■ 論 文 ■

교통망 분석에서 K경로탐색 알고리즘에 관한 연구
(Ordered Heap Tree 구축방식을 중심으로)

A Study on the K Shortest Paths Algorithm in a Transportation Network
(Using Ordered Heap Tree)

임 강 원 (서울대학교 환경대학원 교수)
양 승 묵 (서울대학교 환경대학원 석사과정)
신 성 일 (서울시정개발연구원 도시교통연구부 연구위원)

목 차

Key Words : 힙(Heap) 정렬, Sidetrack, 링크비루프(No Link Repeated Paths), 깊이우선탐색, 꼬리노드, 머리노드

요 약

일반적으로 현실(특히 도시) 교통망에서 교차로를 반복해서 방문하는 통행은 존재하지만, 가로를 반복해서 주행하는 현상은 존재하지 않는다. 교통망에서의 루프형 통행은 링크의 반복이 허용되지 않는 링크 비루프(Link Loopless Path) 통행으로 축소된다.

본 연구에서는 K개의 경로탐색에서 기존의 방식과 달리 Heap Ordered Tree를 이용하여 월등한 수행속도(최악의 경우) $O(m + n\log n + K\log K)$로서 수행되는 Eppstein 알고리즘과 Jiménez et al의 LVEA을 고찰하여, 이들 알고리즘의 문제점인 링크루프의 발생을 제어하는 방안을 제어하도록 한다. 사례연구를 통하여 제안된 알고리즘을 검증·평가한다.

We propose a modified version of "a Lazy Version of Eppstein's k shortest paths Algorithm(LVEA)" which can find the k shortest paths in total time $O(m + n\log n + K\log K)$ in the worst-case. The algorithm we propose, since the Link repeated paths are all eliminated when enumerating k shortest paths, is No link repeated paths algorithm that is suitable in a transportation network.

Ⅰ. 서론

1. 연구의 배경

최근 지능형 교통체계(Intelligent Transportation Systems: ITS)에 대한 연구가 세계적으로 활발하게 진행되면서 ITS분야의 하나인 첨단여행자정보체계(Advanced Traveler Information System: ATIS)에 대한 연구도 더불어 활발하게 진행되어왔다. 첨단여행자정보체계(ATIS)의 주요목적 중 하나는 여행자가 요구하는 최적의 경로정보를 제공하는 것이며, 그 목적을 보다 효과적으로 달성하기 위해서는 경로정보의 제공에 있어 단일의 최단경로에 대한 정보만을 제공하는 것보다 여행자가 그 자신이 처한 환경과 여건에 따라 인지적으로 결정할 수 있도록 선택의 폭이 다양한 복수의 경로정보를 제공하는 것이 바람직하다. 이 복수의 경로정보 제공을 위해 (K)개의 경로를 탐색하는 알고리즘이 활용되고 있으며, 이 알고리즘은 K경로탐색알고리즘(K shortest paths algorithm)이라 칭해진다.

특히 서울시 통합대중교통체계의 개편으로 인해 관계기관간(특히 지하철-전철)의 수입금 정산에 대한 개선방안이 이슈가 되고 있으며, 개선된 수입금 정산 모형에 관한 연구가 현재 진행 중에 있는 실정이다. 기존의 수입금 정산 모형은 단일경로배정방식을 취하고 있기 때문에 출발역과 도착역 사이에서 발생하는 수요를 단일경로에 모두 배정하게 되므로 산정된 단일경로와 유사한 노선을 보유하고 있는 기관은 정산 시 손해가 발생하게 되어 분쟁이 유발하고 있다. 이를 해결하기 위한 방안으로 다수의 유사경로를 탐색하여 그 유사경로들에 대해서 합리적인 수요를 배분하기 위한 모형에 대한 연구가 이루어지고 있으며, 다수의 유사경로 탐색 역시 K경로탐색알고리즘이 활용되고 있다[1].

이외에도 K경로탐색알고리즘은 합리적 통행배정, 혼잡시 우회도로 안내, 복수속성을 갖는 최적경로탐색 등의 분야에서도 그 유용성을 인정받고 있으며(임강원&임용택, 2003, p125), 이러한 이유로 인해 여러 가지 K경로탐색알고리즘에 대한 연구가 현재까지 진행되어 오고 있다.

2. 연구의 목적

기존의 K경로탐색알고리즘은 1)표지확정 및 갱신방식인 Shier(1979)의 알고리즘과 경로삭제방법(Path Deletion Method) 중 2)부분경로삭제 방식(path Partition Algorithm)에 기반을 둔 Yen(1971)의 알고리즘, 3)전체경로삭제방식(Entire Path Deletion)에 기반을 둔 Martins(1984)와 Azevedo et al(1993)의 알고리즘, 그리고 4)수행속도의 개선을 위해 구축된 Ordered-Heap Tree를 기반으로 한 Eppstein(1998)의 알고리즘과 Jiménez et al(2003)의 Lazy Version of Eppstein's Alogrithm인 4가지로 구분될 수 있다.

최근까지 K경로탐색알고리즘에 관하여 진행되어 온 연구 중 수행속도측면에서 가장 진보된 알고리즘으로 평가받으며 활용되고 있는 알고리즘은 Azevedo et al(1993)의 알고리즘이다. 하지만, Azevedo et al(1993) 알고리즘은 현실적인 교통망에 적용되기에는 어려움이 있으며, 현재에는 이 알고리즘의 단점인 회전지체/금지 특성의 반영문제 및 루프문제가 해결되어 복합교통망에 적용하는 연구가 활발히 이루어지고 있다[2].

노드 수를 $|V|$, 링크 수를 $|L|$이라 할 때, Azevedo et al(1993) 알고리즘의 최대수행시간은 $O(K^2|V|)$가 되며, 회전지체/금지 및 루프문제가 해결된 알고리즘의 최대수행시간은 $O(K^2|L|)$이 된다(신성일, 2004). 그렇지만, Azevedo et al(1993) 알고리즘이 수행속도 측면에서 기존의 알고리즘들에 비해 개선되었다 하더라도 서울시와 같은 대규모 네트워크에서 수행이 되면 수행시간측면에서 보다 절감시켜야할 필요성을 가지게 된다.

본 연구에서는 (K)개의 경로를 산출함에 있어 Azevedo et al(1993) 알고리즘 보다 더욱 빠른 시간 내에 (K)개의 경로를 나열하는 Eppstein(1998) 알고리즘과 Eppstein 알고리즘을 보다 개선시킨 Lazy Version of Eppstein's Algorithm을 고찰하고, 보다 현실적인 교통망에 적용이 가능하도록 개선된 알고리즘을 제안하고자 한다.

즉 본 연구의 목적을 요약하자면, **첫째**, 현재까지 제시된 알고리즘들 중 수행속도 측면에서 획기적으로 개선된 방안이라 할 수 있는 Ordered Heap Tree 기반의 Eppstein(1998) 알고리즘을 고찰하고, **둘째**, Eppstein(1998) 알고리즘의 수행속도를 보다 향상시키기 위해 Jiménez et al(2003)에 의해 제안된 Lazy Version of Eppstein's Algorithm(이하 LVEA)을 고찰하여 이를 기반으로 LVEA를 구축하며,

1) 신성일 외, "수도권 도시철도 수입금 정산 분석모형", 대한교통학회 여름특별호, 2005, pp.157-167
2) 신성일 외, "K 링크 비루프 최적경로 탐색알고리즘과 복합대중교통망에의 활용", 한국교통연구원, 「교통정책연구」, 제11권, 2004, pp.83-103

셋째, 이들 알고리즘을 교통망에 적용함에 있어 한계가 되는 K경로 열거 시 링크의 반복을 제어한 링크비루프(No Link Repeated Path) 경로탐색방식을 제안하고자 한다. 링크비루프는 (특히 도시)교통망에서 발생하는 운전자의 합리적 통행행태를 반영하는 장점을 제공한다(신성일, 2004).

Ⅱ. 선행 연구의 고찰

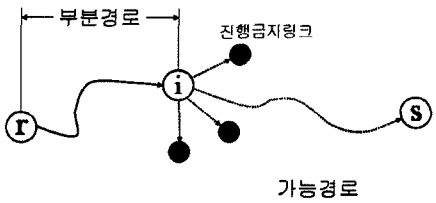
기존의 K경로탐색알고리즘에서 Shier(1979)가 제안한 알고리즘은 일반적인 최적경로알고리즘과 같이 one-to-all (Moore, 1958; Dijkstra, 1959) 방법으로 탐색된다. 반면, Yen(1971)과 Martins (1984), Azevedo et al(1993), Eppstein(1998), Jiménez et al(2003)이 제안한 알고리즘들은 one-to-one 방법에 근거하여 K개의 경로를 탐색한다. 본 장에서는 Eppstein(1998)과 탐색기법이 유사한 Yen(1971), Martins(1984), Azevedo et al(1993), Eppstein (1998), Jiménez et al(2003) 알고리즘을 대상으로 고찰한다.

1. 경로삭제기반 K shortest paths 알고리즘

1) Yen 알고리즘

Yen(1971)은 K개의 경로탐색에 있어 링크단위의 부분 삭제방안(Path Partition Algorithm)에 기반을 둔 비루프 경로탐색을 위한 효율적인 알고리즘을 제시하였다. Yen 알고리즘의 기본개념은 일단 최단경로가 탐색되면 나머지의 다른 경로들 중에서 이전의 최단경로를 제외하고 가장 비용이 낮은 경로를 탐색하면서 두 번째, 세 번째 등 비용이 낮은 경로들을 결정하는 방법이다.

