Journal of The Korea Society of Computer and Information
Vol. 19, No. 7, July 2014
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2014.19.7.103

최소신장트리를 이용한 무방향 그래프의 점대점 최단경로 탐색 알고리즘

이 상 운*

A Point-to-Point Shortest Path Search Algorithm in an Undirected Graph Using Minimum Spanning Tree

Sang-Un Lee *

요 약

본 논문은 실시간 GPS 항법시스템에서 최단경로 탐색에 일반적으로 적용되고 있는 Dijkstra 알고리즘을 양방향 통행로 (무방향 그래프)로만 구성된 도로에 적용하고 문제점을 개선한 알고리즘을 제안하였다. Dijkstra 알고리즘은 방향 그래프에서 출발 노드부터 시작하여 그래프의 모든 노드에 대한 최단경로를 결정하기 때문에 알고리즘 수행에 많은 메모리가 요구되어 실시간으로 정보를 제공하지 못할 수도 있다. 이러한 문제점을 해결하고자, 본 논문에서는 무방향 그래프에 적합하도록 출발과 목적지 정점을 제외한 경로 정점들에 대해 최단경로를 설정하고, 출발 정점부터 시작하여 정점 유출 간선들에 대해 최단경로 설정 간선들과 일치하는 간선들을 모두 선택하는 방식으로 한 번에 다수의 정점들을 탐색하는 방법을 택하였다. 9개의 다양한 무방향 그래프에 제안된 알고리즘을 적용한 결과 모두 최단경로를 탐색하는데 성공하였다. 또한, 수행 속도 측면에서 Dijkstra 알고리즘보다 약 60%를 단축시키는 효과를 얻었으며, 알고리즘 수행에 필요한 메모리도 월등히 적게 요구되었다.

▸ Keywords : 무방향 그래프, 최단경로, Dijkstra 알고리즘, GPS 항법 시스템, 점대점 탐색

Abstract

This paper proposes a modified algorithm that improves on Dijkstra's algorithm by applying it to purely two-way traffic paths, given that a road where bi-directional traffic is made possible shall be considered as an undirected graph. Dijkstra's algorithm is the most generally utilized form of shortest-path search mechanism in GPS navigation system. However, it requires a large amount of memory for execution for it selects the shortest path by calculating distance between the starting

•제1저자 : 이상운
•투고일 : 2014. 4. 25, 심사일 : 2014. 5. 22, 게재확정일 : 2014. 7. 1.
* 강릉원주대학교 멀티미디어공학과 (Dept. of Multimedia Eng., Gangneung-Wonju National University)

node and every other node in a given directed graph. Dijkstra's algorithm, therefore, may occasionally fail to provide real-time information on the shortest path. To rectify the aforementioned shortcomings of Dijkstra's algorithm, the proposed algorithm creates conditions favorable to the undirected graph. It firstly selects the shortest path from all path vertices except for the starting and destination vertices. It later chooses all vertex-outgoing edges that coincide with the shortest path setting edges so as to simultaneously explore various vertices. When tested on 9 different undirected graphs, the proposed algorithm has not only successfully found the shortest path in all, but did so by reducing the time by 60% and requiring less memory.

▸ Keywords : Undirected graph, Shortest path, Dijkstra algorithm, GPS navigation system, Point-to-point search

I. 서 론

최근 들어 실시간 GPS 항법 시스템 (real time GPS navigation system)에 방향 그래프 (digraph)의 최단경로 (SP, shortest path)를 구하는 Dijkstra SP 알고리즘이 일반적으로 적용되고 있다.[1] 실시간 GPS 항법 시스템은 운전자에게 자신의 현재 위치를 화면상에 지정하고, 목적지를 결정하면 최단경로 (시간 또는 거리)를 결정하여 화면상에 표시해주고 운전자가 길을 따라 가도록 유도하는 시스템이다. 최단경로를 찾는 알고리즘은 Dijkstra, Bellman Ford, Topological Ordering과 A-Star (A*) 등이 있다[1,2].

Dijkstra SP 알고리즘은 양의 가중치를 갖는 호들로 구성된 방향 그래프의 단일 출발점 최단경로 (single-source shortest path)를 찾는 알고리즘이다[3-5]. GPS 시스템의 최단경로는 단일 출발점과 단일 목적지의 최단경로를 찾는 점대점 (P2P, point-to-point) 최단경로 알고리즘으로 Dijkstra SP 알고리즘의 특별한 경우이다. 점대점 최단경로 탐색 문제도 Dijkstra SP 알고리즘에 기반을 두고 있다[1,6,7].

만약 도시의 모든 도로가 일방통행로가 없는 양방향 도로로 구성되어 있는 경우, 이는 무방향 그래프 (undirected graph)로 생각할 수 있다. 방향 그래프에 적용된 Dijkstra SP 알고리즘을 무방향 그래프에도 적용할 수 있는가? 만약 Dijkstra SP 알고리즘을 무방향 그래프에 적용할 수 있다면, 무방향 그래프에서 Dijkstra SP 알고리즘보다 메모리를 적게 요구하면서 빠른 알고리즘을 구현할 필요가 있다.

본 논문은 무방향 그래프의 크기가 클 경우라도 적은 메모리 용량을 필요로 하면서도 간단한 수행 횟수로 최단경로를 찾는 알고리즘을 제안한다. 2장에서는 Dijkstra SP 알고리즘의 최단경로를 찾는 과정을 고찰해 보고 문제점을 살펴본다. 3장에서는 Dijkstra SP 알고리즘의 단점을 보완한 새로운 최단경로 탐색 알고리즘을 제안한다. 4장에서는 제안된 알고리즘을 다양한 방향 그래프에 적용하여 성능을 평가해 본다.

