□ 研 究 論 文 □

韓 國 軍 事 運 營 分 析 學 會 誌
第 25 卷, 第 1 號, 1999. 6. 30

K-최단경로문제를 위한 MPS 방법의 효율적인 구현*

(An Efficient Implementation of the MPS Algorithm for the K-Shortest Path Problem*)

도승용, 성명기, 박순달**, 안재근***

Abstract

In this paper, we are concerned with the K-shortest loopless path problem. The MPS algorithm, recently proposed by Martins *et al.*, finds paths efficiently because it solves the shortest path problem only one time unlike other algorithms. But its computational complexity has not been known yet.

We propose a few techniques by which the MPS algorithm can be implemented efficiently. First, we use min-heap data structure for the storage of candidate paths in order to reduce searching time for finding minimum distance path. Second, we prevent the eliminated paths from reentering in the list of candidate paths by lower bounding technique. Finally, we choose the source node as a deviation node, by which selection time for the deviation node is reduced and the performance is improved in spite of the increase of the total number of candidate paths.

* 본 연구는 정보통신부의 97년도 대학기초연구지원사업(97-G-0683)의 지원을 받았음.
** 서울대학교 산업공학과
*** 안성산업대학교 컴퓨터공학과

1. 서 론

K-최단경로문제(K-shortest path problem)란 주어진 네트워크에서 시점(source node)과 종점(sink node)을 연결하는 경로들 중에서 첫 번째, 두 번째, ... , K 번째 최단경로를 구하는 문제로 여러 분야에 활용될 수 있는 문제이다[1][2]. 예를 들어 제약이 있는 최단경로문제의 경우에 제약조건을 제거한 다음 목적함수 값이 증가하는 순서로 경로를 구해 나가다가 제약조건을 만족하게 되면 최적해를 구하는 것이 된다. 또 다목적 최단경로문제는 유효경로(nondominated path)를 구하는 문제인데 이는 K-최단경로문제를 이용하여 구할 수 있다[11].

K-최단경로문제는 방문점의 중복 여부에 따라 일반경로(general path)와 단순경로(simple path)를 구하는 문제로 나눌 수 있다[3].

일반경로에 대한 연구는 크게 세 가지로 분류할 수 있다. 첫째로 Dreyfus[4]의 최적의 원리(principle of optimality)에 바탕을 둔 방법, 둘째로 Shier[10]의 꼬리표 최단경로문제에 바탕을 둔 방법, 마지막으로 Martins[7]의 경로삭제 개념을 이용한 방법 등이 있다. 이 중에서 Martins의 방법을 발전시킨 Azevedo[3]의 방법이 가장 효율적이라고 알려져 있다.

단순경로에 대한 연구는 Yen[12], Katoh[6], Perko[9] 그리고 Martins[8] 등이 있다. Yen은 K의 값에 따라 계산량이 선형으로 증가하는 최초의 방법을 제시하여 그 당시의 해법들에 비해서 매우 효율적인 성능을 보였다. Katoh는 Yen의 방법보다 좋은 수행 결과를 보이는 방법을 제시하였고, Perko는 경로를 보관하는 효율적인 자료구조에 대해 언급하였다. 그리고 Martins 등이 제시한 MPS 방법은 기존의 방법들과 달리 최단경로문제를 한 번만 푸는 방법으로서 아직 이론적인 계산 복잡도가 알려지지는 않았지만 다른 방법에 비해 실험적으로 매우 우수한 성능을 보이고 있음이 알려져 있다[8].

본 논문에서는, 경로 상에 같은 점이 없는 단순경로를 찾는 MPS 방법을 구현하는 데 있어 이진힙(binary heap)을 이용하여 후보경로를 저장하는 방법, 하한을 이용하는 방법 그리고 시점을 분기점(deviation node)으로 선택하는 방법을 통해 효율화하는 방법을 제시하고자 한다.