이는 K번째의 경로는 K-1개의 경로를 포함하며 경로집합을 기반으로 발견된다는 사실에 근거하여 개발되었으며, 결국 이미 탐색된 K-1개의 경로를 고려하여 K번째의 다음 경로를 탐색하는 방안이라고 할 수 있다. 이 알고리즘은 K번째 경로를 발견하기 위해서는 대기경로집합에 포함되어 있는 모든 경로들을 비교하여 최소비용을 가진 경로를 선택하는 것이다. Yen 알고리즘을 간략하게 설명하자면 다음과 같다. 〈그림 1〉에서 출발지 ⓡ에서부터 ⓘ까지의 부분경로와 부분경로의 마지막 노드인 ⓘ를 출발노드로 경로의 중복을 방지하기 위해 다음노드인 (ⓘ+1)로부터 이미 탐색되었던 링크를 제외한 방향으로 경로탐색 트리를 구축하여 도착지 노드 ⓢ까지의 최단경로를 탐색하게 된다. 이때 탐색된



〈그림 1〉 Yen알고리즘 개념도

경로 중 최단경로는 가능경로로 구축하고, 부분경로와 탐색된 최단경로를 결합하여 K번째 경로를 선정하기 위한 후보경로에 포함한다. K번째 경로는 전체 후보경로 중에서 가장 최소 비용인 경로가 결정된다.

Yen 알고리즘의 수행시간은 노드수 $|V|$ 의 일반적인 네트워크에서 $O(K|V|^3)$이며, (K-1)개의 링크와 관련된 부분링크조합을 고려하여 최적경로 탐색을 수행하기 때문에 네트워크의 규모가 커질수록 수행시간이 큰 폭으로 증가하게 되는 단점을 내제하고 있다.

2) Martins 알고리즘

Martins(1984)가 제안한 알고리즘은 네트워크의 변형(Enlarged Network)을 통해 경로의 전체를 삭제하는 기법(Entire Path Deletion)에 기반을 두고 있다.

Martins 알고리즘은 (1)두 지점간 최적경로 p의 발견을 위한 최적경로 알고리즘과, (2)최초 형성된 네트워크 N에 대한 새로운 네트워크 N′를 생성시키기 위한 경로삭제 알고리즘인 2가지의 세부 알고리즘으로 구성되어 있으며, 개략적 개념은 우선 최적경로 p가 탐색되면, 그 p경로를 네트워크 N에서 삭제하여 다시 최적경로 알고리즘을 수행한다는 것이다. 이때 새롭게 확장된 노드와 링크를 네트워크 N에 추가하여 구성된 네트워크 N′는 경로 p를 제외한 모든 경로의 탐색이 가능하도록 구축된다.

N_1을 기본네트워크로 하여 순차적인 (K)개의 경로를 탐색한다는 것은 {N_1, N_2, N_3, …, N_k}의 순차적인 네트워크를 구축함을 의미하며, 이 경우 j번째의 네트워크 N_j로부터 j번째 경로 p_j가 탐색된다. 이 알고리즘에서는 최적경로 알고리즘은 K번 수행되며 경로삭제 알고리즘은 K-1번 수행된다.

경로삭제알고리즘의 세부수행과정은 다음과 같다.

〈Martins의 경로삭제 알고리즘〉

○ **변수의 정의**

• $p = \{v_0, v_2, \cdots, v_{m-1}, v_m\}$: 여기서, p는 노드 v_0 ~

v_m의 순서로 구성된 최적경로, v_0=R은 출발지 노드, v_m=S는 도착지 노드이며, m≥3

- $O(u)=\{(u,v)\in A|u, v\in N\}$: $O(u)$ 어느 노드 u와 연결된 유출링크(Outgoing Links)집합
- $I(u)=\{(v,u)\in A|v, u\in N\}$: $I(u)$는 어느 노드 u와 연결된 유입링크(Incoming Links)집합

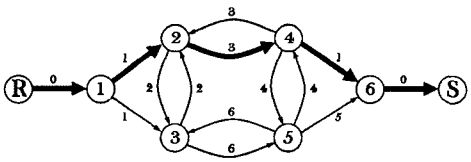
○ **경로삭제 알고리즘**

Martins 알고리즘은 경로 p를 삭제하여 네트워크 N′를 생성하기 위하여 새로운 노드를 추가하고, 이들 노드에서 유출링크(Outgoing Link)를 연결하며, p의 첫 번째 경로를 삭제하는 것으로서 아래의 3단계로 이루어져 있다.

【Step 1】 $N=N\cup\{v_1', \cdots, v_{m-1}'\}$

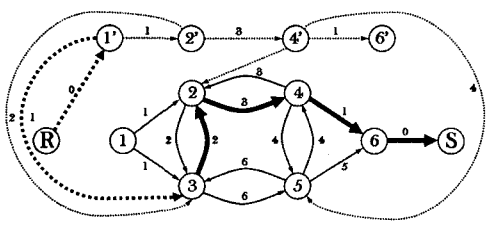
【Step 2】 $O(v_0)=O(v_0)-\{(v_0, v_1)\}\cup\{(v_0, v_1')\}$

【Step 3】 $O(v_j')=\{(v_j', u)|(v_j, u)\in O(v_j); u\neq v_{j+1}\}$
$\cup\{(v_j', v_{j+1}')\}$, $\forall j=\{1, \cdots, m-2\}$
$O(v_{m-1})=\{(v_{m-1}', u)|(v_{m-1}, u)\in O(v_{m-1})\}$



〈그림 2〉 네트워크 N

〈그림 2〉와 같은 간단한 네트워크를 예로 들어 위의 알고리즘을 설명하자면, 최적경로 p는 {R, 1, 3, 4, 6, S}의 노드순서로 네트워크 N에 굵은 실선으로 표시되어 있다.



〈그림 3〉 Martins 알고리즘 네트워크 N′

또한 경로삭제 알고리즘으로 생성된 네트워크 N′는 〈그림 3〉에 나타나 있다. Step1에서새로운 노드집합 {1′, 2′, 4′, 6′}이 추가되었고, Step2에서 p의 첫 번째 링크인 (R, 1)이 삭제되고 링크(R, 1′)로 대체되었으며, Step3에서는 유출링크집합{(R, 1′), (1′, 3), (2′, 3), (4′, 2), (4′, 5), (1′, 2′), (2′, 4′), (4′, 6′)}이 추가되었다.

〈그림 3〉의 네트워크 N′를 기반으로 최적경로 알고리즘을 수행하여 p′는 {R, 1′, 3, 2, 4, 6, S}로서 두 번째 탐색된 경로가 된다. 이상과 같이 Martins가 제안한 알고리즘의 최대수행시간은 $O(K^3|V|)$이다.

3) Azevedo et al 알고리즘

Azevedo et al(1993)는 Martins 알고리즘의 수행능력을 향상시키기 위해 보다 개선된 알고리즘을 제안하였다. 이 알고리즘은 최대수행시간이 $O(K^2|V|)$으로 Martins의 방법보다 효율적이다.

Azevedo et al의 알고리즘 역시 Martins의 알고리즘과 마찬가지로 네트워크의 변형을 통해 경로 전체를 삭제하는 기법(Entire Path Deletion)에 기반을 두고 있다. 하지만, Martins가 제안한 방법과는 달리 경로 p의 마지막 링크를 삭제하고 새로 추가된 노드들의 유입링크(Incoming Link)집합을 네트워크 N에 추가한다는 부분에서 그 차별성을 가지고 있다. Martins 알고리즘에서는 최적경로알고리즘이 K번, 경로삭제 알고리즘이 K-1번 수행되는 반면, Azevedo at al 알고리즘의 경우 경로 p의 마지막 링크를 삭제되고 새로 추가된 노드의 유입링크 집합이 추가되어 N′가 생성되므로 기존의 네트워크인 N의 링크 및 노드의 표지는 새로운 네트워크 N′에서 영구표지로 남아있게 된다. 따라서 기존 네트워크 N에서 삭제된 경로 p의 부분경로(Subpath)가 N′의 최적경로에 포함된다는 사실에 근거하여 (K-1)번의 최적경로알고리즘 수행을 절약할 수 있다. 결국, 최대수행시간은 $O(K^2|V|)$으로 절감된다.

Azevedo et al 알고리즘의 수행과정은 다음과 같다.

【Step 1】 $N=N\cup\{v_1', \cdots, v_{m-1}'\}$

【Step 2】 $I(v_1')=\{(u, v_1')|(u, v_1)\in I(v_1); u\neq v_0\}$
$I(v_j')=\{(u, v_j')|(u, v_j)\in I(v_j); u\neq v_{j-1}\}$
$\cup\{(v_{j-1}', v_j')\}$, $\forall j=\{2, \cdots, m-1\}$

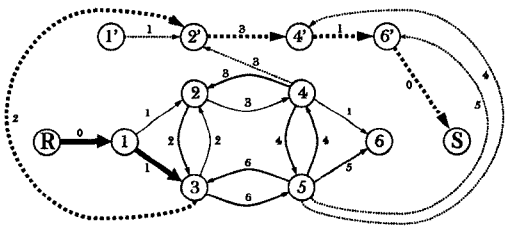
【Step 3】 $I(v)=O(v_m)-\{(v_{m-1}, v_m)\}\cup\{(v_{m-1}', v_m)\}$

Azevedo et al의 알고리즘을 요약하면, 1)최적경로

탐색 알고리즘, 2)네트워크 확장알고리즘, 그리고 3)확장네트워크의 추가노드 및 링크표지확정 알고리즘으로 구분되며 알고리즘의 구성은 다음과 같다.

