 사실을 보여준다. 이것은 표 5로부터 예상할 수 있다.

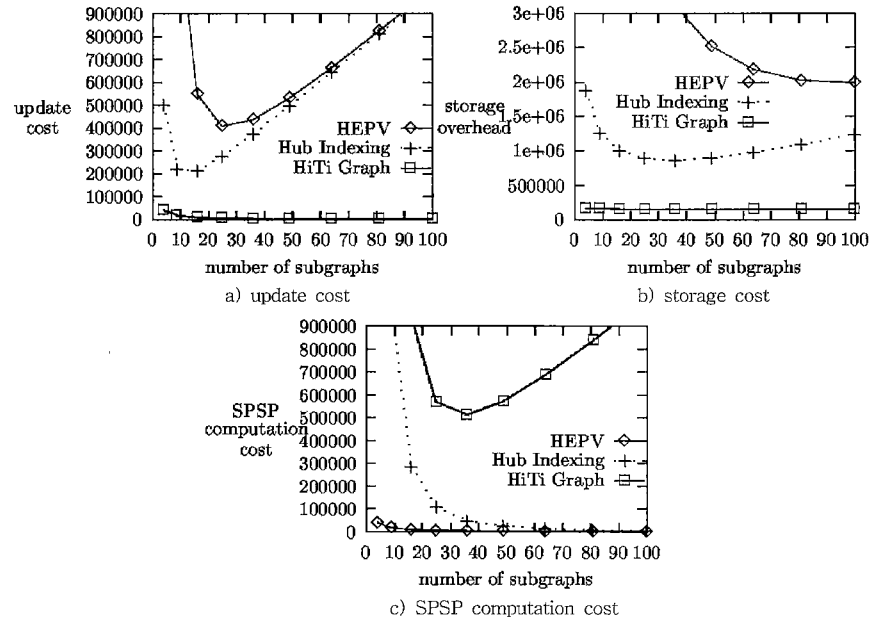
6) 그림 12a)에서 HiTi 그래프 방법의 갱신 비용의 변화는 39800, 17643, 9900, 6320, 4365, 3207, 2446, 1924, 1550 이다.

또한 그림 12 c)는 10000개의 노드들을 가진 격자 그래프가 주어졌을 때, HiTi 그래프 방법에서 최적인 서브그래프의 개수(즉, 36개의 서브 그래프)를 보여준다. 일반적으로, 고정된 크기의 그래프가 주어졌을 때, 갱신 비용, 저장 오버헤드, 그리고 SPSP 계산 비용들을 고려해서 위의 각 세 가지 방법에 대한 서브그래프의 최적인 개수를 결정할 수 있다. 예를 들어, Hub Indexing 방법에서 서브그래프의 최적인 개수는 위의 세 가지 비용 요소들 중 어느 비용 요소가 가장 높은 우선 순위로 주어졌느냐에 따라서, 16, 25 또는, 36이다.

다음 그림 13에서는 격자 그래프의 노드 개수를 100×100에서 150×150, 200×200, 250×250, ... , 400×400까지 변화시키면서, 위의 세 가지 방법에 대한 갱신 비용(Update cost), 저장 오버헤드(Storage overhead), 단일쌍 최단 경로 계산 비용(SPSP Computation cost)을 측정한다. 여기서, 서브그래프의 개수는 36으로 고정한다. 그림 13 a)와 b)는 HiTi 그래프 방법이 갱신 비용과 저장 오버헤드의 관점에서는 HEPV와 Hub Indexing 방법보다 더 우수하다는 것을 보여준다. 그러나, SPSP 계산 비용의 측면에서는 그림 13 c)에서 나타난 것과 같이 HEPV 방법이 나머지 두 방법보다 우수하다. Hub Indexing 방법의 성능은 항상 HiTi 그래프와 HEPV 방법의 중간이다. 이러한 성능 분석 결과는 각 방법에서 저장되는 미리 계산된 최단 경로의 양에 의해 설명될 수 있다. 즉, 미리 계산된 최단 경로의 양은 HiTi 그래프, Hub Indexing, 그리고 HEPV 방법 순서로 증가한다.