II. 관련연구와 연구 배경

그래프는 정점들 (vertices)과 간선들(edges)로 구성되어 있으며, 정점들이 간선들로 연결되어 있고 (connected), 간선들은 방향성 (directed)과 무방향성 (undirected)으로 구분된다. 무방향 그래프는 $G=(V,E)$로 표기하며, 방향 그래프는 $G=(N,A)$로 표기한다. 노드 (nodes, N)는 정점 (V)이라 하며, 호 (arcs, A)는 간선 (E) 또는 화살표 (arrows)라고도 한다. 무방향 그래프의 간선은 $e=\{x,y\}$로, 방향 그래프의 호는 방향성이므로 순서쌍 $a=(x,y)$로 표기한다. 여기서, x는 꼬리 (tail), y는 머리 (head)라 하며, 노드 n은 유출 차수와 유입 차수를 갖는다. 또한 간선과 호는 가중치를 갖고 있다. 방향그래프의 최단거리를 찾는 대표적인 알고리즘인 Dijkstra SP 알고리즘[3-5]은 단일 출발 노드 최단경로를 찾는 알고리즘이다.

Dijkstra SP 알고리즘은 "특정 노드로부터 시작하여 최소 경로 길이를 갖는 노드를 한 번에 하나씩 선택하는 방식"으로,

특정 노드에 인접한 노드들과 이전에 계산된 노드들의 경로의 합이 최소가 되는 노드를 찾는다. 따라서 출발 노드로부터 시작하여 모든 노드들을 방문하는 최단경로를 찾기 때문에 $n-1$회의 알고리즘이 수행된다. 이는 Prim의 최소신장트리(MST, minimum spanning tree)를 찾는 알고리즘[8]과 유사한 방법이다. 참고로, Prim MST 알고리즘[8]은 무방향 그래프에서 모든 정점을 연결하는 최소 신장트리를 구성하는 것으로, 임의의 정점을 선택하고, 이에 연결된 간선들 중에서 최소 가중치 간선 (MWE, minimum-weighted edge)을 가진 정점을 선택한다. 새로 선택된 정점의 간선들 가중치와 기존에 방문하였지만 선택되지 않은 간선들 중에서 MWA를 선택한다. (단, 이 과정에서 사이클이 발생하는 간선은 무시한다.) 모든 정점들이 선택될 때까지 이 과정을 반복적으로 수행한다. Dijkstra SP 알고리즘을 무방향 그래프에 적용하기 위해서는 노드를 정점으로, 호를 간선으로 치환하면 된다.

Dijkstra SP 알고리즘을 그림 1의 G_1 무방향 그래프에 대해 적용하여 보자.

G_1 그래프는 Chen[9]에서 인용되었으며, $|v|=8, |e|=15$인 그래프로 ⓐ에서 출발하여 ⓗ에 도착하는 최단경로를 찾는 문제이다. 출발 정점에서 목적지 정점까지 최단거리를 갖는 경로를 Dijkstra SP 알고리즘으로 찾는 과정은 표 1에 제시되어 있다.

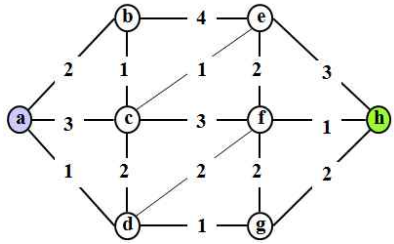


그림 1. G_1 그래프
Fig. 1. G_1 graph

Dijkstra SP 알고리즘은 항상 목적지까지의 최단경로를 찾는데 성공할 수 있다. 그러나 단점은 많은 양의 메모리를 요구하며, 일반적으로 $n-1$회를 수행하기 때문에 그래프가 복잡해질수록 수행속도가 느리다. 즉, 하나의 정점에서 경로의 합과 최소 가중치 간선 저장에 각각 n 개의 메모리가 요구되며, SP 간선들과 새로 탐색을 시작하는 정점의 경로 길이 $l(i)$이 1개씩 필요하다. 따라서 $n+2$개의 메모리가 $n-1$번 소요된다. 또한, 표 1에서 $v_1=d$에서의 $l(b)=2$와 $l(g)=2$, $v_3=g$에서의 $l(c)=3$과 $l(f)=3, v_5=f$에서의 $l(e)=4$와 $l(h)=4$가 동일한 경로 길이를 갖고 있음에도 불구하고 1개씩만 선택하기 때문에 $n-1$번 수행된다. 또한, 한번 경로가 설정된 정점으로 유출되는 간선의 경로 길이는 계산하지 않아야 한다.