【Step 1】 최적경로알고리즘의 수행으로 p_1의 발견
【Step 2】 k=2부터 K까지 반복
네트워크 확장알고리즘으로 N에서 N′의 구축
N′에 추가된 노드 및 링크표지 확정
출발지에서 도착지까지 최적경로탐색

Martins의 알고리즘 설명을 위해 제시되었던 네트워크인 〈그림 2〉를 예로 들어 Azevedo et al 알고리즘을 통하여 수행한 네트워크 N′는 〈그림 4〉에 도식되어 있다.



〈그림 4〉 Azevedo et al 알고리즘 네트워크 N′

Step 1에서 새로운 노드집합 {1′, 2′, 4′, 6′}이 추가되었고, Step 2에서 유입링크집합인 {(3, 2′), (4, 2′), (5, 4′), (1′, 2′), (2′, 4′), (4′, 6′), (5, 6′), (6′, S)}가 추가되었다. Step3에서는 경로 p의 마지막 링크인 (6, S)가 삭제되고 링크(6′, S)로 대체되었다.

네트워크 N의 노드 및 링크표지는 N′에 영구적으로 고정되었고, 삭제된 경로 p의 부분경로는 N′의 표지검색만으로 결정될 수 있기 때문에 최적경로알고리즘의 수행과정 없이 p′={R, 1, 3, 2′, 4′, 6′, S}의 탐색이 가능하다.

2. Heap Ordered Tree기반 K shortest paths 알고리즘

Eppstein(1998) 알고리즘과 Jiménez et al(2003)의 Lazy Version of Eppstein's Algorithm(LVEA)은 Heap Ordered Tree를 기반으로 K shortest paths를 탐색하는 알고리즘으로서, Eppstein(1998) 알고리즘은 n개의 노드와 m개의 링크들로 구성된 네트워크가 주어졌을 때, 주어진 출발지 노드로부터 도착지 노드로의 K최단경로 탐색에 수행되는 총시간은 $O(m+n\log n+k\log k)$, Jiménez et al(2003)의 LVEA는 Eppstein 알고리즘에서 생성되는 $D(G)$를 제어하여 Eppstein 알고리즘의 수행시간을 보다 단축시킨 알고리즘으로서 수행속도 측면에서 기존의 경로삭제기반의 방안들에 비해 보다 향상된 방안이다.

1) Heap 정렬의 고찰

Eppstein(1998)이 제안한 K shortest paths 알고리즘은 K경로의 탐색 시 힙 정렬(Heap Sort)을 이용하는 알고리즘이다. 힙(Heap)이란 우선순위 큐(Priority Queue)의 일종으로 우선순위가 높은 요소를 효율적으로 선택할 수 있는 자료구조를 의미하며, Eppstein 알고리즘에서 힙은 나무(Tree) 구조로 구현된다. 힙 정렬(Heap Sort)은 다른 O(NlogN) 알고리즘에 비해 부가적인 메모리가 전혀 필요 없으면서도 O(NlogN)의 성능을 가지는 매우 빠른 정렬법이며 입력자료에도 거의 무관하게 고른 성능을 보여주는 뛰어난 성능을 가지고 있다(이재규, 1994). 힙(Heap)에 대해 보다 구체적으로 설명하자면, 힙은 전이진 트리(Binary Tree)로서 각 노드의 값이 자식 노드의 값보다 작지 않은 트리이다. 이 힙은 전이진 트리이므로 노드의 삽입과 삭제가 발생해도 힙의 성질은 만족해야 하므로 힙으로 재구성해야 한다. 또한, 힙 정렬(Heap Sort)이란 힙의 특성상 근노드에 가장 큰 값이 있으므로 근노드를 제거한 후 트리를 재구성하며, 이 과정을 반복적으로 수행하면 그 레코드들은 정렬된다. 힙 정렬의 단계는 다음과 같다[3].

【Step 1】 주어진 입력 트리를 힙으로 바꾼다.
【Step 2】 힙의 근노드와 가장 마지막 노드를 교환하고, 이 트리를 힙이 되도록 한다. 단, 교환된 마지막 노드는 마지막 노드로 간주하지 않는다.
【Step 3】 Step 2의 과정을 정렬이 끝날 때까지 반복한다.

2) Eppstein 알고리즘

K shortest paths 탐색에 있어 Eppstein(1998) 알고

3) 지면제약으로 이 부분에 대한 상세한 설명은 김동수 외(2002), C로 구현한 자료구조, 도서출판 OK Press, pp230~235를 참조

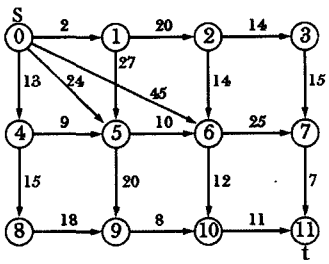
리즘은 크게 3가지 단계로 나뉜다. 첫째, Fibonacci heap이 결부된 Dijkstra 알고리즘을 통한 단일 도착지 최단경로 가지(single-destination shortest path Tree)의 구축과 단일 도착지 최단경로 가지로부터 나오는 sidetrack들의 산출. 둘째, 이들 sidetrack들을 이용한 힙인 $H_{out}(v)$, $H_T(v)$, 그리고 $H_G(v)$ 의 구성. 셋째, 구성된 힙들 간의 연결(pointing)로 인한 방향성 그래프 $D(G)$의 생성 및 K 최단경로의 도출이다. 이들 각 단계에 관한 세부적인 설명은 다음과 같다.

(1) Step 1 : 최단경로 가지의 구축 및 sidetracks 산출

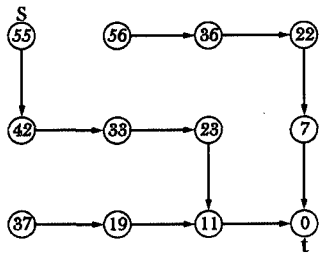
〈그림 5〉과 같이 노드집합인 $V(G)$와 링크집합 $E(G)$로 이루어진 네트워크 $G=(V, E)$에서 출발지 노드(s)를 ⓪, 도착지 노드(t)를 ⑪이라 할 경우, 우선 Fibonacci heap이 결합된 Dijkstra알고리즘에 의해 〈그림 6〉와 같이 도착지 노드(t)를 가지는 단일 도착지 최단경로 가지(single-destination shortest path tree)인 T를 구축한다.

〈그림 6〉의 각 노드 상에 나타나있는 숫자는 노드번호가 아닌 그 노드에서 도착지 노드(t)까지의 최단 비용을 나타내며, 최단경로 가지(tree) T에 형성된 경로들은 각각의 모든 노드 v 에서 도착지 노드 t까지의 최단경로를 의미한다.

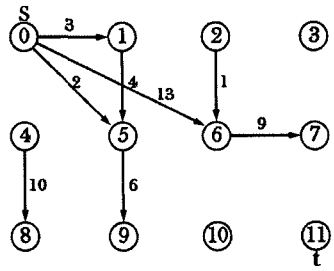
이와 같이 T가 구축되면 T에 나타난 s→t 최단경로를 선택하지 않고 다른 노드를 선택하여, 최단경로가 아닌 다른 경로가 만들어질 때 발생하게 되는 추가비용



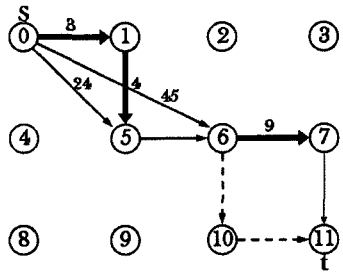
〈그림 5〉 네트워크 G



〈그림 6〉 최단경로 tree T & 거리



〈그림 7〉 $\delta(e)$로 표식된 $G-T$의 링크



〈그림 8〉 경로 p 및 $sidetracks(p)$(굵은선)

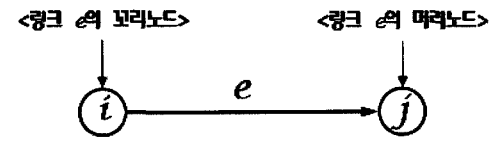
(여기서의 비용은 거리)이 나타나게 되는데, 이렇게 다른 노드를 선택하여 생기는 링크(e)를 sidetrack이라 한다. 네트워크 G에서의 sidetrack들은 〈그림 7〉에 표시되어있다.

이렇게 최단경로 T에 포함되지 않는 링크(e)인 이들 sidetrack들의 값은 $\delta(e)$로 나타내며, 그 값은

$$\delta(e) = l(e) + d(head(e), t) - d(tail(e), t)$$

로 정의된다. 여기서, $l(e)$는 sidetrack이 되는 링크(e)의 거리 값, $d(head(e), t)$는 링크(e)의 머리노드에서 도착지 노드 t까지의 거리, 그리고, $d(tail(e), t)$는 링크(e)의 꼬리노드에서 도착지 노드 t까지의 거리이다.