8. 결 론

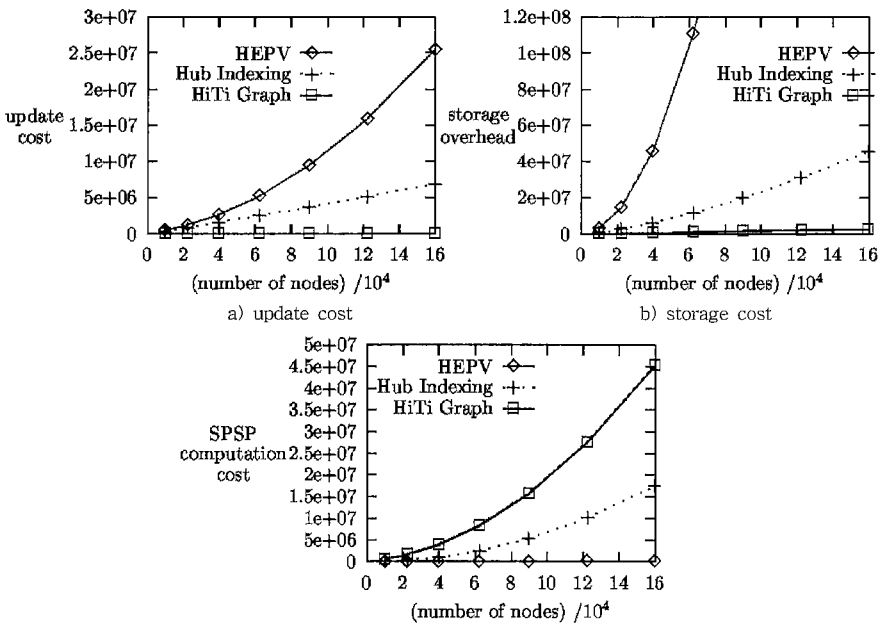
본 논문에서는 매우 큰 디지털 로드맵을 위한 *HiTi* 그래프 모델이라고 하는 새로운 그래프 모델을 개발하였다. *HiTi* 그래프는 디지털 로드맵 데이타를 계층적으로 구조화하기 위한 강력한 형식적인 프레임워크를 제공한다. 먼저, 본 논문에서는 *HiTi* 그래프 모델에 기초하여 제안한 단일쌍 최단 경로 알고리즘인 *SPAH*가 매우 큰 디지털 로드맵상에서 최소 비용 경로를 계산하기



a) update cost b) storage cost

c) SPSP computation cost

그림 12 10000개의 노드를 가진 격자 그래프 상에서 서브그래프들의 개수 변화에 따른 비용 비교



a) update cost b) storage cost

c) SPSP computation cost

그림 13 36개의 서브그래프들을 갖는 격자 그래프 크기의 변화에 따른 비용 비교

위하여 필요한 탐색 공간을 상당히 줄인다는 사실을 보여주었다. 그리고, *SPAH*에 의해 계산된 최단 경로가 최적이라는 사실을 형식적으로 증명하였다. 또한, *HiTi* 그래프의 메모리 요구량과 동적인 교통 상황을 반영하기 위해 많은 항법 시스템에서 필수적인 효율적인 *HiTi* 그래프 갱신 알고리즘도 연구하였다. 마지막으로, *HiTi* 그래프 방법을 갱신 비용(Update cost), 저장 오버헤드(Storage overhead), 단일쌍 최단 경로 비용(SPSP computation cost)의 세 가지 관점에서 HEPV와 Hub Indexing과 같은 다른 유사한 연구들과 이론적 뿐만 아니라 실험적으로 상세하게 비교 분석하였다.

부 록

정리 4.1 SG_i^l을 $k\geq 1$이고 $1\leq l\leq k+1$대해서 level $k+1$ 서브그래프 트리 ST에서의 level l서브그래프 노드라고 하자. 또, $S_C(SG_i^l)=\{SG_1^{l-1}, SG_2^{l-1}, \ldots, SG_p^{l-1}\}$라고 하자. $x, y\in S_N(S_C(SG_i^l))$인 어떤 경계 노드 쌍 x와 y대해서, $SPC_{SG_i^l}(x,y)=SPC_{S_{BW}(S_C(SG_i^l))}(x,y)$가 성립한다.