2. K-최단경로문제

유방향 네트워크 $G=(N, A)$가 있다고 하자. 여기서 $N=\{v_1, \cdots, v_n\}$은 점들의 집합이고 $A=\{a_1, \cdots, a_m\}$은 호들의 집합이다. 각 호 a_k는 호 (v_i, v_j)를 의미한다. 이 때 $tail(a_k)=v_i$, $head(a_k)=v_j$로 표현한다. 그리고 $s \in N$와 $t \in N$를 각각 시점, 종점이라고 할 때 s에서 t로의 경로 p는 연속된 점들의 집합 $p=\langle s=v_1', v_2', \ldots, v_h', v_{h+1}'=t \rangle$로 표현된다. 이 때 단순경로란 경로 p에 속한 모든 점 $i, j \in \{2, \cdots, h\}$에 대해서 $v_i' \neq v_j'$인 경로를 의미한다.

임의의 호 $a_k=(v_i, v_j)$에 대해서, c_{a_k} 또는 c_{ij}를 호 (i, j)의 거리라 하고 P를 시점 s에서 종점 t로의 경로들의 집합이라고 하자.

$p \rightarrow c(p)=\sum_p c_{ij}$는 각 경로에 대해 특정 값을 할당하는 P에서 정의되는 함수이다.

K-최단경로문제는 주어 양의 정수 K에 대해서 다음을 만족하는 $P_K=\{p_1,\cdots,p_K\}\subset P$를 구하는 것이다.

- $i\in\{1,\cdots,K-1\}$에 대해 p_i는 p_{i+1}보다 먼저 결정된다.
- $i\in\{1,\cdots,K-1\}$에 대해 $c(p_i)\le c(p_{i+1})$.
- $q\in P-P_K$에 대해 $c(p_K)\le c(q)$

3. MPS 방법

MPS 방법이 나오기 전의 방법들에서는 최단경로문제를 K회 풀어야만 K-최단경로문제를 풀 수 있었다. 그러나 MPS 방법에서는 다른 모든 점에서 종점까지의 최단경로나무(shortest path tree)를 구한 후 이를 이용하여 두 번째부터 K 번째 최단경로를 찾는다. 즉, K-최단경로를 구할 때 최단경로문제를 해법의 초기 단계에서 한 번만 적용한다.

MPS 방법이 이용하는 기본적인 정리들은 다음과 같다

보조정리 2.1[8] π_i를 v_i에서 t까지의 최단거리라 하고, $\bar{c}_{ij}=\pi_j-\pi_i+c_{ij},\ \forall(v_i,v_j)\in A$이라고 하면, 모든 $p\in P$에 대해서 $\bar{c}(p)=\sum_p \bar{c}_{ij}=\pi_t-\pi_s+c(p)$가 성립한다. ■

정리 2.2[8] $\bar{P}_K=\{\bar{p}_1,\cdots\cdots,\bar{p}_K\}$를 호 (v_i,v_j)와 관련되어진 값으로 $\bar{c}_{ij}$를 사용하였을 경우에 s에서 t까지 K-최단경로로 이루어진 집합이라고 하자. 그러면 $k\in\{1,\cdots\cdots K\}$에 대해 $c(\bar{p}_k)=c(p_k)$이 성립한다. ■

정리 2.3[8] p_{it}를 v_i에서 t까지의 최단경로라고 하고, $\pi_t=0$이라고 하자. 그러면 $v_i\in N$에 대해 $\bar{c}(p_{it})=0$이 성립한다. ■

MPS 방법은 모든 점 $v_i\in N-\{t\}$에서 t까지의 최단경로나무를 구한 다음 각 점 v_i에서 나오는 모든 호들을 $\bar{c}_{ij}$값의 오름차순으로 정렬한다. 그러면 각 점들로부터 첫 번째로 나오는 호는 최단경로상에 있는 호이다. 이 때 $\bar{c}_{ij}$를 호(i,j)의 할인가라고 하자.