링크(e)의 머리노드$(head(e))$와 꼬리노드$(tail(e))$에 대한 개념은 〈그림 9〉를 참조하면 쉽게 이해될 것이다.



〈그림 9〉 링크(e)의 머리노드 및 꼬리노드

상기 과정에 따라 네트워크 G에서의 sidetrack들과 그 δ값들만을 나타낸 것이 〈그림 7〉이며, 이 sidetrack들은 결국 기본 네트워크인 G에서 최단경로 가지(tree)인 T를 뺀 $(G-T)$와도 같다.

최단경로 가지(tree) T에서 s→t로 가는 최단경로 외에 s→t로 가는 다른 경로들은 〈그림 8〉과 같이 앞서 형성된 sidetrack들의 조합으로부터 이루어질 수 있다는 것을 유추할 수 있다.

여기서, s→t로 가는 최단경로 외의 어떤 경로p에 대해서는 아래와 같이 성립된다.

$$l(p)=d(s,t)+\sum_{c\in sidetracks(p)}\delta(e)=d(s,t)+\sum_{c\in p}\delta(e)$$

이상이 Eppstein 알고리즘의 Step 1에 해당한다.

이후 단계로 이들 sidetrack들의 힙 구성 및 연결로 인한 K 경로들의 나열이 이루어지며, 각각의 단계에 관한 세부설명은 아래의 Step 2와 Step 3에서 다루어진다.

(2) Step2 : 힙(Heap)의 구축

Step 2에서는 Step 1에서 도출된 단일 도착지 최단경로 가지(single-destination shortest path tree)인 T와 각 노드들에 대한 sidetrack의 값인 $\delta(e)$의 정보를 이용하여 힙(Heap)을 구축한다.

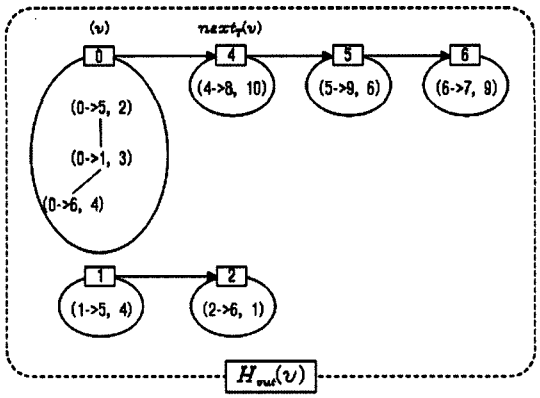
이 단계에서는 도착지 노드 (t)로부터 각 노드로의 최단경로를 따라 깊이우선탐색(Depth First Search: DFS)[4]을 실행하며, 그 경로를 따라 깊이우선탐색 시 방문되는 각 노드번호에 따라 메모리의 각 공간에 힙 $H_{out}(v)$와 $H_T(v)$를 구성한다. 깊이우선탐색의 이유는 이 탐색을 실행하면 최단경로 가지 T를 따라 힙(Heap)을 구성할 수 있기 때문이다.

가. Step 2.1 : $H_{out}(v)$ 구축

T에 있는 각 노드 v로부터 나가는 sidetrack(이하 $out(v)$)들로 이루어진 Heap tree인 $H_{out}(v)$를 각 노드번호에 따라 메모리에 할당한다. 이때, $H_{out}(v)$는 이진 가지(binary tree)이긴 하지만 근노드에서 나오는 자식노드(child)는 오직 하나로만 구성되어야 한다는 제약이 추가된다. $H_{out}(v)$는 힙(Heap)으로 구성되었으므로 각 노드번호에 할당된 힙에서의 근노드는 그 노드에서 나오는 $out(v)$ 중 δ값이 가장 작은 sidetrack이며, 이 근노드를 $outroot(v)$라 한다.

각각의 v에 대한 $H_{out}(v)$의 구성에 요구되는 시간은 $O(|out(v)|)$이며, 이 과정에 걸리는 총시간은 $\sum O(|out(v)|)=O(m)$이다.

네트워크 G로부터 구축되는 $H_{out}(v)$는 아래의 그림과 같다.



〈그림 10〉 네트워크 G로부터 구축되는 $H_{out}(v)$

나. Step 2.2 : $H_{out}(v)$ 및 $H_G(v)$ 구축

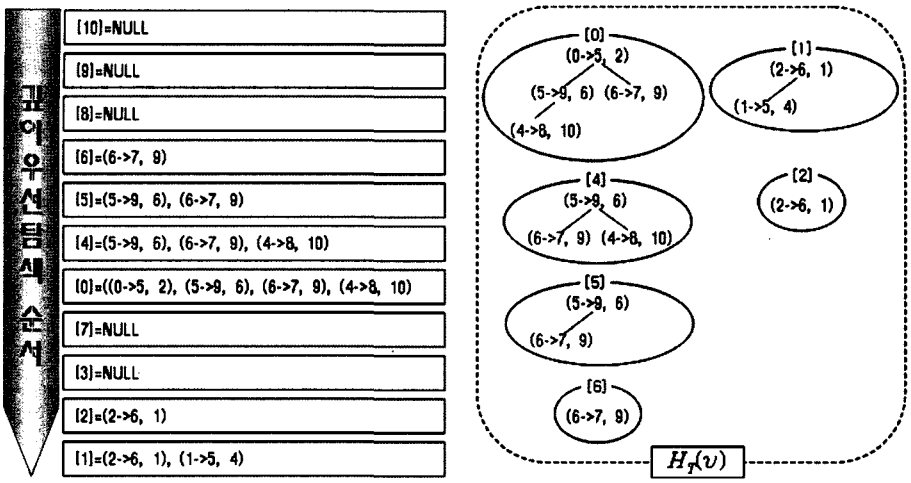
$H_T(v)$의 구축은 깊이우선탐색 시 방문되는 각 노드들의 배열(이 배열은 $H_{out}(v)$의 저장 공간과는 다른 공간에 저장)에 그 경로 상에 있는 각각의 노드로부터 나가는 $out(v)$ 중 가장 작은 δ값을 가지는 $out(v)$들만을 δ값에 따라 힙 정렬한다. 즉, $v\rightarrow t$ 경로에서 $H_{out}(v)$의 근노드인 $outroot(v)$를 $H_T(next_T(v))$에 삽입함으로써 $H_T(v)$를 구축하는 것이다.

각 노드들의 배열에 힙 정렬되어 실제 저장된 값들과 그 값들이 구성된 형태인 힙 $H_T(v)$를 그림으로 나타내면 〈그림 11〉와 같다.

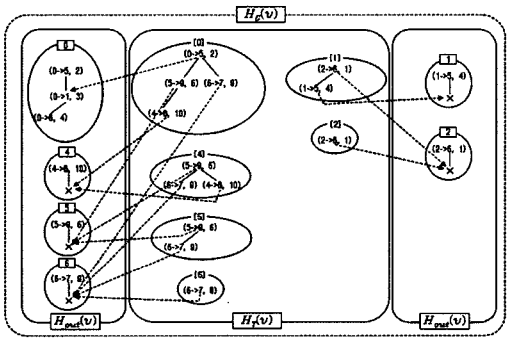
$H_G(v)$의 구축은 $H_T(v)$의 각 노드들이 $H_{out}(v)$의 근노드($outroot(v)$)의 자식노드를 Pointing하도록 함으로써 이루어지는데, 이 과정은 $H_T(v)$가 만들어지면서 동시에 $H_G(v)$를 형성되는 것이다. 따라서 $H_{out}(v)$의 근노드($outroot(v)$)의 자식노드가 존재하지 않는다면, Pointing은 이루어지지 않는다.

이 Pointing으로 인해 $H_G(v)$는 최대 3-heap(자식

4) 깊이우선탐색이란 방문하지 않은 정점이 존재할 때까지 방문하다가 더 이상 방문하지 않은 정점이 존재하지 않으면 방문했던 경로를 되돌아가다가 방문하지 않은 정점이 더 이상 존재하지 않을 때까지 방문하는 방법



〈그림 11〉 네트워크 G로부터 각 노드배열의 저장 값과 $H_T(v)$



〈그림 12〉 네트워크 G로부터 구축되는 $H_G(v)$

노드가 3개인 힙)으로 구성된다. 각각의 힙 $H_G(v)$의 형성은 $O(\log n)$시간내에 이루어지므로, 모든 노드에 대해서는 $O(n\log n)$이 걸린다. 따라서 Step 2에서 수행되는 총시간은 $O(m+n\log n)$이 된다.

네트워크 G에서 구축된 힙 $H_G(v)$는 〈그림 12〉와 같으며, 노드 ⓪의 배열에 형성된 힙 $H_G(v)$만이 $H_{out}(v)$의 근노드의 자식노드를 Pointing하고 있음을 알 수 있다.

(3) Step 3 : 방향성 그래프 $D(G)$의 생성 및 K최단경로 산출

가. Step 3.1 : $D(G)$의 생성

Step 2에서 힙 $H_G(v)$까지 구축이 완료되면, Step 3에서는 이들 $H_G(v)$들 간을 연결하여 방향성 그래프 $D(G)$를 생성한다.