증명: $P_{SG_i^l}(x,y)$가 경계 노드 x에서 y까지의 경로를 나타낸다고 하자. 이때, 어떠한 경로 $P_{SG_i^l}(x,y)$도 경로상의 양쪽 끝 노드 x, y를 포함하여 이 경로상에 있는 모든 경계 노드들의 순서 $x, z_1, z_2, \ldots, z_m, y$ 와 같이 나타낼 수 있다. 그러므로, 이 경로의 비용은 $PC_{SG_i^l}(x,y)=PC_{SG_i^l}(x,z_1)+PC_{SG_i^l}(z_1,z_2)+\cdots PC_{SG_i^l}(z_m,y)$와 같이 나타낼 수 있다. 최적성의 원리(principle of optimality)에 의해서, 최단 경로 비용 $SPC_{SG_i^l}(x,y)$는 다음과 같이 나타낼 수 있다 :

$$SPC_{SG_i^l}(x,y)=SPC_{SG_i^l}(x,z_1)+SPC_{SG_i^l}(z_1,z_2)+\cdots SPC_{SG_i^l}(z_m,y)$$

위의 경계 노드들의 순서에서, 모든 두 개의 연속된 경계 노드들은 $S_C(SG_i^l)$에 속한 어떤 하나의 서브그래프에 들어가는 경계노드와 그 서브그래프를 벗어나는 경계 노드들에 대응한다. 그러므로, 경계 노드들의 순서 $x, z_1, z_2, \ldots, z_m, y$ 에 속한 모든 연속된 노드의 쌍은 $S_C(SG_i^l)$에 포함된 모든 *level* l-1 서브그래프들의 *level l-1 between*과 *within* edge 동일 집합에 속한다. 다시 말하면, $(x,z_1)\in S_{BW}(SG_{j_1}^{l-1})$; $(z_1,z_2)\in S_{BW}(SG_{j_2}^{l-1})$; $\cdots$; $(z_m,y)\in S_{BW}(SG_{j_m}^{l-1})$; 이고, 여기서 $1\leq j_1, j_2, \cdots, j_m\leq p$ 이다. 최단 경로 비용 $SP_{SG_i^l}(x,z_1)$이 $S_{BW}(SG_{j_1}^{l-1})$로부터 얻어질 수 있듯이, $SPC_{SG_i^l}(x,z_1)=SPC_{S_{BW}(SG_{j_1}^{l-1})}$을 얻을 수 있다. 이와 유사하게, $SPC_{SG_i^l}(z_1,z_2)=SPC_{S_{BW}(SG_{j_2}^{l-1})}(z_1,z_2)$, 등등을 얻을 수 있다. 그러므로, 최단 경로 비용 $SPC_{SG_i^l}(x,y)$은 다음과 같이 나타낼 수 있다:

$$SPC_{SG_i^l}(x,y)=SPC_{S_{BW}(SG_{j_1}^{l-1})}(x,z_1)+SPC_{S_{BW}(SG_{j_2}^{l-1})}(z_1,z_2)+\cdots+SPC_{S_{BW}(SG_{j_m}^{l-1})}(z_m,y)$$

이로부터, 최단 경로 비용 $SPC_{SG_i^l}(x,y)$은 $S_{BW}(S_C(SG_i^l))$에 포함된 edge들만을 이용해서 계산할 수 있다. 그러므로, 정리 $SPC_{SG_i^l}(x,y)=SPC_{S_{BW}(S_C(SG_i^l))}(x,y)$은 증명되었다. □

정리 4.2 SG_i^l을 $k\geq 1$이고 $1\leq l\leq k+1$대해서 level $k+1$ 서브그래프 트리 ST에서의 level l 서브그래프 노드라고 하자. 또, $S_C(SG_i^l)=\{SG_1^{l-1}, SG_2^{l-1}, \ldots, SG_p^{l-1}\}$라고 하자. $1\leq q\leq p$인 모든 노드 쌍 $x, y\in SG_q^{l-1}$에 대해서, 다음이 성립한다:

$$SPC_{SG_i^l}(x,y)=SPC_{SG_q^{l-1}\cup S_{BW}(S_C(SG_i^l))}(x,y).$$

증명: x에서 y로의 모든 경로는 두 가지 경우로 분류될 수 있다:

경우 1: 경로는 SG_q^{l-1}에 속한 edge들로만 이루어진다.