표 1. G_1 그래프의 Dijkstra SP 알고리즘
Table 1. Dijikstra's SP algorithm for G_1 graph

		$l(a)$	$l(b)$	$l(c)$	$l(d)$	$l(e)$	$l(f)$	$l(g)$	$l(h)$	SP Edges
초기치	l_{new}	**0**	∞	∞	∞	∞	∞	∞	∞	
$v_0=a$ $l(a)=0$	e_{old} e_{new} $l_{\min}$ MWE l_{new}		– $\{a,b\}=2$ $min(\infty,2)$ $\{a,b\}=2$ 2	– $\{a,c\}=3$ $min(\infty,3)$ $\{a,c\}=3$ 3	– $\{a,d\}=1$ $min(\infty,1)$ $\{a,d\}=1$ 1	– – – – ∞	– – – – ∞	– – – – ∞	– – – – ∞	$\{a,d\}=1$
$v_1=d$ $l(d)=2$	e_{old} e_{new} $l_{\min}$ MWE l_{new}	$\{d,a\}=1$ ×	$\{a,b\}=2$ – – $\{a,b\}=2$ 2	$\{a,c\}=3$ $\{d,c\}=2$ $min(3,3)$ $\{a,c\}=3$ 3		– – – – ∞	– $\{d,f\}=2$ $min(\infty,3)$ $\{d,f\}=2$ 3	– $\{d,g\}=1$ $min(\infty,2)$ $\{d,g\}=1$ 2	– – – – ∞	$\{a,b\}=2$
$v_2=b$ $l(b)=2$	e_{old} e_{new} $l_{\min}$ MWE l_{new}	$\{b,a\}=2$ ×		$\{a,c\}=3$ $\{b,c\}=1$ $min(3,3)$ $\{a,c\}=3$ 3		– $\{b,e\}=4$ $min(\infty,6)$ $\{b,e\}=4$ 6	$\{d,f\}=2$ – – $\{d,f\}=2$ 3	$\{d,g\}=1$ – – $\{d,g\}=1$ 2	– – – – ∞	$\{d,g\}=1$
$v_3=g$ $l(g)=2$	e_{old} e_{new} $l_{\min}$ MWE l_{new}			$\{a,c\}=3$ – – $\{a,c\}=3$ 3	$\{g,d\}=1$ ×	$\{b,e\}=4$ – – $\{b,e\}=4$ 6	$\{d,f\}=2$ $\{g,f\}=2$ $min(3,4)$ $\{d,f\}=2$ 3		– $\{g,h\}=2$ $min(\infty,4)$ $\{g,h\}=2$ 4	$\{a,c\}=3$
$v_4=c$ $l(c)=3$	e_{old} e_{new} $l_{\min}$ MWE l_{new}	$\{c,a\}=3$ ×	$\{c,b\}=1$ ×		$\{c,d\}=2$ ×	$\{b,e\}=4$ $\{c,e\}=1$ $min(6,4)$ $\{c,e\}=1$ 4	$\{d,f\}=2$ $\{c,f\}=3$ $min(3,6)$ $\{d,f\}=2$ 3		$\{g,h\}=2$ – – $\{g,h\}=2$ 4	$\{d,f\}=2$
$v_5=f$ $l(f)=3$	e_{old} e_{new} $l_{\min}$ MWE l_{new}			$\{f,c\}=3$ ×	$\{f,d\}=2$ ×	$\{c,e\}=1$ $\{f,e\}=4$ $min(4,7)$ $\{c,e\}=1$ 4	$\{f,g\}=2$ ×		$\{g,h\}=2$ $\{f,h\}=1$ $min(4,4)$ $\{g,h\}=2$ 4	$\{c,e\}=1$
$v_6=e$ $l(e)=4$	e_{old} e_{new} $l_{\min}$ MWE l_{new}		$\{e,b\}=4$ ×	$\{e,c\}=1$ ×			$\{e,f\}=2$ ×		$\{g,h\}=2$ $\{e,h\}=3$ $min(4,7)$ $\{g,h\}=2$ 4	$\{g,h\}=2$

만약 복잡한 도시의 목적지까지 최단경로를 찾는 경우, GPS 시스템에 Dijkstra SP 알고리즘을 적용하면 실시간으로 정보를 제공할 수 없는 경우도 발생할 수 있다. 따라서 실시간으로 정보를 제공하기 위해서는 수행 횟수를 단축시킬 수 있는 알고리즘이 요구된다.

방향 그래프의 점대점 최단경로 탐색 알고리즘은 Lee[10]이, 레벨 노드 선택에 기반한 점대점 최단경로 알고리즘은 Lee[11]이, 주행시간에 기반한 실시간 점대점 최단경로 탐색 알고리즘은 Lee[12]가 제안하였다. Lee[12]의 주행시간에 기반한 실시간 점대점 최단경로 탐색 알고리즘은 레벨 노드 단위의 최소 주행시간을 선택하는 방식이다.

3장에서는 무방향 그래프의 점대점 최단거리를 찾기 위해 알고리즘 수행에 요구되는 메모리 크기도 줄이면서 수행속도를 월등히 향상시킬 수 있는 MST에 기반한 알고리즘을 제안한다.

III. MST-SP 알고리즘

최소신장트리 (MST)를 이용하여 무방향 그래프의 점대점 최단거리 (SP)를 빠르게 찾는 제안된 알고리즘을 "MST-SP 알고리즘"이라 하자. 무방향 그래프를 양방향 그래프로 취급한다. 따라서 정점을 노드로, 간선을 호로 부르며, 간선 $\{i,j\}$는 순서쌍 (i,j)와 (j,i)로 표현한다. 먼저, 그래프의 노드들을 그림 2와 같이 S,P,P_d와 D의 4 그룹으로 분류한다. 여기서 S는 출발 노드, P는 출발에서 목적지까지의 경로 노드, P_d는 경로 노드들 중 목적지 노드로 유입되는 노드, D는 목적지 노드이다.

