Journal of the Korean Institute of Industrial Engineers
Vol. 32, No. 3, pp. 226-235, September 2006.

복수 경로 탐색을 위한 휴리스틱 알고리즘에 대한 연구

신용욱[1] · 양태용[1†] · 백원장[2]

[1]한국과학기술원 산업공학과 / [2]코리아 텔레매틱스

Heuristic Algorithm for Searching Multiple Paths

Yongwook Shin[1] · Taeyong Yang[1] · Won Baek[2]

[1]Department of Industrial Engineering, KAIST
[2]Korea Telematics Co.

Telematics is expected to be one of the fastest growing businesses in information technology area. It may create a new emerging market in industry related to automotive, telecommunications, and information services. Especially vehicle navigation service is considered as a killer application among telematics service applications. The current vehicle navigation service typically recommends a single path that is based on the traveling time or distance from the origin to the destination. The system provides two options for users to choose either via highway or via any road. Since the traffics and road conditions of big cities are very complicated and dynamic, the demand of multi-path guidance system is increasing in telematics market. The multi-path guidance system should allow drivers to choose a path based on their individual preferences such as traveling time, distance, or route familiarity. Using the Lawler's algorithm, it is possible to find multiple paths; however, due to the lengthy computational time, it is not suitable for the real-time services. This study suggests a computationally feasible and efficient heuristic multiple paths finding algorithm that is reliable for the real-time vehicle navigation services.

***Keywords*:** K-Shortest Paths, Multiple Paths, Vehicle Navigation Service, Telematics

1. 서 론

텔레매틱스(Telematics)는 텔레커뮤니케이션(Telecommunication)과 인포매틱스(Informatics)의 합성어로, 자동차 안의 단말기를 통해서 자동차와 운전자에게 다양한 종류의 정보 서비스를 제공해 주는 것을 의미한다(Song, 2004). 운전자나 탑승자에게 무선 네트워크와 인공위성 위치 정보 시스템(GPS : Global Positioning System)을 이용하여 자동차 안에서 교통정보, 원격 차량진단, 모바일 전자상거래 등의 각종 정보 서비스를 제공할 수 있다. 텔레매틱스 산업이 정보 산업 전반에 미치는 파급효과를 고려하여 대한민국 정부에서도IT 9대 신성장동력 중의 하나로 텔레매틱스를 선택하였다. 이렇듯 시장으로부터 적지 않은 관심을 받고 있으나, 비교적 발전된 네트워크 기반의 기술적인 측면과는 달리 아직 서비스의 질과 양적인 측면에서 소비자들에게 강한 매력을 주지는 못하고 있다. 만성적인 교통 혼잡에서 많은 직간접적 비용을 지불하고 있는 운전자에게 경로 안내 서비스는 텔레매틱스 서비스 중 가장 좋은 반응을 얻고 있으며, 이에 따라 많은 서비스 제공 업체들이 이와 관련된 많은 상품을 시장에 출시하고 있다. 초창기 경로 안내 서비스는 정적 정보 기반의 최단 거리 경로 안내 서비스를 제공하였으나, 최근에는 교통 정보 시스템의 발달로 실시간 동적 정보 기반의 최단 시간 경로 안내 서비스도 활발히 제공되고 있다.

경로 안내 서비스는 운전자가 출발지와 목적지를 입력하면, 서비스 제공 업체는 해당 영역내 노드간의 소요 시간 또는 거리 정보를 기반으로 해서 최단 경로 탐색 방법을 이용하여 최단 경로를 구하여 이를 다시 운전자에게 제공하는 절차로 이루어진다. 최단 경로 탐색 방법으로는 다익스트라 알고리즘(Dijkstra's

† 연락저자 : 양태용 교수, 305-701 대전광역시 유성구 구성동 373-1 한국과학기술원 산업공학과, Tel : 042-869-8358, Fax : 042-869-8357, E-mail : tyang@kaist.ac.kr
2005년 8월 접수; 2005년 12월 수정본 접수; 2006년 6월 게재 확정.

algorithm)과 양방향 다익스트라 알고리즘(Bidirectional Dijkstra's algorithm), A* search 알고리즘, ALT(A* search, Landmarks and the Triangle inequality) 알고리즘 등이 이용되고 있다(Goldberg *et al.*, 2005).

현재 경로 안내 서비스의 대부분은 출발지에서 목적지까지 하나의 경로만을 안내해 주는 단일 최단 경로 안내 서비스만이 제공되고 있다. 예외적으로 고속도로나 간선도로 등의 자동차 전용도로를 우선으로 이용하는 경로 안내 서비스와 비전용도로를 이용한 경로 안내 서비스 등2가지 옵션을 갖는 서비스를 제공하는 경우도 있기는 하나, 이 또한 단일 최단 경로 안내 서비스의 범주라고 할 수 있다. 그러나 최단 거리 경로 하나만을 운전자에게 안내 해 주는 Push 타입의 단일 경로 안내 서비스는 다양한 운전자들의 요구 사항을 만족시키기에 한계를 가지고 있다. 예를 들어, 출발지에서 목적지까지의 최단 경로를 고려했을 때, 안내된 최단 경로가 30분이 소요되고, 운전자의 친숙도와 도로 포장 상태가 안 좋은 경우와 소요시간이 31분이고 소요되고 친숙도가 높고 도로 포장 상태가 양호한 2번째 최단 경로에 대한 정보를 제공하여 운전자로 하여금 다양한 경로를 선택하게 한다면 높은 만족도를 표시하는 운전자들이 많을 것이다. 이렇듯 단 하나의 최단 경로만을 안내하는 Push 타입의 단일 경로 안내 서비스보다는 다수의 대안 경로들을 안내하여 운전자에게 다양한 선택권을 부여하는 Pull 타입의 복수 경로 안내 서비스의 필요성은 매우 높다고 할 수 있다.

복수 경로 안내 서비스 구현을 위해서 사용될 수 있는 복수 최단 경로(K Shortest Paths : 이하 KSP) 알고리즘들은 이미 많이 제시되어 활발히 연구되고 있다. 그러나 실시간 서비스인 차량 경로 안내 서비스에 적용하기에는 알고리즘 계산 시간이 길다는 치명적인 약점을 가지고 있다. 수백개 단위의 노드로 구성된 네트워크에서는 기존의 KSP 알고리즘의 적용이 가능하나, 대도시의 예로 보면 현재 실제 네트워크는 적게는 수천개, 많게는 수만개의 노드로 구성되어 있기 때문에, 기존의 KSP 알고리즘을 적용하기에는 한계를 지니고 있다. 이에 본 논문에서는 실시간 복수 경로 안내 서비스 구현에 기여하기 위해, 기존 KSP 알고리즘 보다 계산 속도가 빠른 휴리스틱 복수 경로 탐색 알고리즘 개발에 대해 역점을 두었다.

2장과 3장에서는 복수 경로 발생을 위한 기존의 KSP 알고리즘들과 본 논문에서 제시하고자 하는 새로운 알고리즘에 사용되는 기존의 알고리즘들에 대해 살펴보도록 하겠다. 4장에서는 양방향 다익스트라 알고리즘에 의해 선택된 경유 노드(Through-Node)를 이용하여 계산 속도를 향상시킨 새로운 복수 경로 탐색 알고리즘인, Through-Node Multiple Paths Search(이하 TMPS) 알고리즘에 대해 설명한다. 마지막으로 실험을 통해 본 연구에서 제안한 TMPS 알고리즘과 Lawler 알고리즘에 의해 발견된 각각의 경로들간의 일치도와 소요된 계산 시간을 바탕으로 두 알고리즘을 비교, 분석하도록 한다.