MPS 방법에서는 경로의 분기점(deviation node)의 개념이 사용되어진다.

모든 $k>1$에 대해, $p_k\in P_K-\{p_1\}$는 s에서부터 $v(p_1),\cdots\cdots,v(p_{k-1})$ 점들까지 경로 $p_1,\cdots,p_{k-1}$와 부분경로를 각각 공유한다. $i\in\{1,\cdots\cdots,k-1\}$에 대해 $p_{s,v(p_i)}$를 s에서부터 $v(p_i)$까지의 p_k의 부분경로(sub-path)라고 하고, $p_{s,v(p_r)}$을 가장 많은 수의 점들을 가진 부분경로라고 하자. 그러면 $v(p_r)$을 경로 p_k의 분기점이라고 한다. 편의상 s는 p_1의 분기점이라고 하자.

p_{ij}와 p_{jh}를 각각 v_i에서 v_j까지 그리고 v_j에서 v_h까지의 경로라고 하고, $p_{ij}\diamond p_{jh}$를 v_i에서 v_j까지는 경로 p_{ij}와 같고 v_j에서 v_h까지는 경로 p_{jh}와 같은 v_i에서 v_h까지의 경로를 나타낸다고 하자. 그러면 MPS 방법은 [그림 1과] 같다.

4. 효율적인 구현방법

1: **Begin**
2: 모든 $v_i \in N-\{t\}$에서 t까지의 최단경로를 구함;
3: 모든 호 $(v_i, v_j) \in A$에 대해 $\bar{c}_{ij} \leftarrow \pi_j - \pi_i + c_{ij}$;
4: $A \leftarrow \{a_1, \cdots, a_m\}$, 여기서 $tail(a_h) = tail(a_{h+1})$이면 $\bar{c}_{a_h} \leq \bar{c}_{a_{h+1}}$, $h \in \{1, \cdots, m\}$;
5: LIST $\leftarrow \{p_1\}$; $k \leftarrow 0$;
6: **while** (LIST $\neq \emptyset$ and $k \leq K$) **do**
7: **begin**
8: $p \leftarrow \bar{c}(p) \leq \bar{c}(q)$, $q \in$ LIST인 LIST에 있는 경로;
9: **if** (p에 환이 없다) **then**
10: **begin**
11: $k \leftarrow k+1$;
12: $p_k \leftarrow p$;
13: **end**;
14: $v_i \leftarrow p$의 분기점;
15: **Repeat**
16: $p_{v_i} \leftarrow s$에서 v_i까지의 p의 부분경로;
17: $a_h \leftarrow$시작점이 v_i인 p의 호;
18: **while** (v_i가 a_{h+1}의 시작점이다) **and** (a_{h+1}가 p_{v_i}와 환을 이룬다) **do** h←h+1;
19: **if** (v_i가 a_{h+1}의 시작점이다) **then**
20: **begin**
21: $v_j \leftarrow a_{h+1}$의 끝점;
22: $\bar{p}_{v_j} \leftarrow v_j$에서 t까지의 최단경로;
23: LIST $\leftarrow$ LIST $\cup$ $(p_{v_i} \diamond \langle v_i, a_{h+1}, v_j \rangle \diamond \bar{p}_{v_j})$;
24: **end**;
25: $v_i \leftarrow p$에서 다음 점;
26: **until** (p_{v_i}는 환을 가진다) **or** ($v_i = t$);
27: **end**;
28: **end.**

[그림 1] MPS 방법[8]

MPS 방법을 구현할 때 수행시간에 크게 영향을 미치는 단계들로는 분기점의 선택단계, 구해진 후보경로들로부터 최소의 거리를 가지는 경로를 선택하는 단계이다. 따라서 위의 단계에서 걸리는 시간을 줄이기 위해 이진힙을 이용하여 후보경로를 저장하는 방법, 하한을 이용하여 후보경로의 수를 줄이는 방법, 새로운 분기점의 선택방법 등을 제시하고자 한다.