경우 2: 경로는 SG_q^{l-1}에 속한 edge뿐만 아니라 $S_C(SG_i^l)$에 포함된 다른 level $l-1$인 서브그래프들에 속한 edge들로 함께 이루어진다.

경우 1에 속하는 x에서 y로의 경로 비용은 $PC_{SG_q^{l-1}}(x,y)$로 표현된다. 만약 x에서 y로의 모든 경로가 이러한 종류이라면, 이때 x에서 y로의 최단 경로 또한 SG_q^{l-1}에서 정의되어 진다고 생각할 수 있다. 즉, $SPC_{SG_i^l}(x,y)=SPC_{SG_q^{l-1}}(x,y)$이다.

경우 2에 속하는 모든 경로는 노드 순서, $x, z_1, z_2, \ldots, z_m, y$에 의해 표현될 수 있는 SG_i^l의 edge들로 이루어진다. 이 경로가 $S_C(SG_i^l)$에 포함된 다른 level $l-1$ 서브그래프들의 edge들을 포함하므로, 이 경로는 서브그래프 SG_q^{l-1}의 노드를 떠나지만 결국 다시 서브그래프 SG_q^{l-1}의 노드로 돌아올 것이다. 그러므로, 이 경로는, $a, b\in S_N(SG_q^{l-1})$ 이고 $a\neq b$을 만족하는 적어도 두 개의 경계 노드 a와 b를 포함한다. 이 경로 상에서, a를 처음 노드로, b를 마지막 노드로 설정해보자. 이 특정 경로의 경로 비용은 다음과 같다:

$$PC_{SG_i^l}(x,y)=PC_{SG_i^l}(x,a)+PC_{SG_i^l}(a,b)+PC_{SG_i^l}(b,y)$$

최단 경로의 노드 순서가 $x, \cdots, a, \cdots, b, \cdots, y$라고 가정한다. 최적성의 원리에 의해서, $SPC_{SG_i^l}(x,y)=SPC_{SG_i^l}(x,a)+SPC_{SG_i^l}(a,b)+SPC_{SG_i^l}(b,y)$을 얻을 수 있다. 최단 경로 $SP_{SG_i^l}(x,a)$가 오직 SG_q^{l-1}의 edge들로 이루어져 있으므로, 이 경로는 경우 1에 해당된다. 그러므로,

$SPC_{SG_i^l}(x,a)=SPC_{SG_u^{l-1}}(x,a)$이다. 유사하게, $SPC_{SG_i^l}(b,y)=SPC_{SG_v^{l-1}}(b,y)$ 이다. $SPC_{SG_i^l}(a,b)$대해서, 정리 4.1에 의해 $SPC_{SG_i^l}(a,b)=SPC_{S_{BW}(S_C(SG_i^l))}(a,b)$ 이다. 그러므로, SG_i^l에 대해서, x에서 y로의 최단 경로 비용은 다음과 같이 주어진다:

$$SPC_{SG_i^l}(x,y)=SPC_{SG_u^{l-1}}(x,a)+$$
$$SPC_{S_{BW}(S_C(SG_i^l))}(a,b)+SPC_{SG_u^{l-1}}(b,y)$$

이 식으로부터, 최단 경로 $SP_{SG_i^l}(x,y)$은 다음 경로 집합에 속한다고 할 수 있다:

$$\{SP_{SG_u^{l-1}}(x,a)+SP_{S_{BW}(S_C(SG_i^l))}(a,b)+SP_{SG_v^{l-1}}(b,y)$$
$$\mid a,b\in S_N(SG_q^{l-1})\wedge a\neq b\}$$