2. K-Shortest Paths(KSP) 알고리즘

기존의 KSP 알고리즘은 경로들이 반복되는 링크를 허용하는 알고리즘과 반복되는 링크를 허용하지 않는 알고리즘으로 분류될 수 있다. Hoffman and Pavley(1959), Bellman and Kalaba (1960), Shier(1976, 1979), Epstein(1994, 1998) 등은 전자에 속하는 알고리즘을 개발하였고, Yen(1971), Lawler(1976), Katoh (1982), Hadjiconstantnou and Christofides(1999) 등은 후자에 속하는 알고리즘을 개발하였다. Katoh(1982), Hadjiconstantnou and Christofides(1999)의 알고리즘은 방향성이 없는 네트워크상에서 작동한다(Zijpp *et al.*, 2005). 경로 안내 서비스 문제에서 반복 경로와 방향성이 없는 네트워크는 고려 대상에서 제외된다. Brander and Sinclair(1995)가 1950년 이후에 제시된 KSP 알고리즘과 그와 관련된 70여개의 논문으로부터 계산 속도의 기대값(Expected speed of operation)을 기준으로 4개의 알고리즘을 선택하여 비교 분석한 결과, 방향성이 있는 네트워크상에서 비순환 경로만을 고려하는 KSP 알고리즘 중에서 Lawler 알고리즘이 가장 좋은 성능을 나타내었다. Fox(1978)가 만든 복잡도(Complexity) 분석에 따르면 Lawler 알고리즘은 복잡도가 $O(kn^3)$으로 4개의 알고리즘 중 가장 적었다(k : 찾고자 하는 경로의 개수, n : 네트워크상의 노드 개수). 이러한 이유로 Lawler 알고리즘은 실시간 복수 경로 안내 서비스를 위해 사용 가능한 복수 경로 탐색 알고리즘 개발을 위한 본 논문의 출발점으로 선택되었다(Zijpp *et al.*, 2005).

3. TMPS 알고리즘에 사용되는 알고리즘들

본 장에서는 본 연구에서 제시하고자 하는 새로운 알고리즘인 TMPS 알고리즘에서 이용되는 기존 알고리즘들과 경유 노드의 개념과 역할에 대해서 살펴보고자 한다.

3.1 경유 노드 정의

본 논문에서 경유 노드는 주어진 네트워크에서 찾은 출발지에서 목적지간 최단 경로를 구성하는 임의의 노드들 중에서 선택된 노드로서, 출발지에서 목적지간 k개의 최단 경로 탐색 문제를 출발지에서 경유 노드간 k'개의 최단 경로 탐색 문제와 경유 노드에서 목적지간 k'개의 최단 경로 탐색 문제로 분리하기 위해 사용된다(k로부터 k'를 구하는 방법은 아래 4장에서 설명한다). 본 연구에서 경유 노드 선택은 출발지에서 경유 노드간 최단 경로 탐색을 위한 탐색 영역(Search space for finding the shortest path from origin to Through-Node)과 목적지에서 경유 노드간 최단 경로 탐색을 위한 탐색 영역(Search space for finding the shortest path from Through-Node to destination)을 합한 전체 탐색 영역의 최소화라는 관점에서 접근한다. 이러한 경유 노드 개념 도입의 목적을 만족하는 네트워크상에서의 최적해를 찾을 수 있는 기존의 방법은 없다. 이에 본 논문에서는

경유 노드를 선택하기 위한 방법으로 양방향 다익스트라 알고리즘(Bidirectional Dijkstra's Algorithm)을 사용하는데, 양방향 다익스트라 알고리즘을 이용하여 최단 경로를 구했을 때 순방향 탐색과 역방향 탐색에 의해 만난 노드(Meeting Node)를 경유 노드로 사용한다. 이러한 미팅 노드가 경유 노드 개념 도입 목적에 부합하는 최적해 여부에 대한 정교한 이론적 근거는 부족하나 일반적으로 실험 자료와 이론적 근거에 의해 양방향 다익스트라 알고리즘을 실행할 때의 탐색 영역이 단방향 탐색 알고리즘(Unidirectional Search Algorithm) 경우의 탐색 영역보다 훨씬 작으므로(Pohl, 1971), 미팅 노드를 경유 노드로 선택하는 것은 합리적이다.

이와 같이 경유 노드 개념을 도입하여 최단 경로 문제를 해결할 시 탐색 영역이 작아짐을4장에서 실험을 통해 설명한다.

3.2 양방향 다익스트라 알고리즘

앞서 설명하였듯이 본 논문에서 제시하고자 하는 알고리즘에서 경유 노드를 선정하는 방법으로 양방향 다익스트라 알고리즘을 사용한다. 양방향 다익스트라 알고리즘은 출발지에서부터의 다익스트라 알고리즘의 순방향 탐색과 목적지에서부터의 역방향 탐색을 교대로 실행하여 최단 경로를 탐색하는 알고리즘이다. 본 논문에서는 네트워크상의 노드들의 밀도와는 독립적으로, 단방향 탐색 사용시 탐색 영역보다 작은 탐색 영역을 갖을 가능성이 매우 높은, Pohl(1971)에 의해 제시된 양방향 다익스트라 알고리즘을 사용한다. Pohl의 양방향 다익스트라 알고리즘을 간단히 살펴보면 다음과 같다.

<기호 정의>

F = 순방향 탐색, B = 역방향 탐색

o = 출발지 노드, d = 목적지 노드

O = o로부터 최단 이동 소요 시간이 알려진 노드 중에서o로부터 도착한 노드들의 집합

D = d까지의 최단 이동 소요 시간이 알려진 노드 중에서d까지 하나의 경로를 갖는 노드들의 집합

$\tilde{O}$ = O에 있는 노드들로부터 하나의 링크로만 연결된 노드들 (O에 포함된 노드는 제외)

$\tilde{D}$ = D에 있는 노드들로부터 하나의 링크로만 연결된 노드들 (D에 포함된 노드는 제외)

$g_o(x)$ = o부터 노드 x까지 현재 최단 이동 소요 시간

$g_d(x)$ = d부터 노드 x까지 현재 최단 이동 소요 시간

<기본 절차>

1) (initialize) F_1(순방향 탐색 알고리즘의 첫 실행)과 B_1 실행
2) (strategy) ($\tilde{O}$의 크기 ≤ $\tilde{D}$의 크기) 이면 단계 3)으로 이동하고, 아니면 단계 4)로 이동
3) (forward expansion) F_2와 F_3 실행. F_3 실행에서 선택된 노드 n가 $n \in D$ 여부 판단. 맞으면 단계 5)로 이동하고 아니면 단계 2)로 다시 돌아감
4) (backward expansion) B_2와 B_3 실행. B_3 실행에서 선택된 노드 n이 $n \in O$ 여부 판단. 맞으면 단계 5)로 이동하고 아니면 단계 2)로 다시 돌아감
5) (halt) $\underset{x \in O \cap (D \cup \tilde{D})}{\forall} [g_o(x) + g_d(x)]$ 중 최소값을 갖는 노드 x를 지나는 경로를 최단 경로로 선택

본 논문에서는 위 단계 5)에서 얻은 노드 x를 경유 노드로 사용한다.