구축된 각각의 힙인 $H_G(v)$내의 각 sidetrack들은 서로 연결되어있지 않고 독립적이라는 성격을 가진다. 다시 말하자면, 이들 sidetrack들은 하나의 $v\rightarrow t$ 경로상에 있는 sidetrack들이기 때문에 하나가 선택되면 다른 하나는 선택될 수 없다는 의미이다.

$D(G)$의 생성은 $H_G(v)$들 간의 연결로서 이루어지며 그 과정과 방법은 다음과 같다.

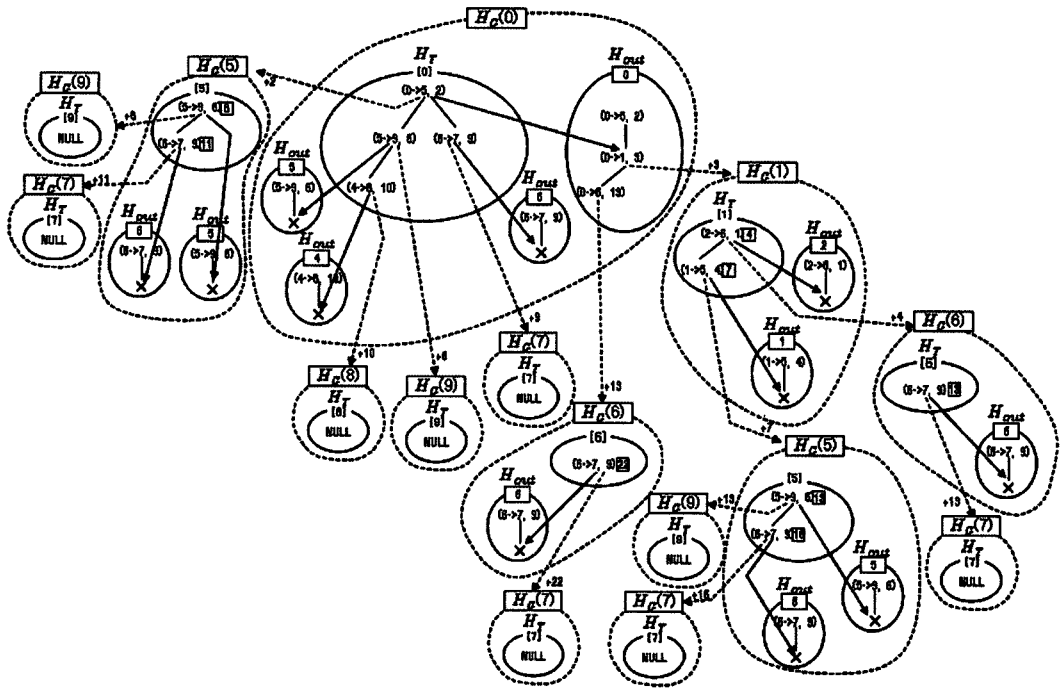
$H_G(v)$와 $H_G(w)$가 연결이 될 때에는 힙 $H_G(v)$내의 각 sidetrack이 선택된 후 그 sidetrack의 머리노드(w)와 연결되는 노드번호의 힙인 $H_G(w)$가 연결된다. 이때, 이 연결이 이루어지면서 선택된 sidetrack의 가중치는 연결되는 $H_G(w)$ 내의 sidetrack들 각각에 더해지게 된다. 즉, 노드 v의 힙인 $H_G(v)$내의 sidetrack들 중 하나가 선택되면서 다른 하나의 $s\rightarrow t$ 최단경로가 만들어지게 되고, 여기에 그 다음에 연결될 수 있는 노드인 w의 힙 $H_G(w)$가 연결이 되는 것이므로, $H_G(w)$내의 sidetrack은 그 $s\rightarrow t$ 최단경로 상에서 또다시 연결될 수 있는 최단경로 상의 sidetrack이므로 이전 sidetrack의 가중치가 $H_G(w)$에 더해지는 것이다.

이 과정의 반복으로 $D(G)$가 생성이 되며, 이 연결은 $H_G(w)$가 존재하지 않을 때까지 계속해서 수행된다. 이해를 돕기 위해 네트워크 G로부터 형성되는 $D(G)$를 〈그림 13〉[5]에 나타내었다.

나. Step 3.2 : K최단경로의 산출

이전단계에서 $D(G)$의 구축이 완료되면, $D(G)$에 있는 sidetrack들의 가중치 크기에 따라 다음과 같이

5) +숫자는 $H_G(v)$ 내의 sidetrack과 $H_G(w)$의 연결 시 추가되는 가중치이며, 사각형 내의 숫자는 가중치가 더해진 후의 sidetrack δ값



〈그림 13〉 네트워크 G로부터 생성된 $D(G)$

K최단경로의 산출이 이루어진다.

【Step 1】 $D(G)$에 있는 sidetrack들의 정보를 δ값의 크기에 따라 순서대로 queue Q에 저장한다. 이때, 가중치가 더해진 sidetrack은 그 이전의 sidetrack에 대한 정보(꼬리노드 및 머리노드)를 함께 가진 채로 저장한다.

【Step 2】 K=2~k인 동안 K최단경로와 총 비용을 산출한다. 여기에서, K최단경로는 Q로부터 차례대로 sidetrack이 산출되면 최단경로가지(Shortest path tree) T 상의 출발노드 (s)로부터 그 sidetrack의 꼬리노드와 머리노드에서 T 상의 도착노드 (t)까지의 최단경로를 열거한 K예약최단경로를 산출한 후 K최단경로를 열거한다. 만약, 이전의 sidetrack의 정보를 함께 가지고 있는 sidetrack이면, 그 sidetrack간의 연결 역시 최단경로 T에 있는 링크와 연결한다. 총 비용의 계산은 최초의 최단경로비용에 가중치가 더해진 sidetrack 값을 합하면 된다.

【Step 3】 queue Q에 있는 sidetrack이 모두 산출되면, 종료.

이상 Step 3.2의 K최단경로 산출에 소요되는 시간은 $O(k\log k)$가 되며, Eppstein 알고리즘의 총 수행시간은 $O(m+n\log n+k\log k)$가 된다. 〈표 1〉은 네트워크 G에서 출발지 노드는 ⓪, 도착지 노드가 ⑪로 주어졌을 때, 산출되는 결과이다.

〈표 1〉 Eppstein 알고리즘의 수행결과

K	경로(노드순서)	비용
1	0 -> 4 -> 5 -> 6 -> 10 -> 11	55
2	0 -> 5 -> 6 -> 10 -> 11	57
3	0 -> 1 -> 2 -> 3 -> 7 -> 11	58
4	0 -> 1 -> 2 -> 6 -> 10 -> 11	59
5	0 -> 4 -> 5 -> 9 -> 10 -> 11	61
6	0 -> 1 -> 5 -> 6 -> 10 -> 11	62
7	0 -> 5 -> 9 -> 10 -> 11	63
8	0 -> 4 -> 5 -> 6 -> 7 -> 11	64
9	0 -> 4 -> 8 -> 9 -> 10 -> 11	65
10	0 -> 5 -> 6 -> 7 -> 11	66
11	0 -> 6 -> 10 -> 11	68
12	0 -> 1 -> 2 -> 6 -> 7 -> 11	68
13	0 -> 1 -> 5 -> 9 -> 10 -> 11	68
14	0 -> 1 -> 5 -> 6 -> 7 -> 11	71
15	0 -> 6 -> 7 -> 11	77

3) Lazy Version of Eppstein's Algorithm(LVEA)

Eppstein 알고리즘은 K최단경로를 산출하기 위해 최단경로가지를 계산한 후 최단경로로부터 도출될 수 있는 모든 우회경로(K최단경로)를 나타내기 위해 방향성 그래프 $D(G)$를 구축하며, $D(G)$를 구축하는데 까지 걸리는 시간은 $O(m+n\log n)$이다.

일단 $D(G)$가 구축되면 K최단경로가 산출되는데, 경로 산출에 걸리는 시간은 $O(k\log k)$의 시간증가에 따라 이루어진다. 그렇지만, 노드와 링크의 수가 많은 대규모의 네트워크에서는 $D(G)$의 구축에 소요되는 시간이 무시할 수 없을 만큼 많이 걸리게 되는 단점을 지니고 있다(Jiménez et al, 2003).

Jiménez et al(2003)은 Eppstein 알고리즘의 수행속도를 더욱 개선하기 위해 K최단경로 선택을 위해 필요한 $D(G)$의 일부만을 구축하여 K최단경로를 산출하는 보다 수정된 알고리즘인 『A Lazy Version of Eppstein's K Shortest Paths Algorithm(LVEA)』을 제안하였다.

가. 기존 Eppstein 알고리즘과의 차이

K최단경로의 산출을 위해서, 기존의 Eppstein 알고리즘은 도출될 수 있는 모든 경로의 정보를 지니고 있는 방향성 그래프 $D(G)$를 Step 3에서 생성한다. 하지만, LVEA는 초기 입력값으로 K를 입력하여 K가 증가 즉, 호출(Call)할 때 마다 필요한 만큼의 $D(G)$를 구축하여 (K)개의 최단경로를 산출하는 방안이다.

K=1은 기존의 Eppstein 알고리즘과 같이 산출이 되지만, 이후 K=2~k 동안은 기존 알고리즘과 차이점을 가지게 된다.