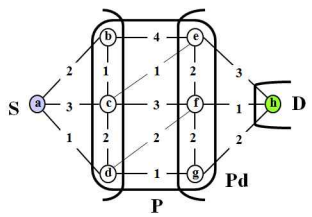


그림 2. G_1 그래프의 노드 분류
Fig. 2. Node classification for G_1 graph

MST-SP 알고리즘은 출발 (S)과 목적지 노드 (D) 가 결정되었을 때, "최단경로 설정", "최단경로 탐색"과 "최단경로 선택" 과정을 거친다. 노드 "최단경로 설정"은 Borůvka의 MST 알고리즘[13,14]의 간선 선택 방법을 변형한 형태를 적용하였다. Borůvka의 MST 알고리즘[13,14]은 그래프의 모든 정점의 유출 최소 가중치 간선 (MWE)을 선택한다. 출발에서 목적지 노드까지의 경로를 설정하기 위한 "최단경로 설정"은 3단계를 거친다. 1단계는 "목적지 노드 S의 유입 최소 가중치 호 선택"으로 D로 유입되는 호 (P_d 유출 호)의 최소 가중치 2개를 선택한다. 이는 정방행렬의 D열, P_d행에서 선택될 수 있다. 2단계는 "경로 노드의 유출 호 선택"으로 경로 노드 (P) 행의 $P \cup P_d$열에 대해 최소 가중치를 갖는 호를 2개씩 선택한다. 3단계는 "경로 노드의 유입 호 선택"으로 $P \cup P_d$열의 P 행에 대해 호가 선택되지 않았으면 최소 가중치 호를 2개 선택하며, 만약 1개만 선택하였으면 나머지 최소 가중치 호를 1개 더 선택한다. 이와 같이 하는 이유는 각 경로 노드들의 유입과 유출 최소 가중치 호들로 경로를 설정하여 출발에서 목적지 노드까지 단절 없이 최단경로를 갖도록 하기 위함이다. 방향 그래프의 경우 각 노드의 유출과 유입 최소 가중치 호를 단 1개만 선택하면 된다. 그러나 양방향 그래프는 대칭성을 갖고 있어 유입 최소 가중치 호와 유출 최소 가중치 호가 중복 선택되는 경우가 많이 발생하기 때문에 2개씩을 선택한다.

다음으로, "최단경로 탐색"은 Prim MST 알고리즘[8]을 간선현한 형태를 적용한다. Prim MST 알고리즘은 임의의 정점 (출발)을 선택하고 이 정점의 유출 간선들에 대해 최소 가중치 간선을 선택하면 최소 가중치를 갖는 부분 신장트리 (PSP, partial spanning tree)가 형성되며, 부분 신장트리의 정점들 (방문한 정점들)의 선택되지 않은 간선들 중에서 최소 가중치를 갖는 간선을 선택하면서 신장트리를 확장해 나가는 방식이다.

MST-SP 알고리즘의 "최단경로 탐색"은 첫 번째로, 출발 노드에 인접한 노드들에 대해 각 노드의 유입 호 경로 길이를 계산하여 최소 길이를 갖는 호를 선택한다. 만약 길이가 동일하면 모두 선택한다. 두 번째로, 새로 추가된 노드들의 유출 호들에 대해 "노드 최소 경로 설정"에서 선택된 호들과 일치하는 호들만 선택하고 나머지는 삭제하는 방식으로 노드들을 확장한다. 목적지 노드의 유입 최소 가중치 호 2개가 선택될 때까지 알고리즘이 수행된다. 이 과정에서 방문한 노드들로 유출되는 호는 사이클이 발생한 경우로 삭제한다. 최종적으로 목적지 노드까지 연결된 경로들에 대해 최단경로를 갖는 경로를 선택하면 SP를 얻을 수 있다. MST-SP 알고리즘의 상세한 내용은 그림 3에 제시되어 있다.

무방향 그래프 간선 $\{i,j\}$를 방향 그래프의 호 (i,j)와 (j,i)로 표현
각 간선 가중치에 대해 $|n|\times|n|$ 인접행렬 작성
【최단경로 설정 】 /* 정방행렬 이용 */
1. 목적지 노드 "유입 호" 선택 (D열의 P_d행)
   1st와 2nd MWA $[(i,d)$와 $(j,d)]$ 선택 (2개 이상)
2. 경로 노드 "유출 호" 선택 (P행의 $P\cup P_d$열)
   1st와 2nd MWA 선택 (단, 동일 가중치 모두 선택)
   if 1st MWA $\geq$ 2개 then 2nd MWA 선택 안함
   else if 1st MWA $<$ 2개 then 2nd MWA 선택
3. 경로 노드 "유입 호" 선택 ($P\cup P_d$ 열의 P행)
   if P행에서 선택된 호의 수 ≥ 2 then 선택 안함
   else if P행에서 선택된 호의 수 <2 then 2개 MWA 선택
   if 1st MWA $\geq$ 2개 then 2nd MWA 선택 안함
   else if 1st MWA $<$ 2개 then 2nd MWA 선택
【최단경로 탐색-1st Step】 /* 출발 경로 선택
출발 노드를 SP Nodes (n_{sp})에 추가
① 출발 노드 (S)행 : 모든 호 (i,j,k)에 대해 최소 경로 길이 선택
(해당 연결 호가 없으면 ∞로 설정, 동일 경로 길이 모두 선택)
$\min[(s,i),(s,j)+(j,i)]$, $\min[(s,j),(s,i)+(i,j),(s,k)+(k,j)]$,
$\min[(s,k),(s,j)+(j,k)]$
최소 가중치 호들을 SP Arcs에 저장, 노드를 SP Nodes (n_{sp})에 추가
【최단경로 탐색-2nd Step】/* 경로 노드 선택
② n_{sp}에 새로 추가된 노드의 유출 호들을 Candidate Arcs에 추가 $(a_c=a_a)$
if Candidate Arcs (i,j)의 $j\in n_{sp}$ then 해당 호 (a_d) 삭제
③ $a_c(a_c=a_a-a_d)$ 와 a_{mwa} 비교
if $a_c=a_{mwa}$ then SP Arcs (a_{sp})에 저장, a_c에서 삭제
else if $a_c\neq a_{mwa}$ then a_c에서 삭제
a_{sp}에 저장된 호 (i,j)의 "j" 노드를 n_{sp}에 저장
if $(i,d)\in a_{sp}$ and $(j,d)\in a_{sp}$ then ④ 수행
/*목적지 노드 유입 2개 MWA가 모두 선택된 경우 */
else if $(i,d)\in a_{sp}$ or $(j,d)\in a_{sp}$ then ②로 복귀
(단, $j\in n_{sp}$인 경우 사이클이지만 삭제하지 않음)
else if $(i,d)\notin a_{sp}$ and $(J,d)\notin a_{sp}$ then ②로 복귀
【최단경로 선택】
④ a_{sp}의 호들을 연결, 목적지 연결 경로에 대해 $\min[l,(s,d)]$ 선택

그림 3. MST-SP 알고리즘
Fig. 3. MST-SP algorithm

그림 1의 G_1 그래프에 MST-SP 알고리즘을 적용한 과정은 표 2에, 적용 결과 얻은 SP는 그림 4에 제시되어 있다.