3.3 Lawler 알고리즘

Lawler 알고리즘은 단일 출발지와 단일 목적지 사이에서 복수 최단 경로를 찾기 위한 매우 효율적인 알고리즘이다. 새로운 알고리즘에서는 Lawler 알고리즘을 출발지에서 경유 노드간 k'개의 최단 경로를 탐색하거나 경유 노드에서 목적지간k'개의 최단 경로를 탐색할 때 사용한다. 또한 5장에서는 새로운 알고리즘의 비교 대상으로 사용된다. Lawler 알고리즘의 기본 절차는 다음과 같다(Zijpp *et al.*, 2005).

1) 네트워크상에서 모든 경로들의 집합을 규정하고 이 집합에서 최단 경로를 결정한다.
2) 나머지 경로들을 상호 배타적인(mutually exclusive) 부분집합들로 분할한다.
3) 부분집합들 내에서 각각의 최단 경로들을 결정한다.
4) 3)에서 구한 부분집합 내에서의 최단 경로들 중에서 최단 경로를 2번째 최단 경로로 결정한다.
5) 2) ~ 4)의 과정을 반복하여k번째 최단 경로를 구한다.

Lawler 알고리즘에 대한 구체적인 설명은 부록 1에 기술되어 있다.

4. TMPS 알고리즘

<기호 정의>

n_i : 네트워크상의 임의의 노드(i = 1, 2, 3,…)

o : 출발지

d : 목적지

m : 경유 노드

$p(n_i, \ldots, n_l, \ldots, n_j)$: 노드 n_l를 경유하는 노드 n_i에서 노드 n_j까지의 최단 경로(i, j, l = 1, 2, 3,…)

$g^{th}\ p(n_i, \ldots, n_l, \ldots, n_j)$: g번째 $p(n_i, \ldots, n_l, \ldots, n_j)$ (i, j, l, g = 1, 2, 3,…)

$T^g{}_p(n_i, \ldots, n_l, \ldots, n_j)$: $g^{th}\ p(n_i, \ldots, n_l, \ldots, n_j)$의 이동 소요 시간($i, j, l, g$ = 1, 2, 3,…)

k : 찾고자 하는 $p(o, d)$의 개수

k' : 찾고자 하는 $p(o, m)$와 $p(m, d)$의 개수
m_i : i번째 찾은 경유 노드(i = 1, 2, 3,...)
$r_{n_i n_j}$: 노드 n_i에서 노드 n_j간에 실제로 존재하는 경로의 개수
List A : m_i, $p(o, m_i, d)$, $T^1_{p(o,m_i,d)}$ 를 저장한 리스트
List B : $p(o, d)$와 $T^1_{p(o,d)}$ 를 저장한 리스트
h : 구하고자 하는 경유 노드의 개수

본 논문에서 제시하고자 하는TMPS 알고리즘의 기본 절차를 간략히 살펴보면 다음과 같다.

1) k를 기반으로 k' 결정
2) 양방향 다익스트라 알고리즘을 이용하여 경유 노드 선택
 가) 첫번째 실행시 2개 선정(두번째 경유 노드 선택시에는 첫번째 경유 노드를 네트워크 상에서 제거한 상태에서 선택)
 나) 두번째 실행부터 1개씩 선택(선행 실행에서 선택된 경유 노드를 네트워크 상에서 제거한 상태에서 선택)
3) 2)에서 제거한 노드들 복원
4) Lawler 알고리즘을 이용하여 k'개의 $p(o, m)$와 $p(m, d)$ 탐색
5) k'개의 $p(o, m)$와 $p(m, d)$를 연결하여 k개의 $p(o, d)$ 구함
6) 알고리즘 종료 조건 만족 여부 확인
 가) 만족시 알고리즘 종료
 나) 불만족시 2)로 되돌아 감

아래에서는 TMPS 알고리즘에 대해서 자세히 설명하도록 한다.

4.1 k' 결정 및 경로 연결 방법

TMPS 알고리즘에서의 경로는 출발지에서 경유 노드간 경로들과 경유 노드에서 목적지간 경로들의 연결을 통해 발생된다(이때, 순환 경로는 탐색하지 않기 위해 같은 노드를 포함하고 있는 경로들은 연결하지 않는다). TMPS 알고리즘의 경유 노드 선택 방법에 의거하여 하나의 경유 노드를 찾은 다음, Lawler 알고리즘을 이용하여 k'개의 $p(o, m)$을 구하고, 마찬가지로 k'개의 $p(m, d)$을 구했다고 한다면, 연결을 통해 만들 수 있는 $p(o, d)$의 수는 ($k' \times k'$)이다. 그러나 ($k' \times k'$) 개의 모든 경로들을 이동 소요 시간 기준으로 정렬하는 경우, $k' = k' + n$ (n : 1, 2, 3,...)가 되더라도 ($k' \times k'$) 개의 경로들 중 정렬 순서가 영향 받지 않을 경로들의 수는 $k'(k'+1)/2$이다. 그 이유를 <Table 1>를 통해 설명한다.

<Table 1>에 대해 설명하면 다음과 같다.

'Paths from origin to destination' 열에서 ($i^{th} + j^{th}$)는 i^{th} $p(o, m)$와 j^{th} $p(m, d)$를 연결한 $p(o, d)$를 의미하며, 'Group' 열은 ($i + j$)의 크기를 기준으로 $p(o, d)$를 정렬하였을 때 순위 그룹을 의미한다. 이때 임의의 그룹 G에 속하는 경로들은 같은 그룹내에서는 비교할 수 없지만 그룹 $G + n$ (n = 1, 2,...)에 속하는 경로들보다 이동 소요 시간이 작다고 가정한다. 이러한 가정에 대해 <Table 2>를 이용하여 설명하면 다음과 같다.

Table 1. Path Group by Travel Time(k' = 6)

Group	Paths from origin to destination ($p(o, m) + p(m, d)$)	Number of paths
1	$1^{st} + 1^{st}$	1
2	$1^{st} + 2^{nd}$, $2^{nd} + 1^{st}$	2
3	$1^{st} + 3^{rd}$, $3^{rd} + 1^{st}$, $2^{nd} + 2^{nd}$	3
4	$2^{nd} + 3^{rd}$, $3^{rd} + 2^{nd}$, $1^{st} + 4^{th}$, $4^{th} + 1^{st}$	4
5	$3^{rd} + 3^{rd}$, $2^{nd} + 4^{th}$, $4^{th} + 2^{nd}$, $1^{st} + 5^{th}$, $5^{th} + 1^{st}$	5
6	$3^{rd} + 4^{th}$, $4^{th} + 3^{rd}$, $2^{nd} + 5^{th}$, $5^{th} + 2^{nd}$, $1^{st} + 6^{th}$, $6^{th} + 1^{st}$	6
7	$4^{th} + 4^{th}$, $3^{rd} + 5^{th}$, $5^{th} + 3^{rd}$, $2^{nd} + 6^{th}$, $6^{th} + 2^{nd}$	5
8	$4^{th} + 5^{th}$, $5^{th} + 4^{th}$, $3^{rd} + 6^{th}$, $6^{th} + 3^{rd}$	4
9	$5^{th} + 5^{th}$, $4^{th} + 6^{th}$, $6^{th} + 4^{th}$	3
10	$5^{th} + 6^{th}$, $6^{th} + 5^{th}$	2
11	$6^{th} + 6^{th}$	1