K=2~k인 동안의 알고리즘 수행과정은 다음과 같다. 특히, K=2일때 호출되는 최초의 $D(G)$는 『$s\rightarrow t$ 최단경로에 구축된 출발지 노드번호 배열에 구성된 힙인 H_G가 있을 경우, 출발지 노드 s에 구성된 힙 $H_G(s)$의 호출로서 형성되는 $D(G)$이고, 그렇지 않으면 $s\rightarrow t$ 최단경로 상에서 구성된 힙인 H_G들 중 출발지 노드 s와 가장 가까운 노드번호 배열에 구성된 힙 $H_G(v)$를 호출함으로서 형성되는 $D(G)$』이다.

【Step 1】 만약 K=2이면 〔Step 1.1〕실행, 그렇지 않으면 〔Step 1.2〕 실행

【Step 1.1】 최초의 $D(G)$로부터 Sidetrack을 queue Q에 저장; 최초의 $D(G)$내에서 가장 작은 (δ + 가중치)값을 가지는 sidetrack(결국, K=2일 때는 $H_G(v)$의 근노드)을 복사하여 heap 영역내의 queue Q에 저장(이때 sidetrack은 ($v\rightarrow w$, δ + 가중치)의 정보를 가지며, 가중치가 더해진 sidetrack은 가중치를 넘겨준 sidetrack들에 대한 정보(꼬리노드 및 머리노드)를 함께 가진 채로 저장). 만약, Q에 저장될 sidetrack이 없으면 프로그램 종료.

【Step 1.2】 $D(G)$로부터 Sidetrack을 queue Q에 저장; 생성된 $D(G)$내에서, 경로산출을 위해 queue Q에서 추출된 sidetrack과 동일한 sidetrack(여기서의 sidetrack은 $H_G(v)$내에 있는 sidetrack을 의미)의 자식노드(최대 3개; $H_T(v)$로부터 2, $H_{out}(v)$로부터 1)와 연결된 $H_G(w)$내의 가장 작은 (δ + 가중치)값을 가지는 sidetrack을 복사하여 queue Q에 저장(이외의 저장되는 sidetrack의 정보는 〔Step 1.1〕과 동일).

【Step 2】 Sidetrack 정렬; queue Q내에 저장된 sidetrack들을 (δ + 가중치)값의 크기에 따라 정렬(즉, K-1 단계의 〔Step 3〕 후 남아있는 sidetrack들이 있으면 그 sidetrack들과 함께 정렬됨을 의미).

【Step 3】 Sidetrack 추출; 만약, queue Q에 sidetrack이 없으면 프로그램 종료. 그렇지 않으면, 정렬된 Q내의 sidetrack 중 가장 작은 값의 sidetrack을 추출.

【Step 4】 Sidetrack과 최단경로가지 T의 연결(K예약최단경로); 추출된 sidetrack ($v\rightarrow w$, δ + 가중치)과 출발지 노드 $s\rightarrow v$ 최단경로와 $w\rightarrow t$ 최단경로를 연결(이때, 추출된 sidetrack이 가중치가 더해진 값을 가지는 sidetrack이면, 꼬리노드 v와 연결되는 출발지 노드 $s\rightarrow v$의 최단경로에는 가중치를 넘겨준 이전의 sidetrack들이 포함된 최단경로임).

【Step 5】 K최단경로 산출 및 총비용; 총비용의 계산은 K=1의 비용에 추출된 sidetrack의 (δ + 가중치)값.

【Step 6】 새로운 $D(G)$의 생성: 경로 산출을 위해 Q에서 추출된 sidetrack과 같은, $H_G(v)$ 내부에 있는 sidetrack은 머리노드 번호(w)의 배열에 구성된 힙 $H_G(w)$를 Pointing한 후 K=K+1과 함께 go to 〔Step 1〕(여기서, Pointing 할 때에는 기존의 Eppstein 알고리즘과 같이 sidetrack이 가진 가중치를 $H_G(w)$ 내에 있는 sidetrack들에 더해주게 되며, 이로써 전체 $D(G)$의 일부 $D(G)$가 생성). 만약, 연결할 $H_G(w)$가 존재하지 않으면 K=K+1과 함께 go to 〔Step 1〕.

나. 속도 추정

K가 증가함에 따라, LVEA는 노드집합 V의 모든 노드 v에 대하여 $H_{out}(v)$ 와 $H_G(v)$ 를 산출하기 때문에 결국, 최악의 경우(worst-case)의 시간복잡도는 $O(m+n\log n)$으로서 Eppstein 알고리즘과 동일하게 된다. 하지만, 실제 실험에서의 결과에 따르면, 절약되는 시간의 정도는 매우 큰 차이를 보이는 것으로 나타났다(Jiménez et al, 2003).

Ⅲ. 알고리즘 개선방안

본 연구의 목적은 K최단경로 탐색 시 수행속도 측면에서는 기존의 알고리즘과는 비교할 수 없을 정도로 월등한 Ordered Heap Tree를 기반으로 한 알고리즘인 Eppstein 알고리즘과 Eppstein 알고리즘의 속도를 보다 더 향상시키기 위해 Eppstein 알고리즘을 수정한 Jiménez et al(2003)이 제안한 Lazy Version of Eppstein's Algorithm(LVEA)을 교통망에 적용 시 문제가 될 수 있는 알고리즘의 문제점을 파악하여, LVEA를 현실 교통망에의 적용이 가능하도록 알고리즘을 개선하는 방안을 제안하도록 한다.

1. 현실교통망과 링크 비루프 경로

네트워크에서 루프는 노드와 링크의 반복이 제한 없이 허용되는 개념이다. 루프를 교차로와 가로로 구성된 현실 통행으로 확대하면 루프형 통행은 교차로와 가로를 무제한 반복해서 통과하는 개념으로 정의된다. 일반적으로 현실(특히 도시) 교통망에서 교차로를 반복해서 방문하는 통행은 존재하나, 가로를 반복해서 주행하는 현상은 존재하지 않는다. 즉, 교통망에서의 루프형 통행은 링크의 반복이 허용되지 않는 링크 비루프(Link Loopless Path) 통행으로 축소된다(신성일, 2004). 따라서 본 연구에서는 알고리즘의 개선에 있어서 링크 비루프의 적용을 고려하도록 한다.

2. 알고리즘의 문제점

1) Eppstein 알고리즘의 문제

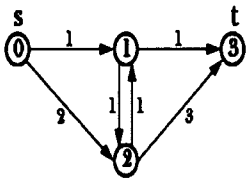
Eppstein 알고리즘은 K최단경로를 산출하기 이전에 (K)개의 최단경로 정보를 모두 가지고 있는 방향성 그래프 $D(G)$를 생성한다.

제2절에서의 예시되었던 네트워크 G에서와 같이 사이클(cycle)이 존재하지 않으면 문제가 되지 않지만, cycle이 있을 경우에는 방향성 그래프 $D(G)$의 생성이 무한대가 되어버리는 문제점을 내제하고 있다. 그렇기 때문에, $D(G)$가 무한으로 생성되면 K최단경로를 산출조차 할 수 없게 되어버린다. 하지만, LVEA에서는 이 문제에 대한 제어방안이 마련되어있다.

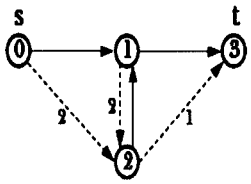
2) LVEA의 문제

Jiménez et al(2003)의 LVEA는 (K)의 호출(Call)이 있을 때 마다 필요한 만큼의 $D(G)$를 구축하여 (K)개의 최단경로를 산출하는 방안이므로, $D(G)$의 무한생성에 대한 제어는 가능하다. 하지만, 네트워크 G에 cycle이 있는 경우, 즉 간단히 설명하자면 노드와 노드사이에 양방향 링크가 존재하는 경우, $D(G)$의 생성인 힙 $H_G(v)$와 $H_G(w)$의 연결에서 $H_G(v)$와 $H_G(w)$ 내에 동일한 sidetrack이 존재하게 되어 그 동일한 sidetrack이 서로 연결되어 경로가 생성되므로 링크루프(Link Loop, Link Repeated Path)가 발생하게 되는 문제점과 중복되지 않는 sidetrack들 간을 연결하게 되더라도 K경로 산출 시 sidetrack과 최단경로 링크가 연결될 때 최단경로 링크의 중복이 발생함으로 인해 링크루프가 발생하게 되는 2가지 경우에 있어서의 링크루프 발생 문제점을 가지고 있다. 이로 인해 LVEA는 K가 커지게 되면 동일한 링크루프를 가지는 경로를 무한히 산출하게 되는 한계를 보여주게 된다.

LVEA의 문제점인 링크루프의 발생과 그것으로 인한 무한 K최단경로의 산출에 대해 설명하기 위해 〈그림 14〉와 같이 K최단경로 탐색 시 루프가 필연적으로 발생하게 되는 네트워크 G의 경우를 예로 들어 LVEA를 실행하도록 한다.

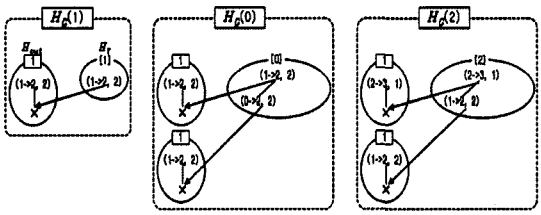


〈그림 14〉 네트워크 G



〈그림 15〉 최단경로가지 T 및 sidtracks

〈그림 14〉의 네트워크 G로부터 〈그림 15〉와 같이 최단경로 가지(Shortest paths tree) T와 각 노드로부터의 sidetrack들이 산출된다(Step 1). 이후 도착지 노드(t)로부터 각 노드로의 깊이우선탐색(DFS)이 실행되며, 각각의 힙 $H_{out}(v)$, $H_T(v)$, 그리고 〈그림 16〉과 같이 $H_G(v)$가 구축된다(Step 2).