표 2. G_1 그래프의 MST-SP 알고리즘
Table 2. MST-SP algorithm for G_1 graph

경로 MWA (a_{mwa}) 대상	선택
1. [열] 목적지 노드 (ⓗ): 1st와 2nd MWA 선택 (e,h)=3, (f,h)=1, (g,h)=2 중 (f,h)=1,(g,h)=2 선택	
2. [행] 경로 노드: 출발(ⓐ), 목적지 (ⓗ)와 목적지 연결 노드 ⓔ,ⓕ,ⓖ 제외 대상 : 행 경로 노드 ⓑ,ⓒ,ⓓ에 대해 2개 MWA 선택 1st MWA 2nd MWA ⓑ : (b,c)=1 (b,e)=4 ⓒ : (c,b)=1,(c,e)=1 - /* 1st MWA 2개로 2nd MWA 선택안함 */ ⓓ : (d,g)=1 (d,c)=2, (d,f)=2 /* 1st MWA (d,a)=1은 출발 노드로 연결되어 선택 안함 */	(f,h)=1 (g,h)=2 (b,c)=1 (b,e)=4 (c,b)=1 (c,e)=1 (c,f)=3 (d,c)=2 (d,f)=2 (d,g)=1
3. [열] 열 경로 ⓑ,ⓒ,ⓓ,ⓔ,ⓕ,ⓖ 노드에 대한 ⓑ,ⓒ,ⓓ 행 노드 : 2개 MWA 선택 ⓑ (c,b)=1만 존재, 행 경로 노드에서 이미 선택 ⓒ (b,c)=1,(d,c)=2가 존재, 행 경로 노드에서 이미 선택 ⓓ (c,d)=2가 존재하며, 선택되지 않았으므로 선택 ⓔ (b,e)=4와 (c,e)=1은 행 경로 노드에서 이미 선택 ⓕ (c,f)=3과 (d,f)=2가 존재하며, (d,f)=2는 행 노드에서 선택되어 (c,f)=3을 선택함 ⓖ (d,g)=1이 존재하며, 행 경로 노드에서 이미 선택	

A	SP Nodes	Candidate Arcs (a_c) a_a	a_d	a_a-a_d	a_{mwa}	SP Arcs
{b,c,d,e, f,g,h}	{a}	**(a,b)=2**, (a,c)=3 **(a,d)=1**	min((a,b)=2,(a,c)+(c,b)=4) min((a,c)=3,(a,b)+(b,c)=3, (a,d)+(d,c)=3) min((a,d)=1,(a,c)+(c,d)=2)	-	**(a,b)=2 (a,c)=3 (a,d)=1 (b,c)=1 (d,c)=2**
{e,f,g,h}	{a,b,c,d}	**(b,a)=2**, **(b,c)=1** (b,e)=4, **(c,a)=3** **(c,b)=1**, **(c,d)=2** (c,e)=1, (c,f)=3 **(d,a)=1**, **(d,c)=2** (d,f)=2, (d,g)=1	(b,a)=2, (b,c)=1 (c,a)=3, (c,b)=1 (c,d)=2, (d,a)=1 (d,c)=2	**(b,e)=4**, **(c,e)=1** **(c,f)=3**, **(d,f)=2** **(d,g)=1**	(f,h)=1 (g,h)=2 (b,c)=1 (b,e)=4 (c,b)=1 (c,e)=1 (c,f)=3 (d,c)=2 (d,f)=2 (d,g)=1	**(b,e)=4 (c,e)=1 (c,f)=3 (d,f)=2 (d,g)=1**
{h}	{a,b,c,d ,e,f,g}	**(e,b)=4,** **(e,c)=1** **(e,f)=2,** (e,h)=3 **(f,c)=3**, **(f,d)=2** **(f,g)=2**, (f,h)=1 **(g,d)=1**, **(g,f)=2** (g,h)=2	(e,b)=4, (e,c)=1 (e,f)=2, (f,c)=3 (f,d)=2, (f,g)=2 (g,d)=1, (g,f)=2	(e,h)=3, **(f,h)=1** **(g,h)=2**		**(f,h)=1** **(g,h)=2**

알고리즘 적용 결과, Dijkstra SP 알고리즘은 1개의 경로를 얻는데 반해, MST-SP 알고리즘은 2개 경로를 얻는데 성공하였다. 알고리즘 수행 횟수 측면에서 볼 때, Dijkstra SP 알고리즘은 $n-1=7$회를 수행한데 비해 MST-SP 알고리즘은 3회로 단축시킬 수 있었으며, 2개의 최단경로를 찾는데 성공하였다. MST-SP 알고리즘은 a_{mwa}, SP Nodes, SP Arcs와 a_c만이 요구되어 Dijkstra SP 알고리즘에 비해 월등히 적은 메모리를 사용함을 알 수 있다.