Table 2. Path Group Agreement Ratio

(Unit : %)

# of Nodes \ k'	5	10	Agreement Ratio
1,000	94	91	92
2,000	82	93	91
Agreement Ratio	88	92	

위 <Table 2>는 임의의 $p(o, m)$와 $p(m, d)$를 연결한 $p(o, m, d)$가 <Table 1>에서와 같이 기 정의된 자신의 그룹을 포함한 상위 그룹에 매핑될 확률을 파악하기 위한 실험의 결과를 보여준다. 실험은 k'와 네트워크상의 노드 수를 2가지 경우로 구분하여 각각에 대해 30회씩 무작위로 네트워크 문제를 만들어서 실시되었다. <Table 2>에서의 90% 정도의 확률을 기반으로, 본 논문에서는 알고리즘의 속도 향상을 위해 일정 부분 정확도 손실(Loss of Accuracy)을 감수하여 $p(o, m)$와 $p(m, d)$를 연결하여 만든 $p(o, m, d)$는 기 정의된 자신의 그룹을 포함한 그 상위 그룹에 매핑된다고 가정한다.

'Number of paths' 열은 이동 소요 시간별 그룹내의 경로의 개수를 의미하며, 1에서부터 순차적으로 그룹, k'까지 1씩 증가하며, 그 이후로 다시 1씩 감소하는 형태를 갖는다.

위 <Table 1>에서 k'=3인 경우, 총 (3 × 3) = 9개의 출발지에서 목적지간 경로들을 만들 수 있다. 그러나 그룹 4의 $1^{st} + 4^{th}$, $4^{th} + 1^{st}$와 그룹 5의 $2^{nd} + 4^{th}$, $4^{th} + 2^{nd}$ 등과 같이 k'가 4일 경우에 만들어 질 수 있는 경로들에 의해 $2^{nd} + 3^{rd}$, $3^{rd} + 2^{nd}$, $3^{rd} + 3^{rd}$ 경로들은 순위가 변동될 수 있다. 이에 k' = 3일 경우, '확실한' $p(o, d)$의 개수는 그룹 1에서 그룹 3까지의 그룹내 경로 개수의 합인 (1 + 2 + 3) = 6이다.

k'가 주어졌을 때 '확실한' $p(o, d)$의 개수, k는 그룹 1에서 그룹 k'까지의 그룹내 경로 개수의 합으로, 식 (1)과 같다.

$$k = \frac{k'(k'+1)}{2} \quad (1)$$

식 (1)으로부터 k'를 구하는 식은 다음과 같이 유도할 수 있다.

$$k' = \left\lfloor \frac{\sqrt{8k+1}-1}{2} \right\rfloor \quad (2)$$

위 식 (2)에서 사용된 ⌊ ⌋ 기호를 다음과 같이 정의한다.

$y = \lfloor x \rfloor$ 일때, x값이 정수일 경우에는 $y = x$이며, x가 정수가 아닌 실수인 경우에는 $y = [x]+1$이 된다.

그런데 k'를 결정하는 데 있어 주어진 네트워크 상에 k개의 $p(o, d)$는 있으나, 존재하는 출발지에서 특정 경유 노드간 경로의 수, r_{om} 또는 특정 경유 노드에서 목적지간 경로의 수, r_{md}가 k' 보다 작은 경우가 있을 수 있다. 이러한 경우, $(r_{om} \times r_{md})$가 k보다 같거나 크다면 추가 작업이 필요없으나, 그렇지 않은 경우에는 경유 노드를 추가로 선택해서 새롭게 경로를 탐색하거나 출발지에서 경유 노드간 또는 경유 노드에서 목적지간 구간 중에서 실제로 존재하는 경로의 수가 k'보다 작은 경로 수를 가진 구간에서 추가로 아래 식 (3)의 k'' 만큼 경로를 탐색한다.

$$k'' = \left\lfloor \frac{k}{r} - k' \right\rfloor \quad (3)$$

위 식 (3)은 $r \cdot (k''+k') = k$에서 유도한 것이다. 여기서 r은 k'보다 작은 경로 수를 가진 구간에서 실제로 존재하는 경로의 수이다. K''을 구하는 방법에 대한 구체적인 설명은 부록 2에 기술되어 있다.

아래 절차는 위와 같은 특수한 경우를 위한 절차를 설명한 것이다.

1) $(r_{om} \times r_{md}) < k$인 경우, $\{(r_{om} < k') \text{ and } (r_{md} < k')\}$ 여부 확인
 가) $\{(r_{om} < k') \text{ and } (r_{md} < k')\}$ 인 경우
  (1) $p(o, m)$과 $p(m, d)$을 연결하여 만든 $(r_{om} \times r_{md})$개의 $p(o, m, d)$ 중에서 $1^{st}, 2^{nd}, \ldots, (r_{om} \times r_{md})^{th}$ $p(o, d)$와 $T^1_{p(o,d)}$, $T^2_{p(o,d)}$, …, $T^{(r_{om} \times r_{md})}_{p(o,d)}$을 List B에 추가
  (2) 마지막 단계로 이동
 나) $\{(r_{om} < k') \text{ and } (r_{md} < k')\}$ 가 아닌 경우
  (1) $(r_{om} < k')$ 이면,
   (가) 출발지 o와 경유 노드 m간에 경로를 k'' 만큼 추가로 탐색
   (나) 다음 단계로 이동
  (2) $(r_{om} < k')$ 가 아니면,
   (가) 경유 노드 m와 목적지 d간에 경로를 k'' 만큼 추가로 탐색
   (나) 다음 단계로 이동
2) $(r_{om} \times r_{md}) < k$ 가 아닌 경우, 다음 단계로 이동

4.2 경유 노드 선택 방법

TMPS 알고리즘에서는 경유 노드 선택 방법으로 양방향 다익스트라 알고리즘을 이용한다. 그러나 양방향 다익스트라 알고리즘은 $p(o, d)$와 미팅 노드 각 하나씩을 구하고 종료된다. TMPS 알고리즘에서는 다수의 경유 노드가 필요한 바, 기존의 양방향 다익스트라 알고리즘을 수정할 필요가 있다. 수정된 양방향 다익스트라 알고리즘을 이용하여 경유 노드를 선택하는 기본적인 절차는 다음과 같다.