그러므로 최단 경로 비용 $SPC_{SG_i^l}(x,y)$은 다음과 같이 나타낼 수 있다:

$$SPC_{SG_i^l}(x,y)=\min(\{SPC_{SG_q^{l-1}}(x,a)$$
$$+SPC_{S_{BW}(S_C(SG_i^l))}(a,b)+SPC_{SG_v^{l-1}}(b,y)$$
$$\mid a,b\in S_N(SG_q^{l-1})\wedge a\neq b\})$$

경우 1과 2를 결합하여, $SPC_{SG_i^l}(x,y)=\min(SPC_{SG_q^{l-1}}$ $\min(\{SPC_{SG_q^{l-1}}(x,a)+SPC_{S_{BW}(S_C(SG_i^l))}(a,b)+SPC_{SG_q^{l-1}}(b,y)$ $\mid a,b\in S_N(SG_q^{l-1})\wedge a\neq b\})$을 얻을 수 있는데 이는 바로 $SPC_{SG_i^l}(x,y)=SPC_{SG_q^{l-1}\cup S_{BW}(S_C(SG_i^l))}(x,y)$ 임을 증명하고 있다. □

정리 4.3 SG_i^l을 $k\geq 1$이고 $1\leq l\leq k+1$대해서 level $k+1$ 서브그래프 트리 ST에서의 level l인 서브그래프 노드라고 하자. 또, $S_C(SG_i^l)=\{SG_1^{l-1},SG_2^{l-1},\ldots,SG_p^{l-1}\}$라고 하자. $1\leq u,\ v\leq p$이고 $u\neq v$인 모든 노드 쌍 $x\in SG_u^{l-1}$, $y\in SG_v^{l-1}$에 대해서, 다음이 성립한다:

$$SPC_{SG_i^l}(x,y)=SPC_{SG_u^{l-1}\cup SG_v^{l-1}\cup S_{BW}(S_C(SG_i^l))}(x,y).$$

증명: SG_i^l에 속한 x에서 y로의 모든 경로는 노드 순서 $x,z_1,z_2,\ldots,z_m,y$와 같이 나타낼 수 있다. 이때 $a\in S_N(SG_u^{l-1})$이고 $b\in S_N(SG_v^{l-1})$인 두 경계 노드 a와 b는 이 경로 상에 반드시 존재한다. 왜냐하면, 경로는 결국 SG_u^{l-1}을 떠나 SG_v^{l-1}에 들어가기 때문이다. 이 경로에서 이와 같은 처음 경계 노드를 a로 마지막 경계 노드를 b로 하자. SG_i^l의 경로 비용은 다음과 같이 나타낼 수 있다:

$$PC_{SG_i^l}(x,y)=PC_{SG_i^l}(x,a)+PC_{SG_i^l}(a,b)+PC_{SG_i^l}(b,y)$$

x에서 y로의 최단 경로에 대한 노드 순서가 $x,\cdots,a,\cdots,b,\cdots,y$라고 가정하자. 최적성의 원리에 의해서, $SPC_{SG_i^l}(x,y)=SPC_{SG_i^l}(x,a)+SPC_{SG_i^l}(a,b)+SPC_{SG_i^l}(b,y)$ 를 얻을 수 있다. 최단 경로 $SP_{SG_i^l}(x,a)$는 SG_u^{l-1}에 속한 edge들로만 이루어지고, 최단 경로 $SP_{SG_i^l}(b,y)$는 SG_v^{l-1}에 속한 edge들로 만 이루어지므로, $SPC_{SG_i^l}(x,a)=SPC_{SG_u^{l-1}}(x,a)$ 이고 $SPC_{SG_i^l}(b,y)=SPC_{SG_v^{l-1}}(b,y)$ 이다. $SPC_{SG_i^l}(a,b)$는 정리 4.1에 의해 $SPC_{SG_i^l}(a,b)=SPC_{S_{BW}(S_C(SG_i^l))}(a,b)$ 임을 알 수 있다. 그러므로, 최단 경로 비용 $SPC_{SG_i^l}(x,y)$는 다음과 같이 나타낼 수 있다:

$$SPC_{SG_i^l}(x,y)=SPC_{SG_u^{l-1}}(x,a)+$$
$$SPC_{S_{BW}(S_C(SG_i^l))}(a,b)+SPC_{SG_i^{l-1}}(b,y)$$