4.1 후보경로들의 저장

MPS 방법에서는 매회 후보경로들 중에서 최소의 거리를 가지는 경로를 선택하여, 이 경로가 환을 가지고 있지 않으면 K-최단경로 목록에 저장한다. 매회 이 후보경로들의 거리를 모두 비교하여 최소의 거리를 가지는 경로를 선택한다는 것은 비효율적이다. 왜냐하면 K-최단경로를 구하는 동안에 매우 많은 수의 후보경로들이 생성되기 때문이다. 따라서 이 후보경로들을 최소힙(min-heap)에 저장하면 매회 최소의 거리를 가지는 후보경로를 선택하는 시간을 줄일 수가 있다.

본 논문에서는, MPS 방법을 구현하는 데 후보경로들을 저장하기 위해 최소힙 중에서 이진힙(binary heap)을 사용하였다. 이 이진힙에는 생성되어지는 후보경로를 가리키는 주소가 저장되어진다. 후보경로들을 저장하기 위해 이차원 배열을 사용할 수도 있으나 이는 기억공간의 낭비를 초래할 수가 있다. 따라서 후보경로를 일차원 배열에 저장하기 위해서는 각 후보경로들의 시작위치를 나타내는 역할을 하는 배열을 도입하면 된다. 후보경로들의 저장을 위한 자료구조는 다음과 같다.

DHEAP[i] : 이진힙에서 i번째 위치에 있는 경로를 가리키는 포인터.

KTREE[MAX] : 후보경로를 저장하는 배열.

POINT[i] : 후보경로 i의 KTREE에서의 시작위치.

KDIST[i] : 후보경로 i의 거리.

TCOUNT[i] : 후보경로 i에 있는 점의 수.

KSHORT[i] : i번째 최단경로를 가리키는 배열.

4.2 후보 경로의 수를 줄이는 방법

MPS 방법에서 생성되는 후보경로들 중에는 힙에 저장되어 있는 경로일 수도 있고 힙에서 제거되어진 경로일 수도 있다. 현재 생성되어진 후보경로가 이전에 생성되어진 경로인지 확인하는 방법은 현재의 후보경로의 길이와 같은 길이를 가지는 경로가 힙에 존재하면 두 경로를 비교하여 같은 경로인지 확인하는 방법이다. 하지만 이 방법은 힙에 저장되어진 모든 경로를 검색해야 하기 때문에 시간이 많이 걸린다.

힙에 저장되어진 경로들과 생성된 후보경로를 비교하지 않고 구현하는 방법에는 다음의 두 가지가 있다.

첫 번째 방법은, 생성되어진 후보경로를 힙에 저장하고 난 후 힙에서 최소의 거리를 가지는 경로가 선택되어질 때 선택되어진 경로가 현재까지 발견되어진 최단경로들과 비교하여 같은 경로가 있으면 이 경로를 K-최단경로를 저장하는 목록에 저장하지 않는 것이다.

두 번째 방법은, 힙에 저장되어진 경로를 검색하는 대신에 경로에 대한 하한을 이용하여 이 하한보다 거리가 큰 경로들만 힙에 저장한다. 그 다음에 힙에서 제거되어진 경로를 이미 발견되어진 최단경로들 중 큰 거리를 가지는 경로부터 차례로 비교하여 같은 경로인 경우에는 이 경로를 K-최단경로를 저장하는 목록에 저장하지 않는다. 이때하한 값으로는 가장 최근에 힙에서 제거되어진 경로의 길이로 정한다. 물론 이 경로는 환을 포함할 수도 있고 포함하지 않을 수도 있다. 이 거리보다 더 짧은 거리를 가진 경로는 이미 힙에서 제거되어진 경로임을 알 수 있다.