1) TMPS 알고리즘 실행 횟수 확인
 가) 첫번째 실행이면 $h = 2$로 설정
 나) 첫번째 실행이 아니면 $h = 1$로 설정
2) 선행 TMPS 알고리즘 실행시에 경유 노드로 선정된 노드는 네트워크 상에서 제거한다
3) 양방향 다익스트라 알고리즘을 이용하여 $p(o, d)$ 탐색
4) $p(o, d)$ 탐색 성공시 얻은 미팅 노드를 경유 노드로 선택
5) h개의 경유 노드를 선택했는지 여부 확인
 가) 만족이면 2)에서 제거한 노드를 네트워크 상에 복원하고 다음 단계로 진행
 나) 불만족이면 2)로 되돌아 감

TMPS 알고리즘에서는 현재 i번째 실행에서 k'개의 $p(o, m)$ 또는 $p(m, d)$를 탐색하기 위한 경유 노드, m_i를 포함하여 선행 실행에서 사용하였던 모든 경유 노드들, $m_1, m_2, \ldots, m_{(i-1)}$을 제거한 네트워크 상에서 양방향 다익스트라 알고리즘을 사용하여 $p(o, d)$를 구하고 이로부터 구한 미팅노드를 또 하나의 경유 노드, $m_{(i+1)}$로 선택하여 $p(o, m_{(i+1)}, d)$와 $T^1_{p(o,m_{(i+1)},d)}$를 저장한다. 이는 TMPS 알고리즘 종료 여부를 결정할 때 이용하기 위함이다. 참고로 이와 같은 경유 노드 선택 방법을 사용한다면, 다음 식 (4)가 항상 성립한다.

$$T^1_{p(o,m_i,d)} \le T^1_{p(o,m_{(i+1)},d)} \quad (4)$$

4.3 알고리즘 종료 규칙

아래의 <Figure 1>은 TMPS 알고리즘의 매 시행 후 결과의 경우를 2가지로 나누어 나타낸 것이다.

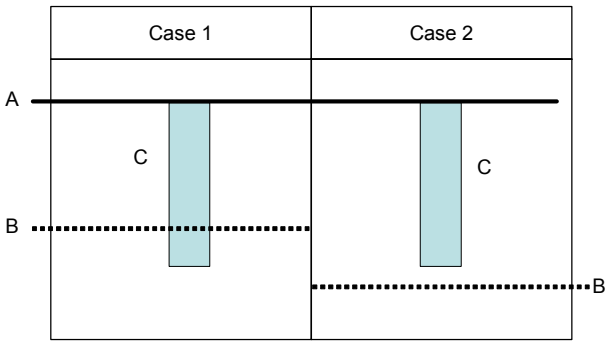

Figure 1. Two Cases of TMPS Results.

<Figure 1>에 대해 설명하면 다음과 같다.

세로 축을 시간 축이라고 하고TMPS 알고리즘을 i회 실행했을 때,

A : $T^1_{p(o,d)}$

B : $T^1_{p(o,m_{(i+1)},d)}$

C : TMPS 알고리즘을 *i*회 실행하여(*i*개의 경유 노드를 이용하여) 찾은 모든 *p*(*o*, *d*)를 이동 소요 시간 기준으로 정렬하였을 때, $1^{st} \sim k^{th}$ *p*(*o*, *d*)의 이동 소요 시간 범위

Case 1 : $T^1_{p(o,m_{(i+1)},d)}$ 이 TMPS 알고리즘 i회 실행하여 찾은 $T^1_{p(o,d)}$ 보다 작거나 같은 경우

Case 2 : $T^1_{p(o,m_{(i+1)},d)}$ 이 TMPS 알고리즘 i회 실행하여 찾은 $T^1_{p(o,d)}$ 보다 큰 경우

<종료 규칙>

TMPS 알고리즘을 *i*회 실행했을 때, Case 2인 경우에는 TMPS 알고리즘이 종료되지만 Case 1인 경우에는 1회 더 실행해야 한다. 왜냐하면 Case 1인 경우, TMPS 알고리즘 *i*회 실행하여 찾은 경로들 중 k번째 최단 경로의 이동 소요 시간 $T^1_{p(o,d)}$ 보다 더 빠른 *p*(*o*, $m_{(i+1)}$, *d*)가 있다는 것을 의미하기 때문이다.

4.4 TMPS 알고리즘 절차 요약

<초기화>

1) 출발지 노드 및 목적지 노드 선정;
2) *k*, *k'* 결정;
3) Set *i* = *j* = *h* = 1;
4) List A = List B = m_0 = Φ;

<단계 1>

If (TMPS 알고리즘의 첫번째 실행인가?)
 Then
  1) Set *h* = 2 ; 구하고자 하는 경유 노드의 개수를2로 설정
  2) Set *j* = *i* ;
 Otherwise
  1) Set *h* = 1 ; 구하고자 하는 경유 노드의 개수를1로 설정
  2) Set *j* = *i* + 1 ;

Set *a* = 1 ;
While (*a* ≤ h)
{
 1) List A에 있는 모든 경유 노드를 네트워크상에서 제거한다(이미 경유 노드로 선택된 노드가 다시 경유 노드로 선택되는 것을 방지);
 2) 양방향 다익스트라 알고리즘에 의해 *p*(*o*, *d*)를 구한다. 여기서 구한 미팅 노드를 경유 노드, m_j라 하고 *p*(*o*, m_j, *d*), $T^1_{p(o,m_j,d)}$ 와 m_j를 List A에 추가한다;
 3) Set *j* = *j* + *1* ;
 4) Set *a* = *a* + *1* ;
}

네트워크상에서 제거된 노드들을 모두 복원한다;
네트워크상에서 경유 노드, $m_{(i-1)}$를 제거한다;

<단계 2>

Lawler 알고리즘을 이용하여 각 *k'*개의 *p*(*o*, m_i)와 *p*(m_i, *d*)를 찾는다;

If ((r_{om_i} × $r_{m_i d}$) < *k*)
 Then
  If ((r_{om_i} < *k'*) and ($r_{m_i d}$ < *k'*))
   Then
    1) *p*(*o*, m_i)과 *p*(m_i, *d*)을 연결하여 (r_{om_i} × $r_{m_i d}$)개의 *p*(*o*, m_i, *d*)을 만든다;
    2) (r_{om_i} x $r_{m_i d}$)개의 *p*(*o*, *d*) 중 1^{st}, 2^{nd}, …, (r_{om_i} × $r_{m_i d}$)th *p*(*o*, *d*)와 $T^1_{p(o,d)}$, $T^2_{p(o,d)}$, …, $T^{(r_{om_i}\times r_{m_i d})}_{p(o,d)}$ 을 List B에 추가한다;
    3) 단계 5로 이동;
   Otherwise
    If ((r_{om_i} < *k'*)
     Then
      1) 출발지 *o*와 경유 노드 m_i간 경로를 $\left\lfloor \frac{k}{r_{om_i}} - k' \right\rfloor$ 만큼 추가로 탐색;
      2) 단계 3으로 이동;
     Otherwise
      1) 경유 노드 m_i와 목적지 *d*간 경로를 $\left\lfloor \frac{k}{r_{m_i d}} - k' \right\rfloor$ 만큼 추가로 탐색;
      2) 단계 3으로 이동;
 Otherwise
  단계 3으로 이동;

<단계 3>

단계 2에서 찾은 모든 *p*(*o*, m_i)과 *p*(m_i, *d*)을 연결하여 (*k'* × *k'*)개의 *p*(*o*, m_i, *d*)을 만든다;

<단계 4>

단계 3에서 연결하여 만든 (*k'* × *k'*)개의 *p*(*o*, *d*) 중 1^{st}, 2^{nd}, …, k^{th} *p*(*o*, *d*)와 $T^1_{p(o,d)}$, $T^2_{p(o,d)}$, …, $T^k_{p(o,d)}$ 을 List B에 추가한다;

<단계 5>

If (List B에 있는 $T^k_{p(o,d)}$ 이 List A에 있는 $T^1_{p(o,m_{(j+1)}d)}$ 보다 작은가?)