이 식으로부터, 최단 경로 $SP_{SG_i^l}(x,y)$은 다음 경로 집합에 속한다고 할 수 있다:

$$\{SP_{SG_u^{l-1}}(x,a)+SP_{S_{BW}(S_C(SG_i^l))}(a,b)+SP_{SG_i^{l-1}}(b,y)$$
$$\mid a\in S_N(SG_u^{l-1})\wedge b\in S_N(SG_v^{l-1})\}$$

따라서, 최단 경로 비용 $SPC_{SG_i^l}(x,y)$은 다음과 같이 주어진다:

$$SPC_{SG_i^l}(x,y)=\min(\{SPC_{SG_u^{l-1}}(x,a)$$
$$+SPC_{S_{BW}(S_C(SG_i^l))}(a,b)+SPC_{SG_i^{l-1}}(b,y)$$
$$\mid a\in S_N(SG_u^{l-1})\wedge b\in S_N(SG_v^{l-1})\}).$$

위의 식은 $SPC_{SG_i^l}(x,y)=SPC_{SG_u^{l-1}\cup SG_i^{l-1}\cup S_{BW}(S_C(SG_i^l))}(x,y)$ 을 증명하고 있다. □

정리 4.4 $k\geq 1$인 Level $k+1$ 서브그래프 트리 ST를 $G(V,E)$로부터 구성하고 ST는 level k인 $HiTi$ 그래프 $H^k(P^k,A^k)$를 정의한다고 하자. 어떤한 노드 쌍 $x\in SG_i^1$, $y\in SG_j^1$대해서도 $D=S_{BW}(S_C(S_A(\{SG_i^1,SG_j^1\})))$ 인 $SPC_G(x,y)=SPC_{SG_i^1\cup SG_j^1\cup D}(x,y)$가 성립한다.

증명: level $k+1$인 ST가 루트 노드로 SG_1^{k+1}을 가지므로 $G=SG_1^{k+1}$이며, $SPC_G(x,y)=SPC_{SG_1^{k+1}}(x,y)$임을 알 수 있다. $l=LUB_{ST}(SG_i^1,SG_j^1)$라고 하자. 이때, $l<t\leq k+1$인 모든 t 대해서 $x,y\in S_A^t(SG_i^1)$인데, 이는 $S_A^t(SG_i^1)$가 $S_A^t(SG_j^1)$와 같기 때문이다. 정리 4.2에 의해서, 다음을 얻을 수 있다.

$$SPC_G(x,y)=SPC_{S_A^t(SG_i^1)\cup S_{BW}(S_C(S_A^{t+1}(SG_i^1)))}(x,y) \quad (1)$$

(1)로부터, $l<t\leq k+1$인 모든 t가 $x,y\in S_A^t(SG_i^1)$만족할 동안, 정리 4.2를 반복적으로 적용하여 탐색 공간 $S_A^k(SG_i^1)$을 줄일 수 있다. 그 결과 다음을 얻는다.

$$SPC_G(x,y)=SPC_{\cup_{t=l+1}^{k}S_{BW}(S_C(S_A^t(SG_i^1)))\cup S_A^l(SG_i^1)}(x,y) \quad (2)$$

이제, (2)로부터 정의된 아래와 같은 감소된 탐색 공간을 살펴보자.

$$\cup_{t=l+1}^{k+1}S_{BW}(S_C(S_A^t(SG_i^1)))\cup S_A^l(SG_i^1) \quad (3)$$

이때, (3)상에서 계산된 최단 경로는 G에서 계산된 것과 같다. (3)에서의 $S_A^t(SG_i^1)$로부터, $S_A^{l-1}(SG_i^1)\neq S_A^{l-1}(SG_j^1)$

일 때, $S_C(S_A^l(SG_i^1))$가 $S_A^{l-1}(SG_i^1)$와 $S_A^{l-1}(SG_j^1)$을 포함한다는 사실을 알 수 있다. $x \in S_A^{l-1}(SG_i^1)$이고 $y \in S_A^{l-1}(SG_j^1)$이므로, (3)은 정리 4.3을 적용하여 다음과 같이 나타낼 수 있다.