<표 1> 하한을 사용한 경우와 사용하지 않은 경우의 비교

문제	점	호	하한을 사용하지 않은 경우		하한을 사용한 경우	
			후보경로	수행시간(초)	후보경로	수행시간(초)
1	120	480	5063	0.45	5023	0.42
2	390	1200	6225	0.41	6202	0.39
3	409	2000	----	----	8094	0.48
4	508	3000	7543	0.53	7431	0.49
5	890	5000	7779	0.32	7779	0.31
6	999	5003	6702	0.37	6352	0.33
7	1200	6000	7242	0.37	7236	0.36
8	1367	7645	8844	0.44	8842	0.45
9	1789	3837	7872	0.48	7820	0.50
10	2000	8999	6644	0.42	6644	0.41
평균			7102	0.42	7036	0.41

---- 후보경로의 수가 저장공간의 용량 초과하여 풀지 못함

첫 번째 방법보다는 두 번째 방법이 이진힙에 저장되어지는 후보경로들의 수가 적으므로 이진힙에서 필요한 연산이 적어진다. 따라서 두 번째 방법이 효율적이다.

4.3 분기점의 선택

[그림 1]의 MPS 방법의 14 번째 줄에서 분기점을 선택하게 되는데, 분기점을 선택하기 위해서는 이전에 발견되어진 최단경로들과 차례로 비교하여야 한다. 이 방법은 현재 발견되어진 최단경로들의 수가 커질수록 시간이 많이 걸린다. MPS 방법에서 분기점을 찾는 시간은 전체 수행시간 중에서 80-90%를 차지한다.

따라서 MPS 방법을 효율적으로 구현하기 위해서는 분기점을 찾는 시간을 줄이는 것이 중요하다. 따라서 본 논문에서는, 분기점을 기존에 발견되어진 경로들과 비교하여 분기점을 찾는 원래의 방법에서 제시한 방법 대신에, 시점을 분기점으로 선정하는 방법을 사용하였다. 그러면 분기점을 선택하는 시간은 없어지지만 원래의 방법보다 분기점이 경로의 앞부분에서 선택되어지기 때문에 생성되어지는 후보경로의 수는 같거나 많아지게 된다. 이 방법의 단점은 후보경로에 있는 점의 수가 많은 격자형 네트워크(grid network)에서는 후보경로의 수가 MPS 방법보다 많이 증가하기 때문에 후보경로를 보관하는데 많은 기억용량이 필요하다는 단점이 있다.

5. 실험결과

앞에서 제시한 방법들이 얼마만큼 효율적인가를 비교하기 위해 Goldberg가 개발한 문제생성기를 사용하여 만든 무작위 네트워크(random network)에 대하여 실험해 보았다. 실험한 컴퓨터는 LINUX를 설치한 PENTIUM-II 233Mhz CPU에 64M RAM을 가진 PC이고, 최적화 옵션을 -O3로 하여 gcc v2.7로 컴파일하였다.

1000개의 최단경로를 구하는 시간을 측정하였다. 여기서 측정시간은 처음의 최단경로나무를 구하

<표 2> 분기점의 선택에 따른 수행속도의 비교

문제	점	호	MPS 방법		출발점을 분기점으로 선택	
			후보경로	수행시간(초)	후보경로	수행시간(초)
1	20,001	80,004	10,397	0.37	12,081	0.06
2	40,388	200,007	9,661	0.44	11,156	0.06
3	60,038	400,002	10,933	0.40	12,502	0.07
4	90,023	500,001	10,864	0.34	12,345	0.08
5	100,000	400,000	11,015	0.48	12,702	0.08
6	120,000	480,000	12,991	0.43	14,646	0.14
7	139,998	500,284	12,506	0.44	14,308	0.14
8	173,883	754,843	12,221	0.42	13,844	0.11
9	190,000	902,744	11,583	0.50	13,484	0.12
10	195,000	499,000	13,799	0.50	15,584	0.11
평균			11,597	0.43	13,265	0.10

는 시간은 제외되었다.