Then

1) List B에 있는 경로들 중 1^{st}, 2^{nd}, …, k^{th} $p(o, d)$과 $T^1_{p(o,d)}$, $T^2_{p(o,d)}$, …, $T^k_{p(o,d)}$ 을 출력한다;

2) TMPS 알고리즘 종료;

Otherwise

1) Set $i = i + 1$;

2) 단계 1로 되돌아 간다;

4.5 TMPS 알고리즘과 Lawler 알고리즘 비교

4.5.1 경로의 일치도 비교

TMPS 알고리즘에 의해 구한 경로들과 Lawler 알고리즘에 의해 구한 경로들을 비교하면, k가 커질 수록 그 일치성이 떨어질 수 있다. 그 이유를 살펴보면 다음과 같다.

TMPS 알고리즘을 i회 실행한 결과, ($T^{k'}_{p(o,m_i)} + T^1_{p(m_i,d)}$) 또는 ($T^1_{p(o,m_i)} + T^{k'}_{p(m_i,d)}$)이 $T^1_{p(o,m_{(j+1)},d)}$ 보다 작다고 했을 때, $T^{k'}_{p(o,m_i)}$ 보다 크지만 ($T^1_{p(o,m_{(j+1)},d)} - T^1_{p(m_i,d)}$) 보다는 작은 임의의 이동 소요 시간을 갖는 출발지에서 경유 노드(m_i)간 경로들과 $T^{k'}_{p(m_i,d)}$ 보다 크지만 ($T^1_{p(o,m_{(j+1)},d)} - T^{k'}_{p(m_i,d)}$) 보다는 작은 임의의 이동 소요 시간을 갖는 경유 노드(m_i)에서 목적지간 경로들을 연결한 경로들은 찾지 못한다. 이러한 이유로 TMPS 알고리즘으로 찾지 못한 경로들은 Lawler 알고리즘에 의해서는 찾을 수 있게 되어 두 알고리즘간 경로의 불일치가 발생할 수 있다.

4.5.2 탐색 영역 비교

$p(o, d)$를 구하기 위해 경유 노드를 이용하는 경우와 이용하지 않는 경우의 탐색 영역의 크기를 비교하면 아래<Figure 2>와 같다. <Figure 2>는 1,000개의 노드를 갖는 네트워크에서 각 노드에서 연결된 평균 링크 개수를 기준으로 3가지 경우로 구분하여 각각의 경우에 대해서 동일한 출발지와 목적지를 임의로 선정하여 최단 경로를 구했을 때 탐색 영역의 크기를 나타낸다.

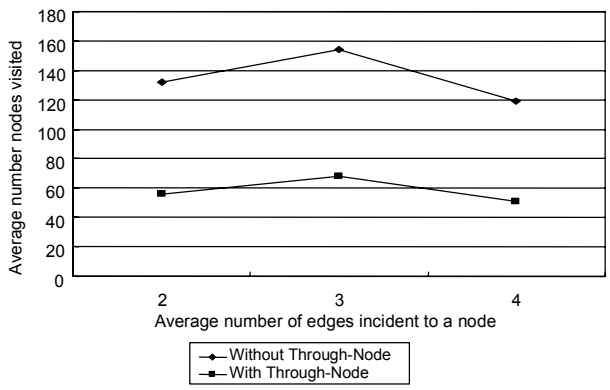

Figure 2. Search Space Size Comparison.

1) With Through-Node : Pohl의 양방향 다익스트라 알고리즘을 이용하여 경유 노드를 선택하여 $p(o, m)$, $p(m, d)$를 각각 구한 다음 이를 연결하여 $p(o, d)$를 구하는 경우(경유 노드를 구할 때 탐색 영역은 제외함).

2) Without Through-Node : Pohl의 양방향 다익스트라 알고리즘을 이용하여 직접 $p(o, d)$를 구하는 경우

위 실험의 결과를 바탕으로 경유 노드를 이용하는TMPS 알고리즘의 탐색 영역이 경유 노드를 사용하지 않는Lawler 알고리즘의 탐색 영역 보다 작음을 알 수 있다.

4.6 예제

아래 <Figure 3>의 샘플 네트워크는 26개의 노드를 가지고 있으며, 노드간 이동 소요 시간(단위 : 분)이 각 노드간 링크에 표시되어 있다.

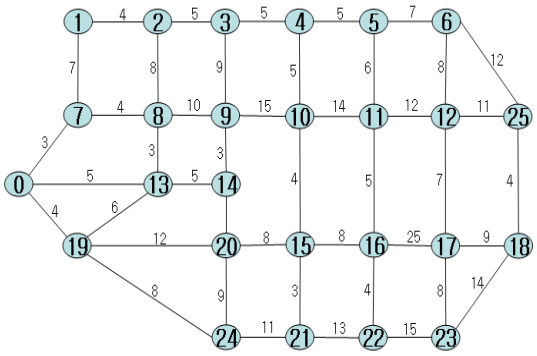

Figure 3. An Example Traffic Problem.

<초기화>

o = 노드 0, d = 노드 25 ;

k = 5, k' = 3

$i = j = n = 1$;

List A = $m_0 = \Phi$

<단계 1>

'경유 노드 선택 방법'을 이용하여 다음을 구함

m_1 = 노드 4, m_2 = 노드 16

$T^1_{p(o,m_1,d)} = 48$

$T^1_{p(o,m_2,d)} = 60$

<단계 2>

Lawler 알고리즘을 이용하여 다음의 결과를 구했음

양 구간 모두 가능 경로 수가 k'보다 크므로, 추가 경로 탐색 필요치 않음

Table 3. Obtaining k' (= 3) Paths

	Origin → Through-Node (4)		Through-Node (4) → Destination	
	Path	Travel time	Path	Travel time
1	0-7-1-2-3-4	24	4-5-6-25	24
2	0-7-8-2-3-4	25	4-5-6-12-25	31
3	0-13-8-2-3-4	26	4-5-11-12-25	34

<단계 3>
경로 연결 방법에 의해5개의 $p(o, m, d)$ 구함

<단계 4>
현재까지 찾은 모든 $p(o, 4, d)$를 이동 소요 시간 기준으로 정렬

Table 4. Final Result

	Origin → Through-Node (4) → Destination	
	Path	Travel time
1	0-7-1-2-3-4-5-6-25	48
2	0-7-8-2-3-4-5-6-25	49
3	0-13-8-2-3-4-5-6-25	50
4	0-7-1-2-3-4-5-6-12-25	55
5	0-7-8-2-3-4-5-6-12-25	56

<단계 5>
위 <Table 4>에서 5번째 최단 경로의 이동 소요 시간은56분으로, $T^1_{p(o,m_2,d)}$ = 60분 보다 빠르다. 그러므로 TMPS 알고리즘 종료 규칙을 만족. 위 <Table 4>의 경로들이 최종 결과값 임.

참고로 샘플 네트워크를 대상으로 Lawler 알고리즘에 의해 구한 결과값은 아래 <Table 5>와 같다.