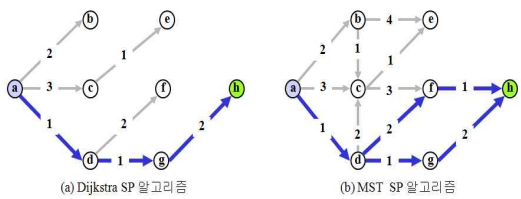


그림 4. G_1 그래프의 최단경로
Fig. 4. Shortest path for G_1 graph

IV. 실험 및 결과 분석

본 장에서는 그림 5와 같이 8개 그래프를 대상으로 알고리즘의 적용성을 평가해 본다. G_2, G_3와 G_7그래프는 Chen[9]에서, G_4와 G_5그래프는 Wenger[15]에서, G_6과 G_8 그래프는 Peiper[16]에서 인용되었다. G_9는 Lim과 Kim[17]에서 인용되었으며, 미국 South Dakota 소재의 Sioux Fall 대학

의 네트워크이다. 실험 데이터들에 대해 Dijikstra SP 알고리즘과 MST-SP 알고리즘으로 최단경로를 구한 결과는 그림 6에 제시되어 있다.

실험 데이터들에 대해 Dijikstra SP 알고리즘과 MST-SP 알고리즘으로 최단경로를 구한 결과는 그림 6에 제시되어 있다.

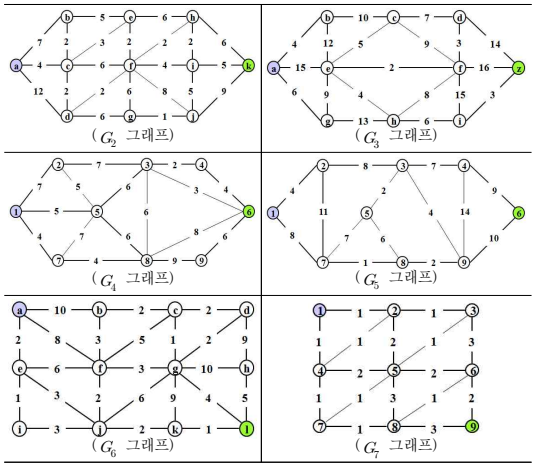
G_2 그래프는 ⓐ에서 ⓚ로의 최단경로를 찾는 문제로, 알고리즘 적용 결과 Dijkstra SP 알고리즘은 10회 수행인데 반해, MST-SP 알고리즘은 4회 수행으로 Dijkstra SP 알고리즘과 동일한 최단경로를 찾았다.

G_3 그래프는 ⓐ에서 ⓩ로의 최단경로를 찾는 문제로, 알고리즘 적용 결과 Dijkstra SP 알고리즘은 9회 수행되었다. 반면에 MST-SP 알고리즘은 4회 수행으로 2개의 최단경로를 찾았다.

G_4 그래프는 ①에서 ⑥으로의 최단경로를 찾는 문제로, 알고리즘 적용 결과 Dijkstra SP 알고리즘은 8회인데 반해, MST-SP 알고리즘은 3회 수행으로 Dijkstra SP 알고리즘과 동일한 최단경로를 찾았다.

G_5 그래프는 ①에서 ⑥으로의 최단경로를 찾는 문제로, 알고리즘을 적용 결과 Dijkstra SP 알고리즘은 8회인데 반해, MST-SP 알고리즘은 4회 수행으로 Dijkstra SP 알고리즘과 동일한 최단경로를 찾았다.

G_6 그래프는 ⓐ에서 ①로의 최단경로를 찾는 문제로, 알고리즘을 적용 결과 Dijkstra SP 알고리즘은 11회인데 반해 MST-SP 알고리즘은 4회 수행으로 Dijkstra SP 알고리즘과 동일한 최단경로를 찾았다.



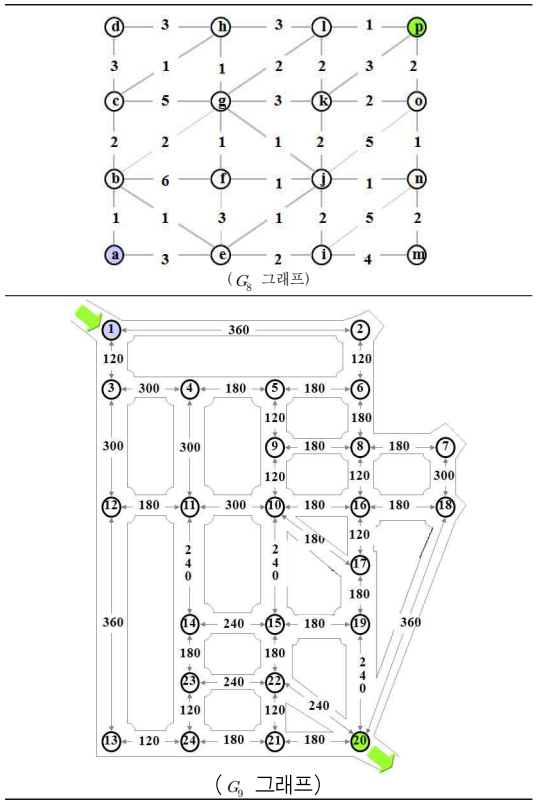
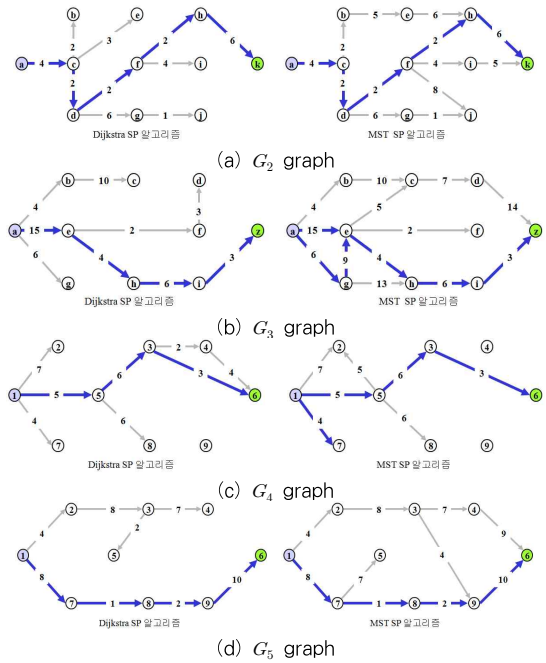