<표 1>을 보면 하한을 사용한 경우와 하한을 사용하지 않는 경우가 큰 차이는 나지 않지만 모든 문제에 대해서 하한을 사용한 경우가 우수함을 알 수 있다. 특히 문제 3은 하한을 사용하지 않는 경우는 후보경로의 수가 많아서 풀 수 없는 문제이다.

[표 2]를 보면 초기점으로 분기점을 선택한 경우가 그렇지 않은 경우에 비해 4배 정도 빠름을 알 수 있다. 하지만 MPS 방법에 비해서 생성되는 후보경로의 수가 10-20%정도 많음을 알 수 있다. 즉, 분기점을 시점으로 선택하는 방법은 후보경로의 수는 증가하지만, 분기점을 선택하는 시간을 줄일 수 있어 속도면에서는 유리한 것으로 보인다.

6. 결 론

K-최단경로를 구하는 방법 중에서 MPS 방법은 계산 복잡도는 아직 밝혀지지 않았지만 실험적으로 효율적인 방법으로 알려져 있다. 본 논문에서는 MPS 방법을 구현하는 데 있어 이진힙을 이용하여 후보경로를 저장하는 방법, 하한을 이용하는 방법 그리고 시점을 분기점으로 선택하는 방법 등을 통한 효율적인 구현 방법을 제시하였다.

그리고 시점을 분기점으로 선택하는 방법은 격자형 네트워크에서는 후보경로의 수가 많이 증가하여 많은 기억공간을 차지하게 되어 큰 문제를 풀 때 어려움이 있다. 따라서 후보경로의 수를 줄일 수 있는 분기점 선택방법에 대한 추후연구가 필요하다.

참고문헌

[1] 박순달, 경영과학, 3정판, 민영사, 1998.

[2] 성기석, "수송네트워크에서 최대 물동량경로 문제의 최적해법", 한국경영과학회지, 제16권, 제1호, pp. 1-12, 1991.

[3] Azevedo José Augusto de, Joaquim João E. R. Silvestre Madeira, Ernesto Q. Viera

Martins, Filipe Manuel A. Pires, "A Shortest Path Ranking Algorithm", Proceedings of AIRO'90 - Models and Methods for Decision Support, pp. 1001-1011, 1990.

[4] Dreyfus S., "An Appraisal of Some Shortest Paths Algorithms", Operations Research, Vol. 17, pp. 395-412, 1969.

[5] Horowitz, E., S. Sahni, S. Anderson-Freed, Fundamentals of Data Structures in C, Computer Science Press, 1993.

[6] Katoh N., T. Ibaraki, H. Mine, "An Efficient Algorithm for K Shortest Simple Paths", Networks, Vol. 12, pp. 411-427, 1982.

[7] Martins E. Q. V., "An Algorithm for Ranking Paths that May Contain Cycles", European Journal of Operational Research, Vol. 18, pp. 123-130, 1984.

[8] Martins, E. Q. V., M. M. B. Pascoal, J. L, E. Santos, "A New Algorithm for Ranking Loopless Paths", Technical Report, Department of Mathematics, University of Coimbra, Portugal, 1997.

[9] Perko A., "Implementation of Algorithms for K Shortest Loopless Paths", Networks, Vol. 16, pp. 149-160, 1986.

[10] Shier D. R., "On Algorithms for Finding the k Shortest Paths in a Network", Networks, Vol. 9, pp. 195-214, 1979.

[11] Skiscim C. C., B. L. Golden, "Solving k-Shortest and Constrained Shortest Path Problems Efficiently", Annal of Operations Research, Vol. 20, pp. 249-282, 1989.

[12] Yen J. Y., "Finding the K Shortest Loopless Paths in a Network", Management Science, Vol. 17, pp. 712-716, 1971.

[98년 10월 30일 접수, 99년 6월 2일 심사완료]