Table 5. Lawler's Algorithm Result

	Origin → Through-Node (4) → Destination	
	Path	Travel time
1	0-7-1-2-3-4-5-6-25	48
2	0-7-8-2-3-4-5-6-25	49
3	0-13-8-2-3-4-5-6-25	50
4	0-13-14-9-3-4-5-6-25	51
5	0-7-1-2-3-4-5-6-12-25	55

TMPS 알고리즘 결과값과 Lawler 알고리즘 결과값을 비교하면, 최단 경로 5개 중 4개가 일치한다. TMPS 알고리즘에서 51분이 소요되는 경로를 찾지 못했는데, 그 이유는 앞서 설명했듯이 출발지에서 경유 노드(4)간 경로 중 3번째 최단 경로의 이동 소요 시간을 더해도 $T^1_{p(o,16,d)}$ 보다 작기 때문이다. 즉 51분이 소요되는 경로는 4^{th} $p(o, 4)$와 $p(4, d)$를 연결하여야만 찾을 수 있기 때문이다.

5. 실험 및 결과 분석

5.1 실험 개요

본 연구에서 제안한 TMPS 알고리즘에 대한 성능 평가는 일련의 실험에 의해 실시되었다. 실험의 목적은 다음과 같다.

1) TMPS 알고리즘 성능 평가를 위한 CPU 시간 측정
2) 비순환 경로 발생을 위한 가장 좋은 KSP 알고리즘으로 알려진 Lawler 알고리즘의 CPU 시간과 비교
3) TMPS 알고리즘 결과값과 Lawler 알고리즘 결과값 비교 - 경로 일치도 평가

실험 형태는 발생 경로 수(k = 10, 50, 100)에 대해 네트워크 크기(노드 수, n = 1000, 2000, 3000, 4000, 5000) 당 랜덤 샘플을 30회 발생시켜 평균 CPU 시간과 경로 일치도를 산출하는 것이다. 실험에 사용된 컴퓨터의 CPU 성능은 Pentium 4 3.6GHz이며, 메모리는 2GB이다.

5.2 실험 결과

5.2.1 CPU 시간 비교
아래 <Table 6>은 TMPS 알고리즘과 Lawler 알고리즘의 CPU 시간을 보여주고 있다.

Table 6. CPU Time Comparison (LA : Lawler's algorithm)
(unit : sec)

# of paths / # of nodes	5		10		50		100	
	TMPS	LA	TMPS	LA	TMPS	LA	TMPS	LA
1,000	0.5	0.6	0.6	0.8	1.2	2.5	1.8	3.8
2,000	1.9	2.4	2.7	3.4	5.8	11.1	7.2	18.7
3,000	3.4	5.3	6.4	12.6	15.5	45.7	19.7	70.2
4,000	5.8	9.2	14.2	24.5	27.7	84.8	32.3	111.6
5,000	6.9	14.5	27.1	48.7	54.5	162.4	68.5	254.4

위 <Table 6>에서 볼 수 있듯이, 실험 결과값은 네트워크상의 노드 수가 증가할수록, 또한 발생 경로 개수가 증가할수록 TMPS알고리즘의 CPU 시간과 Lawler 알고리즘의 CPU 시간의 격차가 커짐을 알 수 있다. 본 논문에서의 실험에서는 노드 수를 최대 5000으로 제한하였으나 실제 서비스 환경에서는 노드 수가 대도시의 경우 1만여개 정도가 되는데, 이런 경우 TMPS 알고리즘의 상대적인 계산상의 효율성은 더욱 커지게 될 것이다. 참고로 복수 경로 안내 서비스에서 5개 정도의 경로를 추천하는 것을 목표로 한다고 하였을 때, 위와 같은 조건에서 1

만개 노드를 가진 네트워크상에서 5개의 경로를 발생시키는데 소요되는 CPU 시간은 평균 11.8초였다.

5.2.2 경로 일치도 비교

아래 <Table 7>는 Lawler 알고리즘의 결과값을 기준으로 한 TMPS 알고리즘의 결과값과 Lawler 알고리즘의 결과값의 일치도를 보여주고 있다.

Table 7. Path Agreement Ratio (TMPS vs. Lawler's Algorithm)

(Unit : %)

# of paths / # of nodes	5	10	50	100
1,000	88	87	86	84
2,000	85	83	76	73
3,000	83	80	74	73
4,000	82	83	69	77
5,000	86	90	79	74
Average	85	85	77	76

위 <Table 7>에서 볼 수 있듯이, TMPS 알고리즘의 결과값과 Lawler 알고리즘의 결과값의 일치도 비교에서 노드 수의 크기에는 큰 상관관계가 없지만, 경로 수와는 상관관계가 있음을 알 수 있다. 경로 수가 적을 수록 높은 일치도를 보였다. 앞서 설명하였듯이 실제 서비스 환경에서 추천 경로 수를5개로 한다고 가정했을 때, 경로 수 5에서 보인 85%의 일치도는 비교적 만족스러운 결과이다.

6. 결 론

본 논문에서는 방향성이 있는 네트워크상의 비순환 경로를 찾기 위한 가장 효율적인 KSP 알고리즘 중 하나인 Lawler 알고리즘을 개선시킨 TMPS 알고리즘을 제안하였다. TMPS 알고리즘은 경로의 일치도 측면에서 비록 Lawler 알고리즘의 85% 정도이나, CPU 시간 측면에서 괄목할 만한 개선 효과를 보였고 이에 따라 TMPS 알고리즘은 실제로 복수 경로 안내 서비스를 구현할 수 있는 높은 가능성을 열었다. 그리고 TMPS 알고리즘에 CKSP(Constrained KSP) 알고리즘을 접목하거나, 과거에 축적된 데이터를 바탕으로 가능 경로(feasible path)를 구성할 가능성이 매우 높은 네트워크 상의 노드간 링크를 그룹핑하는 방법을 TMPS 알고리즘에 적용한다면 실제 탐색 대상 노드의 수를 줄일 수 있고, 계산 시간도 많이 향상시킬 수 있을 것으로 추론되어 향후 실시간 복수 경로 안내 서비스 구현을 위한 연구의 좋은 방향이 될 수 있으리라 생각된다. 본 논문에서는 경로들의 품질에 대한 문제는 고려 대상 밖이었으나, 향후 연구에서는 경로들의 품질을 결정에 영향을 미치는 변수들을 고려한 경로 발생 방법에 대해서 검토할 필요가 있다.

참고문헌

Bellman, R. and Kabala, R. (1960), On kth best policies, *Journal of SIAM*, **8**, 582-585.

Brander, A. W. and Sinclair, M. C. (1995), A comparative study of k-shortest path algorithms, In : Proc. 11th UK Performance Engineering Workshop for Computer and Telecommunications Systems.

Eppstein, D. (1994), Finding the K shortest paths, In : Proc. 35th IEEE Symp, Foundations of Computer Science (FOCS_94), 154-165.

Eppstein, D. (1998), Finding the K shortest paths, *SIAM Journal Computing*, **28**(2), 652-673.

Goldberg, A. and Harrelson, C. (2005), Computing the Shortest Path : A* Search Meets Graph Theory, Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, 156-165.

Fox, B. L. (1978), Data structures and computer science techniques in operation research, *Operations Research*, **26**, 686-717.

Hadjiconstantinou, E. and Christofides, N. (1999), An efficient implementation of an algorithm for finding k shortest simple paths, *Networks*, **34**, 88-101.