그림 5. 실험에 적용된 그래프
Fig. 5. Experimental graph



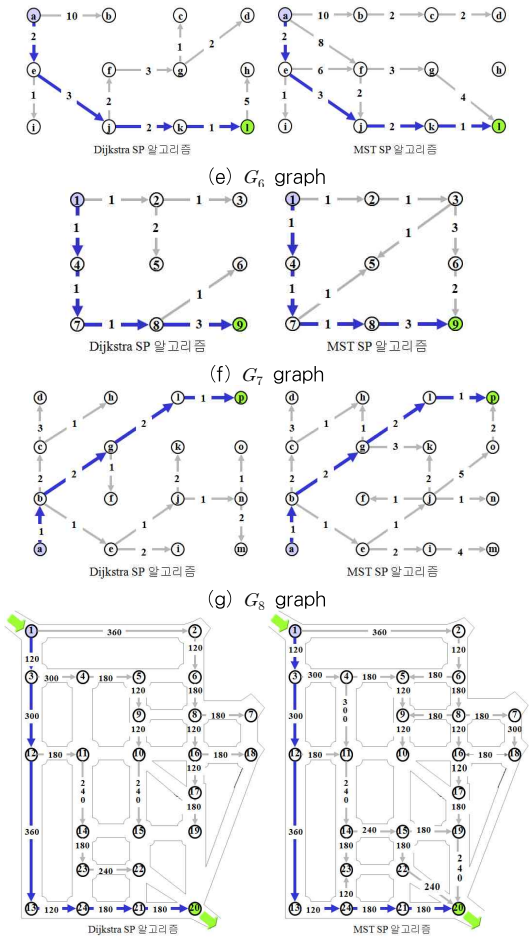


그림 6. 최단경로
Fig. 6. Shortest path

G_7 그래프는 ①에서 ⑨로의 최단경로를 찾는 문제로, 알고리즘 적용 결과 Dijkstra SP 알고리즘은 8회인데 반해, MST-SP 알고리즘은 4회 수행으로 Dijkstra SP 알고리즘과 동일한 최단경로를 찾았다.

G_8 그래프는 ⓐ에서 ⓟ로의 최단경로를 찾는 문제로, 알고리즘 적용 결과 Dijkstra SP 알고리즘은 15회인데 반해, MST-SP 알고리즘은 4회 수행으로 Dijkstra SP 알고리즘과 동일한 최단경로를 찾았다.

G_9 그래프는 ①에서 "20"으로의 최단경로를 찾는 문제로, 알고리즘 적용 결과 Dijkstra SP 알고리즘은 23회인데 반해, MST-SP 알고리즘은 7회 수행으로 Dijkstra SP 알고리즘과 동일한 최단경로를 찾았다.

본 장에서 8개 그래프에 MST-SP 알고리즘을 적용한 결과 모든 그래프에서 최단경로를 1개 또는 2개를 찾는데 성공하였으며, Dijkstra SP 알고리즘에 비해 수행 횟수를 크게 줄이는 효과를 얻었다.

본 논문에서 알고리즘 적용에 활용된 9개 그래프에 대해 Dijkstra SP 알고리즘과 MST-SP 알고리즘의 수행횟수를 비교한 결과는 표 3에 제시하였다. MST-SP 알고리즘은 Dijkstra SP 알고리즘에 비해 동일한 SP를 얻으면서, 평균적으로 알고리즘 수행 횟수를 60% 단축시키는 효과를 얻었다. 또한, 그래프의 노드수가 큰 G_9 그래프는 약 70%까지 단축시킴을 알 수 있다. 따라서 실시간 GPS 항법 시스템에서 최단경로를 찾아 제공하는데 본 논문에서 제안된 알고리즘을 활용하면 고객의 만족을 향상시킬 수 있을 것이다.

표 3. 최단경로 탐색 알고리즘 성능 비교
Table 3. Compare with performance of SP search algorithm

그래프	$\|v\|$	$\|e\|$	Dijkstra SP 알고리즘		MST-SP 알고리즘		
			수행횟수	$l(d)$	수행횟수	$l(d)$	비율
G_1	8	15	7	4	3	4	42.9%
G_2	11	22	10	16	4	16	40.0%
G_3	10	19	9	28	4	28	44.4%
G_4	9	16	8	14	3	14	37.5%
G_5	9	14	8	21	4	21	50.0%
G_6	12	23	11	8	4	8	36.4%
G_7	9	16	8	6	4	6	50.0%
G_8	16	33	15	6	4	6	26.7%
G_9	24	38	23	1260	7	126 0	30.4%
평균							**39.8%**

V. 결론 및 향후 연구과제

본 논문에서는 실시간 GPS 항법 시스템에 일반적으로 적용되고 있는 Dijkstra SP 알고리즘의 문제점을 고찰해보고 MST에 기반한 새로운 최단경로 탐색 알고리즘을 제안하였다. Dijkstra SP 알고리즘은 방향 그래프에 대해 출발 노드에서 그래프의 모든 노드로의 최단경로를 찾는다. 그러나 GPS 항법 시스템은 출발에서 목적지 노드까지의 점대점 최단경로를 찾아 경로 정보를 제공해야만 한다. 이 시스템은 실시간으로 최단경로 정보를 고객에게 제공해야 하기 때문에 최단경로를 탐색하는 알고리즘의 수행 속도가 가장 중요한 요인으로 작용할 수 있다.