〈그림 16〉 각 노드별로 구축된 $H_G(v)$

$H_G(v)$의 구축이 완료되면, 다음 단계로 K의 호출(Call)에 따라 $H_G(v)$와 $H_G(w)$의 연결인 $D(G)$가 생성이 되면서 경로를 산출하게 된다. 이 과정에서 $D(G)$가 생성이 되면서 연결되는 sidetrack들의 중복이 나타나게 된다. 링크루프의 발생원인은 첫째, sidetrack의 중복이 발생함으로 인해 K경로 산출 시 링크루프가 나타나게 되는 것과 둘째, sidetrack의 중복이 없더라도 경로산출 시 K=7의 경로와 같이 sidetrack인 (0->2, 2)와 (1->2, 2)의 연결 시에 최단경로·링크의 중복이 발생함으로써 링크루프가 발생하게 되는 두 가지 경우가 있음을 알 수 있다.

양방향 링크가 존재하는 네트워크 G에서 LVEA에 의한 수행결과는 〈표 2〉과 같이 링크루프가 존재하게 된다.

〈표 2〉 LVEA의 수행결과

K	경로(노드순서)	비용
1	0→1→3	2
2	0→1→2→1→3	4
3	0→2→1→3	4
4	0→1→2→3	5
5	0→2→3	5
6	0→1→2→1→2→1→3	6
7	0→2→1→2→1→3	6
8	0→1→2→1→2→3	7
9	0→2→1→2→3	7
10	0→1→2→1→2→1→2→1→3	8
11	0→2→1→2→1→2→1→3	8
12	0→1→2→1→2→1→2→3	9
13	0→2→1→2→1→2→3	9
14	0→1→2→1→2→1→2→1→2→1→3	10
15	0→2→1→2→1→2→1→2→1→3	10
16	0→1→2→1→2→1→2→1→2→3	11
17	0→2→1→2→1→2→1→2→3	11
18	0→1→2→1→2→1→2→1→2→1→2→1→3	12
19	0→2→1→2→1→2→1→2→1→2→1→3	12
20	0→1→2→1→2→1→2→1→2→1→2→3	13
...	...	...

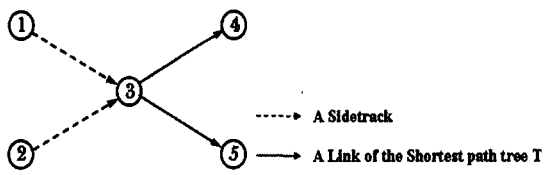
주 : 밑줄은 링크루프가 존재하는 경로

3) 개선 방안

링크루프의 발생 원인에 대해서 간략히 요약한 후 그 2가지 경우에서의 제어방안을 제시하도록 한다.

(1) 링크루프의 발생 원인

링크루프의 발생원인은 〈그림 17〉을 예로 들어 설명할 수 있다.



〈그림 17〉 링크루프의 발생원인

가. 동일한 sidetrack의 중복

링크루프의 첫 번째 원인으로는 K의 호출(Call)로 인한 새로운 $D(G)$의 생성으로 $H_G(v)$와 $H_G(w)$가 연결되고, 그 후에 K최단경로의 산출과정에서 $H_G(v)$내

의 sidetrack과 동일한 $H_G(w)$ 내의 sidetrack이 연결됨으로써 K경로를 산출하게 되는, 동일한 sidetrack의 중복이 그 첫 번째 원인이다. 이는 sidetrack (①, ③)과 최단경로 링크(③, ④)가 연결되는 경로에서 다시 한번sidetrack (①, ③)이 연결된 후 최단경로 링크(③, ⑤)가 연결되어 산출되는 최단경로와 같다.

나. 동일한 최단경로의 중복

두 번째 원인은 비록 sidetrack은 중복되지 않더라도 K경로 산출 시 sidetrack과 연결되는 최단경로 링크가 중복이 되는 경우이다. 예를 들면, sidetrack(①, ③)과 최단경로 링크(③, ⑤)가 연결되는 경로에서 sidetrack (②, ③)이 연결된 후 최단경로 링크(③, ⑤)가 중복되어 산출되는 최단경로로서 설명될 수 있다.

(2) Link Loop의 제어방안

Eppstein 알고리즘과 이 알고리즘을 수정한 LVEA는 sidetrack들의 힙을 이용한 구조이기 때문에 전체 수행속도에 큰 영향을 미치지 않고서도 이러한 루프를 제거 할 수 있는 장점을 지니고 있다.

첫 번째 동일한 sidetrack의 중복을 제어하는 방안은 $H_G(v)$ 내의 sidetrack이 $H_G(w)$의 sidetrack을 연결하여 비교할 때 만약, $H_G(w)$의 sidetrack이 동일할 경우 Pointing을 하지 않도록 제어하는 방안이다. 이 방법으로 제어하게 되면 그 이후에 연결되는 $H_G(x)$와의 연결도 끊어져 더 이상 그 sidetrack과 중복되어 연결되는 $D(G)$는 생성되지 않으므로 수행속도의 향상도 가져오게 된다.

두 번째 경우의 링크루프 발생의 제어는 K번째 최단경로의 산출 시 중복되는 최단경로 링크의 존재 유무를 검색하여, 존재할 경우 그 경로는 산출되지 않고 건너뛰도록(Pass)하는 방안을 사용하였다.

4) 속도 추정

본 연구에서 제안된 Heap Ordered Tree 구축을 중심으로 한 링크 비루프 알고리즘의 경우, K 경로 산출 시 단지 힙들 간의 연결을 제어한 것이므로 결국, 기존의 Eppstein 알고리즘과 마찬가지로 $H_{out}(v)$가 구성되는데 소요되는 시간 $O(m)$과 각각의 노드에 대한 $H_G(v)$의 구축 시 걸리는 $O(n\log n)$, K 산출시 소요되는 시간이 $O(k\log k)$로서 모든 경로탐색에 소요되는 총 시간은 $O(m+n\log n+k\log k)$가 된다.

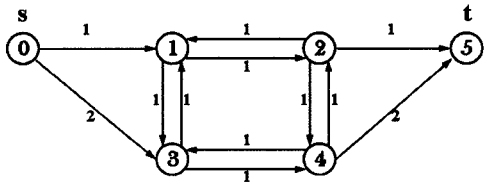
하지만, 본 연구에서 제안된 알고리즘은 Lazy Version of Eppstein's Algorithm에 기반을 두고 있기 때문에, 원하는 (K)값에 따라 $D(G)$의 생성이 이루어지게 되므로 K가 모든 경로탐색에 요구되는 값이 아니라면 수행속도는 Eppstein 알고리즘의 수행속도인 $O(m+n\log n+k\log k)$ 보다 월등히 빠를 수밖에 없다.

또한 본 연구에서 비루프 경로(No Link Repeated Path) 알고리즘은 첫 번째 경우의 링크루프 제어로 인한 속도의 향상과 두 번째 경우의 제어 방안인 건너뛰기(Pass)로 말미암아 기존의 LVEA와 비교하여 수행속도의 저하는 거의 없음을 알 수 있다.

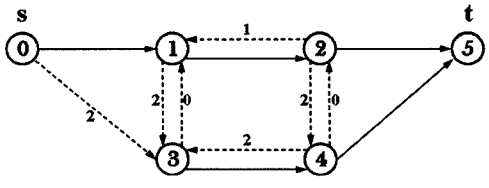
Ⅳ. 사례연구

본 연구에서 제안된 Jiménez et al(2003)의 Lazy Version of Eppstein's Algorithm 기반의 링크 비루프 K최단경로 탐색 알고리즘의 평가를 위해 우선 모의 네트워크인 〈그림 18〉에 적용함으로서 〈표 3〉과 같이 링크루프가 제어된 결과를 얻을 수 있었다. 이때 K=1~55로 설정하였다.

본 연구에서 제안한 알고리즘에 의한 K최단경로 결과, LVEA의 결과에 나타난 링크루프가 존재하는 경로들이 모두 삭제되었음을 나타내고 있으며, 링크루프가 없는 K최단경로는 모두 50개로 도출되었다. 이에 반해 LVEA의 경우, K의 값이 무한대에 가까이 수렴되면 K최단경로 역시 루프경로가 되어 무한한 값으로 도출되는 결과를 보인다.