$$\cup_{t=l+1}^{t=k+1} S_{BW}(S_C(S_A^t(SG_i^1))) \cup S_A^{l-1}(SG_i^1) \cup S_A^{l-1}(SG_j^1) \cup S_{BW}(S_C(S_A^l(SG_i^l))) \quad (4)$$

(4)에서의 $S_A^{l-1}(SG_i^1)$을 알기 위해서, 노드 x에서 $S_N(S_A^{l-2}(SG_i^1))$의 경계 노드들까지의 최단 경로와 $S_N(S_A^{l-2}(SG_i^1))$의 경계 노드에서 $S_N(S_C(S_A^{l-1}(SG_i^1)))$의 경계 노드들까지의 최단 경로를 계산할 필요가 있다. (4)에서의 $S_A^{l-1}(SG_j^1)$을 알기 위해서, 노드 y에서 $S_N(S_A^{l-2}(SG_j^1))$의 경계 노드들까지의 최단 경로와 $S_N(S_A^{l-2}(SG_j^1))$의 경계 노드에서 $S_N(S_C(S_A^{l-1}(SG_j^1)))$의 경계 노드들까지의 최단 경로를 계산할 필요가 있다. 이때, 정리 4.1을 이용해서 (4)를 다음과 같이 다시 표현할 수 있다:

$$\cup_{t=l+1}^{t=k+1} S_{BW}(S_C(S_A^t(SG_i^1))) \cup S_A^{l-2}(SG_i^1) \cup S_A^{l-2}(SG_j^1) \cup_{t=l-1}^{t=l} S_{BW}(S_C(S_A^t(SG_i^1, SG_j^1))) \quad (5)$$

$l < t \le k+1$일 때, $S_A^t(SG_i^1, SG_j^1)$는 $S_A^t(SG_i^1)$ 또는 $S_A^t(SG_j^1)$와 같다. 유사하게, 정리 4.1을 반복적으로 (5)에 적용하면, 다음을 얻을 수 있다:

$$\cup_{t=l+1}^{t=k+1} S_{BW}(S_C(S_A^t(SG_i^1))) \cup S_A^1(SG_i^1) \cup S_A^1(SG_j^1) \cup_{t=1}^{t=l} S_{BW}(S_C(S_A^t(SG_i^1, SG_j^1))) \quad (6)$$

(6)을 다시 간단하게 정리하고 다음과 같은 노드 x에서 y로의 최단 경로 비용을 계산하기 위한 감소된 탐색 공간을 얻는다:

$$\cup_{t=1}^{t=k+1} S_{BW}(S_C(S_A^t(SG_i^1, SG_j^1))) \cup SG_i^1 \cup SG_j^1 \quad (7)$$

(6)과 $\cup_{t=1}^{t=k+1} S_A^t(SG_i^1, SG_j^1) = S_A(SG_i^1, SG_j^1)$로부터, $D = S_{BW}(S_C(S_A(\{SG_i^1, SG_j^1\})))$일 때 $SPC_G(x, y) = SPC_{SG_i^1 \cup SG_j^1 \cup D}(x, y)$임을 증명한다. □

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] R. Agrawal, S. Dar, and H. Jagadish, "Direct Transitive Closure Algorithms: Design and Performance Evaluation," In *ACM Transactions on Database Systems*, Vol. 15, No.3, pp,427-458, September 1990.

[4] S. Dar and R. Ramakirishnan, "A Performance Study of Transitive Closure Algorithms," In *Proc. ACM-SIGMOD 1994 Int'l Conference on Management of Data*, pp.454-465, 1994.

[5] K. Hua, J. Su, and C. Hua, "Efficient Evaluation of Traversal Recursive Queries Using Connectivity Index," In *Proc. IEEE 9th Int'l Conf. on Data Engineering*, pp.549-558, 1993.

[6] Y. Ioannidis and R. Ramakirishnan, "Efficient Transitive Closure Algorithms," In *Proc. of the 14th VLDB Conference*, pp.382-394, 1988.