본 논문에서는 무방향 그래프에 대해 Dijkstra SP 알고리즘을 적용하여 보고, 무방향 그래프를 양방향 그래프로 취급하여 최단경로를 빠르게 찾는 알고리즘을 제안하였다. 제안된 알고리즘은 "경로 노드에 대해 최단경로를 설정"하고 이 경로와 일치하는 호들을 선택함으로서 한 번에 다수의 노드들을

방문하는 방법을 택하였다. 방향그래프의 호는 순서쌍 특징으로 인해 최단경로 설정은 각 노드의 유출과 유입 최소 가중치 호를 1개씩 설정하면 된다. 그러나 무방향 (양방향) 그래프의 간선은 동일한 간선이 중복 선택될 수 있기 때문에 유입과 유출 최소 가중치 간선을 1개씩만 선택하면 출발에서 목적지까지의 경로를 설정하지 못할 수 도 있기 때문이다. 따라서 2개씩 선택하는 방법을 택하였다. 이와 같은 방법을 적용한 결과 실험에 적용된 9개 그래프 모두에서 최단경로를 찾는데 성공하였다. 또한 Dijkstra SP 알고리즘보다 메모리도 적게 요구하면서, 약 60%의 수행 횟수를 단축시키는 효과도 얻었다.

본 논문에서는 무방향 그래프의 최단경로 탐색 알고리즘을 가상의 그래프와 실제 그래프에 대해 최단경로를 찾는지에 대한 이론적으로 고찰하였으며, 실제 GPS 항법 시스템에 적용하여 실용성을 검증하지는 못하였다. 따라서 추후 실제 GPS 시스템에 적용하여 얼마나 빨리 최단경로를 탐색하여 고객 만족도를 향상시킬 수 있는지 검증하고자 한다.

참고문헌

〔1〕 M. Abboud, L. Mariya, A. Jaoude, and Z. Kerbage, "Real Time GPS Navigation System," 3rd FEA Student Conference, Department of Electrical and Computer Engineering, American University of Beirut, 2004.

〔2〕 T. H. Cormen, C. E. Leserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms," 2nd Edition, MIT Press and McGraw-Hill, 2001.

〔3〕 E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," Numerische Mathematik, Vol. 1, No. 1, pp. 269-271, 1959.

〔4〕 Wikipedia, "Dijkstra's Algorithm," http://en.wikipedia.org/wiki/Dijkstra_algorithm, Wikimedia Foundation Inc., 2007.

〔5〕 J. Misra, "A Walk Over the Shortest Path: Dijkstra's Algorithm Viewed as Fixed-Point Computation," Department of Computer Science, University of Texas at Austin, USA, 2000.

〔6〕 Y. T. Lim and H. M. Kim, "A Shortest Path Algorithm for Real Road Network Based on Path Overlap," Department of Civil Engineering, Institute of Transportation Studies, University of California, Irvine, USA, 2005.

〔7〕 F. B. Zhan, "Three Fastest Shortest Path Algorithms on Real Road Networks: Data Structures and Procedures," Journal of Geographic Information and Decision Analysis, Vol. 1, No. 1, pp. 69-82, Feb. 1997.

〔8〕 R. C. Prim, "Shortest Connection Networks and Some Generalisations," Bell System Technical Journal, Vol. 36, pp. 1389-1401, Nov. 1957.

〔9〕 W. W. L. Chen, "Discrete Mathematics," Department of Mathematics, Division of ICS, Macquarie University, Australia, 2003.

〔10〕 S. U. Lee, "A Point-to-Point Shortest Path Search Algorithm for Digraph," Journal of Korean Institute of Intelligent Systems, Vol. 17, No. 7, pp. 893-900, Dec. 2007.

〔11〕 S. U. Lee, "A Point-to-Point Shortest Path Algorithm Based on Level Node Selection," Journal of the Institute of Internet, Broadcasting and Communication, Vol. 12, No. 1, pp. 133-140, Feb. 2012.

〔12〕 S. U. Lee, "A Real-time Point-to-Point Shortest Path Search Algorithm Based on Traveling Time," Journal of the Institute of Internet, Broadcasting and Communication, Vol. 12, No. 4, pp. 131-140, Aug. 2012.

〔13〕 O. Borůvka, "O Jistem Problemu Minimalnim," Prace Mor. Prrodved. Spol. V Brne (Acta Societ. Natur. Moravicae), Vol. Ⅲ, No. 3, pp. 37-58, 1926.

〔14〕 J. Nešetřil, E. Milková, and H. Nešetřilová, "Otakar Borůvka on Minimum Spanning Tree Problem (Translation of the both 1926 Papers, Comments, History)," DMATH: Discrete Mathematics, Vol. 233, No. 1-3, pp. 3-36, Apr. 2001.

〔15〕 R. Wenger, "CIS 780: Analysis of Algorithms," http://www.cse.ohio-state.edu/~wenger/cis780/shortest_path.pdf, 2004.

〔16〕 C. Peiper, CS 400 - Data Structures for Non CS-Majors," http://www.cs.uiuc.edu/class/fa05/cs400/_labs/Lab12/suuri/, 2005.

〔17〕 Y. T. Lim and H. M. Kim, "A Shortest Path

Algorithm for Real Road Network Based on Path Overlap," Journal of the Eastern Asia Society for Transportation Studies, Vol. 6, pp. 1426-1438, 2005.

저 자 소 개

이 상 운(Sang-Un, Lee)
1983년 ~ 1987년 :
한국항공대학교 항공전자공학과 (학사)
1995년 ~ 1997년 :
경상대학교 컴퓨터과학과 (석사)
1998년 ~ 2001년 :
경상대학교 컴퓨터과학과 (박사)
2003.3 ~ 현 재 :
강릉원주대학교 멀티미디어공학과 부교수
관심분야 : 소프트웨어 프로젝트 관리,
소프트웨어 개발 방법론,
소프트웨어 신뢰성,
그래프 알고리즘
e-mail : MST@gwnu.ac.kr