〈그림 18〉 모의 가로망 G



〈그림 19〉 최단경로가지 및 sidetracks

〈표 3〉 LVEA과 제안된 알고리즘의 수행결과 비교

Lazy Version of Eppstein's Algorithm			K 링크 비루프 최단경로 알고리즘		
K	경로(노드순서)	비용	K	경로(노드순서)	비용
1	0→1→2→5	3	1	0→1→2→5	3
2	0→1→2→1→2→5	5	2	0→1→3→4→5	5
3	0→1→3→4→5	5	3	0→3→4→5	5
4	0→3→4→5	5	4	0→1→3→4→2→5	5
5	0→1→3→4→2→5	5	5	0→3→4→2→5	5
6	0→3→4→2→5	5	6	0→1→3→1→2→5	5
7	0→1→3→1→2→5	5	7	0→1→2→4→5	5
8	0→1→2→4→5	5	8	0→3→1→2→5	5
9	0→3→1→2→5	5	9	0→1→2→4→2→5	5
10	0→1→2→4→2→5	5	10	0→1→3→4→2→1→2→5	7
11	0→3→1→2→1→2→5	7	11	0→1→2→4→3→4→5	7
12	0→1→2→4→2→1→2→5	7	12	0→1→2→4→3→4→2→5	7
13	0→1→2→1→2→1→2→5	7	13	0→3→1→3→4→5	7
14	0→1→3→4→2→1→2→5	7	14	0→3→1→2→4→5	7
15	0→1→2→4→3→4→5	7	15	0→3→1→3→4→2→5	7
16	0→3→1→3→4→5	7	16	0→3→1→2→4→2→5	7
17	0→1→2→4→3→4→2→5	7	17	0→1→3→4→3→1→2→5	7
18	0→3→1→3→4→2→5	7	18	0→3→4→2→1→2→5	7
19	0→1→3→4→3→4→5	7	19	0→1→3→1→2→4→5	7
20	0→3→1→2→4→5	7	20	0→3→4→2→4→5	7
21	0→3→1→3→1→2→5	7	21	0→1→3→1→2→4→2→5	7
22	0→1→3→4→3→4→2→5	7	22	0→1→2→1→3→4→5	7
23	0→3→1→2→4→2→5	7	23	0→1→3→4→2→4→5	7
24	0→1→2→4→2→4→5	7	24	0→1→2→1→3→4→2→5	7
25	0→1→3→4→3→1→2→5	7	25	0→3→4→3→1→2→5	7
26	0→1→2→4→2→4→2→5	7	26	0→1→2→1→3→4→2→4→5	9
27	0→1→3→1→2→1→2→5	7	27	0→3→1→2→1→3→4→5	9
28	0→3→4→2→1→2→5	7	28	0→3→1→3→4→2→1→2→5	9
29	0→1→3→1→3→4→5	7	29	0→1→3→4→2→1→2→4→5	9
30	0→1→2→1→3→4→5	7	30	0→3→1→2→1→3→4→2→5	9
31	0→1→2→1→2→4→5	7	31	0→3→1→3→4→2→4→5	9
32	0→1→2→1→3→4→2→5	7	32	0→1→2→4→2→1→3→4→5	9
33	0→1→2→1→2→4→2→5	7	33	0→3→1→2→4→3→4→5	9
34	0→1→3→1→2→4→5	7	34	0→3→1→2→4→3→4→2→5	9
35	0→3→4→2→4→5	7	35	0→1→3→1→2→4→3→4→5	9
36	0→1→3→1→2→4→2→5	7	36	0→1→3→1→2→4→3→4→2→5	9
37	0→3→4→2→4→2→5	7	37	0→3→4→2→1→2→4→5	9
38	0→1→3→1→3→4→2→5	7	38	0→3→4→2→1→3→1→2→5	9
39	0→1→2→1→3→1→2→5	7	39	0→1→2→4→3→1→3→4→5	9
40	0→3→4→3→4→5	7	40	0→1→3→4→3→1→2→4→5	9
41	0→1→3→4→2→4→5	7	41	0→1→2→4→3→1→3→4→2→5	9
42	0→3→4→3→4→2→5	7	42	0→1→3→4→3→1→2→4→2→5	9
43	0→1→3→4→2→4→2→5	7	43	0→1→3→4→2→4→3→1→2→5	9
44	0→3→4→3→1→2→5	7	44	0→3→4→3→1→2→4→5	9
45	0→1→2→4→3→1→2→5	7	45	0→3→4→3→1→2→4→2→5	9
46	0→1→3→1→3→1→2→5	7	46	0→3→4→2→4→3→1→2→5	9
47	0→1→2→4→3→1→2→1→2→5	9	47	0→3→1→3→4→2→1→2→4→5	11
48	0→1→3→1→3→1→2→1→2→5	9	48	0→3→1→2→1→3→4→2→4→5	11
49	0→3→4→3→4→2→1→2→5	9	49	0→3→1→2→4→2→1→3→4→5	11
50	0→1→2→1→2→1→2→1→2→5	9	50	0→3→4→2→1→3→1→2→4→5	11

주 : 밑줄은 링크루프가 존재하는 경로

Lazy Version of Eppstein's Algorithm			K 링크 비루프 최단경로 알고리즘		
K	경로(노드순서)	비용	K	경로(노드순서)	비용
51	0→1→2→4→3→4→3→4→5	9	51	-	-
52	0→1→3→1→2→1→2→1→2→5	9	52	-	-
53	0→1→2→4→3→4→3→4→2→5	9	53	-	-
54	0→1→3→1→3→4→3→4→5	9	54	-	-
55	0→1→3→4→3→4→2→1→2→5	9	55	-	-

주 : 밑줄은 링크루프가 존재하는 경로

V. 결론 및 향후과제

1. 결론

첫째, 본 연구는 기존의 경로삭제방식 기반의 알고리즘에 비해 월등한 수행속도를 가지는 알고리즘인 Eppstein(1998)이 제안한 Heap Ordered Tree 기반의 K 최단경로 탐색 알고리즘과 Eppstein 알고리즘을 보완한 Jiménez et al(2003)의 Lazy Version of Eppstein's Algorithm(LVEA)을 분석하여 LVEA를 실제 C Programming 하였다.

둘째, K 경로탐색에 있어 그 수행속도는 월등하나 Computer Science 분야를 중심으로 연구되어 교통분야 적용에 생소한 기법인 Heap Ordered Tree를 기반의 알고리즘을 소개하고 또한 보완하여 현실 교통망에 반영이 가능할 수 있도록 경로의 산출 시 링크루프(No Link Repeated Path)를 제어하는 방안을 마련하여 교통분야의 적용에 있어 보다 현실적인 알고리즘을 개발하였다.

셋째, 결과적으로 기존의 K경로탐색 알고리즘에 비해 수행속도 측면에서 향상이 되었으며, 본 알고리즘의 기반이 되는 LVEA의 수행속도에 비해서도 거의 저하가 없는 방안으로 링크루프제어가 가능하게 되었음에 그 의미를 부여할 수 있다.

2. 향후 연구과제

이상의 연구에서 향후에 추진되어야 할 과제로서는 첫째, 모의 네트워크가 아닌 서울시와 같은 대규모 네트워크에서의 적용에서도 문제점의 발생이 나타나지 않는가에 대한 검증이다.

둘째, 본 연구에서는 기존의 알고리즘을 보다 현실적인 교통망으로의 적용을 위해 일차적으로 링크루프를 제어하는 방안만을 제시하였지만, 활용가치를 더욱 높이기 위해서는 본 연구에서 개선한 알고리즘을 보완하여 노드(즉 교차로)에서의 비용과 좌회전 금지나 U-turn 금지와 같은 회전 페널티를 반영할 수 있는 알고리즘의 개발이 필요하다.

셋째, 최근 이슈가 되고 있는 복합교통망(Intermodal)에서의 활용을 위하여 단일수단의 최적조건에서 복합수단의 최적조건으로 구축되도록 하는 네트워크 표현과 알고리즘의 구축에 대한 추가적인 연구가 필요하다고 할 수 있다.

참고문헌

1. 임강원 · 임용택(2003), 「교통망 분석론」, 서울대학교 출판부.
2. 신성일(2004), 「교통망에 적합한 K 비루프 경로 탐색 알고리즘」, 대한교통학회지, 제22권 제6호, 대한교통학회, pp.121~131.
3. Azevedo J. A., Costa M. E. O. S., Madeira J.J.E.R.S., and Martins E.Q.V(1993), 「An algorithm from the ranking of shortest paths」, European Journal of Operational Research, Vol.69, pp.97~106.
4. Dijkstra E. W.(1959), 「A note of two problems in connected with graphs」, Numerical Mathematics. I, pp.269~271.
5. Eppstein D.(1998), 「Finding the k shortest paths」, SIAM J. Computing, Vol. 28, No. 2, pp.652~673.
6. Jiménez V. M. and Marzal A.(2003), 「A lazy version of Eppstein's k shortest pahts algorithm」, WEA 2003, LNCS 2647, pp. 179~191.
7. Martins E.Q.V(1984), 「An algorithm for ranking paths that may contain cycles」, European Journal of Operational Research, Vol.18, pp.123~130.

8. Moore E. F.(1957) The Shortest Path through A Maze, Proc. Int. Conf. on the Theory of Switching, Harvard Univ., Cambridge, MA.
9. Shier R. D.(1979) On Algorithms from Finding the K Shortest Paths in A Networks, Vol. 9, pp.195~214.
10. Yen J.Y.(1971), 「Finding the K shortest loopless paths in a network」, Management Science, Vol.17, pp.711~715.

♧ 주 작 성 자 : 임강원
♧ 교 신 저 자 : 양승묵
♧ 논문투고일 : 2005. 10. 29
논문심사일 : 2005. 11. 30 (1차)
2005. 12. 15 (2차)
심사판정일 : 2005. 12. 15
♧ 반론접수기한 : 2006. 4. 30