Hoffman, W. and Pavley, R. (1959), A method for the solution of the nth best path problem, *Journal of the Association for Computing Machinery* (*ACM*), **6**, 506-514.

Katoh, N., Ibaraki, T., and Mine, H. (1982), An efficient algorithm for k shortest simple paths, *Networks*, **12**, 411-427.

Lawler, E. L. (1976), *Combinatorial Optimization* : *Networks and Matroids*, Holt, Rinehart & Winston, New York.

Pohl, I. (1971), Bi-directional Search, *Machine Intelligence*, **6**, 127-140.

Shier, D. R. (1976), Iterative methods for determining the K shortest paths in a network, *Networks*, **6**, 205-229.

Shier, D. R. (1979), On algorithms for finding the K-shortest paths in a network, *Networks*, **9**, 195-214.

Schnabel, W. and Lohse, D. (1997), Grundlagen der Strassen- Verkehrstechnik under Verkehrsplanung, Baud 2, neue bearb, Aufl.-1997, Verlag für Bauwesen Gmbh, Berlin.

Song, J. H. (2004), *Introduction to Telematics*, Hongneung Chulpansa, Seoul, Korea, 3-8.

Van der Zijpp, N. J. and Fiorenzo Catalano (2005), Path enumeration by finding the constrained K-shortest paths, *Transportation Research Part B*, **39**, 545-563.

Yen, J. Y. (1971), Finding the K shortest loopless paths in a network, *Management Science*, **17**, 712-716.

부록 1 (APPENDIX 1)

Lawler 알고리즘(Zijpp *et al*., 2005)

<초기화>

1) 출발지에서 목적지까지의 모든 경로들의 집합을$S_{0,1}$이라고 정의한다(여기서 $S_{<k><부분집합>}$는 <*K*>번째 최단 경로를 포함하는 집합의 <부분집합>번째 부분집합).
2) 부분집합 $S_{0,1}$ 내에서 최단 경로를 구하고, 구한 경로를 $P(S_{0,1})$라고 정의한다(이 경로는 전체 최단 경로가 된다).
3) $S_{0,1}$- $P(S_{0,1})$ 를 $q(1)$개의 상호 배타적인 부분집합으로 분할한다. 그리고 이들 부분집합을 $S_{1,1}, S_{1,2},\ldots, S_{1,q(1)}$이라 한다.
4) m = 1으로 설정한다.

<Step : Finding (m+1)th shortest path until m=k>

5) 부분집합 $S_{m,1}, S_{m,2},\ldots, S_{m,q(m)}$에서 각각의 최단 경로를 구하고, 이들을 $P(S_{m,1}), P(S_{m,2}),\ldots, P(S_{m,q(m)})$이라 한다.
6) $\{P(S_{1,1}),\ldots, P(S_{1,q(1)}), P(S_{2,1}),\ldots, P(S_{2,q(2)}), P(S_{m,1}),\ldots, P(S_{m,q(m)})\}$에서 $(m+1)$번째 최단 경로를 찾는다.
7) 6)에서 찾은 $(m+1)$번째 최단 경로를 포함하는 부분집합을 $S_{a,j}$이라 하고, $S_{a,j}$ - $P(S_{a,j})$을 $q(m+1)$개의 상호 배타적인 부분집합으로 분할하고, 이들을 $S_{m+1,1}, S_{m+1,2},\ldots, S_{m+1,q(m+1)}$이라 한다.
8) $m = m + 1$

Lawler 알고리즘의 핵심은 집합 S-$P(S)$을 부분집합으로 분할하는 메커니즘인데, 이 메커니즘을 간략히 살펴보면 다음과 같다.

<Lawler의 파티션 규칙>

S를 출발지 'o'와 목적지 'd' 사이의 비순환 경로들의 집합이라 하고, 노드 $\{a_1, a_2, a_3,\ldots,a_q\}$로 구성된 $P(S)$를 집합 S에서 최단 경로로 하자. 그런 다음, Lawler 알고리즘은 집합 S-$P(S)$을 q개의 부분집합 $S_1, S_2,\ldots, S_q$으로 분할한다. 여기서 S_i는 노드 $\{a_1, a_2, \ldots,a_{i-1}\}$로 시작하지만, 노드 $\{a_i\}$는 배제한 출발지 'o'와 목적지 'd' 사이의 비순환 경로들의 집합이다.

부록 2 (APPENDIX 2)

k'' 결정 방법

k'는 k를 이용하여 식(3)으로부터 구한다. 그러나 주어진 네트워크 상에 k개의 $p(o, d)$는 있으나, 존재하는 $p(o, m)$ 또는 $p(m, d)$의 개수가 k'보다 작은 경우에는 TMPS 알고리즘을 이용하여 k개의 $p(o, d)$를 구할 수 없게 된다. 이러한 경우에서는 식(3)으로부터 얻은 k'보다 더 많은 $p(o, m)$ 또는 $p(m, d)$를 구해야 한다. 이에 대한 구체적인 내용을 아래 <Figure 4>를 이용하여 설명하면 다음과 같다.

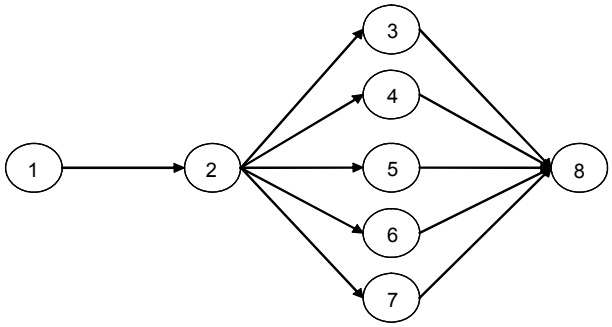

Figure 4. An example network.

위 <Figure 4>의 네트워크에서 5개의 $p(1, 8)$를 구하고자 하는 경우, TMPS 알고리즘에 의해 노드 2가 경유 노드로 선택되었다고 하자. K=5를 만족하기 위해서는 k'=3이 되어야 하지만, 가능한 $p(1, 2)$의 개수는 1에 불과하다. 이런 경우에는 아래 <Table 8>과 같은 결과가 나올 것이다.

Table 8. Obtaining k' (= 3) Paths

K'	Origin → Through-Node (2)	Through-Node (2) → Destination
	Path	Path
1	1-2	2-3-8
2	1-2	2-4-8
3	1-2	2-5-8

이러한 경우 $p(1, 8)$는 3개 밖에 구할 수 없을 것이다. 따라서 구하고자 하는 k값을 얻기 위해서는 식(3)을 이용해서 구한 k'' 만큼의 경로를 k'보다 작은 수의 가능 경로 수를 가진 구간에서 추가로 탐색해야 한다. 위의 예에서는 k'' = 5/1 - 3 = 2가 된다. 아래 <Table 9>와 <Table 10>은 k'를 5로 재설정하여 시행한 결과이다.

Table 9. Obtaining k' (= 5) Paths

k'	Origin → Through-Node (2)	Through-Node (2) → Destination
	Path	Path
1	1-2	2-3-8
2	1-2	2-4-8
3	1-2	2-5-8
4	1-2	2-6-8
5	1-2	2-7-8

Table 10. Final Result

k	Origin → Through-Node (2) → Destination
	Path
1	1-2-3-8
2	1-2-4-8
3	1-2-5-8
4	1-2-6-8
5	1-2-7-8