[7] Y. Ioannidis, R. Ramakirishnan, and L. Winger "Transitive Closure Algorithms Based on Graph Traversal," In *ACM Transactions on Database Systems*, Vol. 18, No.3, pp.513-576, September 1993.

[8] B. Jiang, "A Suitable Algorithm for Computing Partial Transitive Closures in Databases," In *Proc. IEEE 6th Int'l Conf. Data engineering*, pp.264-271, 1990.

[9] G. Qadah, L. Henschen, and J. Kim, "Efficient Algorithms for the Instantiated Transitive Closure Queries," In *IEEE Transactions on Software Engineering*, Vol. 17, No. 3, pp.296-309, March 1991.

[10] A. Rosenthal, S. Heiler, U. Dayal, and F. Manola, "Traversal Recursion: A practical approach to supporting recursive applications" In *Proc. IEEE 3rd Int'l Conf. Data Engineering*, pp.580-590, 1987.

[11] I. Toroslu and G. Qadah, "The Efficient Computation of Strong Partial Transitive-Closures," In *Proc. IEEE 9th Int'l Conf. Data Engineering*, pp.530-537, 1993.

[12] 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.

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

[14] 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 Application*

(CAIA '94), pp.149-155, 1994.

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

[16] 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.

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

[18] 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.

[19] 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.

[20] S. Shekhar, A. Fetterer, and B. Goyal, "Materialization Trade-Offs in Hierarchical Shortest Path Algorithms," In *Proc. 1997 Symposium on Spatial Databases*, 1997.

[21] W. McCormick Jr., P. Schweitzer, and T. White, "Problem Decomposition and Data Reorganization by a Clustering Technique," In *Operation Research*, Vol. 20, No. 5, pp.993-1009, 1972.

[22] R. Lipton and R. Tarjan, "Application of a planar seperator theorem," In *SIAM Journal on Computing*, Vol. 9, No. 3, pp.615-627, 1980.

[23] G. Miller, S. Teng, and S. Vavasis, "A unified geometric approach to graph separators," In *Proceedings of 31st Annual Symposium on Foundations of Computer Science*, pp.538-547, 1991.

[24] G. Miller, S. Teng, W. Thurston, and S. Vavasis, "Automatic mesh partitioning," In *A. George, J. Gilbert, and J. Liu, editors, Sparse Matrix Computations: Graph Theory Issues and Algorithms*, (An IMA Workshop Volume), Springer-Verlag, New York, 1993.

[25] M. Houstma, P. Apers, and G. Schipper, "Data Fragmentation for Parallel Transitive Closure Strategies," In *Proc. IEEE 9th Int'l Conference on Data Engineering*, pp.447-456, 1993.

[26] D. Galperin, "On the optimality of A*," In *Artificial Intelligence*, Vol. 8, No. 1, pp.69-76, 1977.

[27] R. Agrawal and J. Kiernan, "An Access Structure for Generalized Transitive Closure Queries," In *Proc. IEEE Ninth Int'l Conf. on Data Engineering*, pp.429-438, April, 1993.

[28] J. Banerjee, S. Kim, W. Kim, and J. Garza, "Clustering a DAG for CAD Databases," In *IEEE Transaction on Software Engineering*, Vol. 14, No. 11, pp.1684-1699, 1998.

[29] P. Larson and V. Deshpande, "A File Structure Supporting Traversal Recursion," In *Proc. ACM-SIGMOD 1989 Int'l Conference on Management of Data*, pp.243-252, 1989.

[30] S. Shekhar and D. Liu, "CCAM: A Connectivity-Clustered Access Method for Networks and Network Computations," In *IEEE Transaction on Knowledge and Data Engineering*, Vol. 9, No. 1, pp.102-119, 1997.

[31] 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.

[32] J. Pearl, In *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison Wesley, Reading, Mass., 1984.

[33] Y. Huang, N. Jing, and E. Rundensteiner, "Effective Graph Clustering for Path Queries in Digital Map Databases," In *Proc. 5th Int'l Conference on Information and Knowledge Management*, pp.215-222, 1996.

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

정 성 원

1988년 서강대학교 전자계산학과 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 Agents in Wireless Enviornment ITS/GIS, parallel processing in database systems, distrivuted database, spatial database