The Journal of IIBC. Vol. 14, No. 1, pp.51-59, Feb. 28, 2014. pISSN 2288-0879, eISSN 2288-2618

http://dx.doi.org/10.7236/JIIBC.2014.14.1.51

JIIBC 2014-1-7

간선 모집단 규모축소 기법을 적용한 빠른 최소신장트리 결정

Fast Determination of Minimum Spanning Tree Based on Down-sizing Technique of Edges Population

이상운*, 최명복**

Sang-Un Lee*, Myeong-Bok Choi**

요 약 본 논문은 최소신장트리를 보다 빠르게 구하기 위해 그래프의 간선 모집단을 축소시키는 방법을 제안하였다. 기존의 최소신장트리 알고리즘은 그래프의 모든 간선을 대상으로 한다. 반면에, 제안된 알고리즘은 사전에 결합가가 3 이상인 정점에 대해 최대 가중치 간선을 삭제하는 방법을 적용하여 간선 모집단 크기를 축소시킨다. 다음으로 축소된 간선 모집단을 대상으로 Borůvka, Prim, Kruskal과 역-삭제 알고리즘을 최적으로 종료시키는 종료시점 기준을 적용하였다. 9개 그래프에 제안된 알고리즘을 적용한 결과 MST에 기여를 하지 못하는 간선을 사전에 평균 38% 축소시킬 수 있었다. 또한, 원래의 그래프를 대상으로 하는 경우와 비교 결과 알고리즘에서 비교되는 간선의 수를 Borůvka는 38%, Prim은 37%, Kruskal은 39%, 역-삭제 알고리즘은 73%를 단축시켜 신속하게 최소신장트리를 구하였다.

Abstract This paper suggests a method of lessening number of a graph's edges population in order to rapidly obtain the minimum spanning tree. The present minimum spanning tree algorithm works on all the edges of the graph. However, the suggested algorithm reduces the edges population size by means of applying a method of deleting maximum weight edges in advance from vertices with more than 2 valencies. Next, it applies a stopping criterion which ideally terminates Borůvka, Prim, Kruskal and Reverse-Delete algorithms for reduced edges population. On applying the suggested algorithm to 9 graphs, it was able to minimize averagely 83% of the edges that do not become MST. In addition, comparing to the original graph, edges are turned out to be lessened 38% by Borůvka, 37% by Prim, 39% by Kruskal and 73% by Reverse-Delete algorithm, and thereby the minimum spanning tree is obtained promptly.

Key Words : Minimum Spanning Tree, Valency, Edges Population, Stopping Criteria

Ⅰ. 서 론

그래프 $G=(V,E)$는 정점들 (Vertices, v)과 간선들 (Edges, e)로 구성되어 있으며, 일반적으로 그래프란 모든 정점들을 연결하는 간선들이 무방향성 (Undirected)을 갖고 있는 무방향 그래프 (Undirected Graph)를 의미한다. 무방향 그래프의 간선들이 가중치 (Weight)를 갖고 있는 경우 이를 가중 그래프라 한다. 가중 그래프에서

*정회원, 강릉원주대학교 멀티미디어공학과
**종신회원, 강릉원주대학교 멀티미디어공학과
접수일자 2013년 10월 18일, 수정완료 2013년 12월 26일
게재확정일자 2014년 2월 7일

Received: 18 October, 2013 / Revised: 26 December, 2013
Accepted: 7 February, 2014
*Corresponding Author: cmb5859@gmail.com
Dept. of Multimedia Eng., Gangneung-Wonju National University, Korea

모든 정점들이 간선들로 연결되어 있고, 사이클이 발생하지 않으면서 간선들의 가중치 합 ($\Sigma w(e)$)이 최소가 되는 트리를 최소신장트리(Minimum Spanning Tree, MST)라 한다. 즉, MST에서 사이클이 발생하지 않도록 모든 정점들을 간선들로 연결하기 위해서는 $|e|=|v|-1$가 선택되어야만 한다. MST는 전기, 전화, 가스 또는 수도 등의 분야에 활용되고 있다.[1]

MST를 찾는 대표적인 알고리즘으로 Borůvka[2,3], Prim[4], Kruskal[5]과 역-삭제 (Reverse-Delete) 알고리즘[6]이 있다. 이들 알고리즘은 모든 간선들의 가중치가 서로 다르다(Distinct)는 가정에 기반하고 있다. 그러나 현실적으로 동일한 가중치 값을 가진 간선들이 다수 포함될 수 있다. 또한, 그래프의 모든 간선들을 모집단(Population)으로 하여 선택된 간선 $|v|-1$개의 가중치 합이 최소가 되는 MST를 찾는다.

MST 알고리즘은 욕심쟁이 알고리즘 (Greedy Algorithm)으로 한번에 하나의 정점이나 간선을 선택하는 방식을 채택하고 있다. 위 4개 알고리즘 모두 수행 속도에는 차이가 별로 없어 어떤 알고리즘이 가장 좋은지 우열을 판가름하기 어렵다.

Borůvka 알고리즘[2,3]은 MST를 찾는 알고리즘으로 1926년에 처음 제안되었으며, 현재는 1950년대에 제안된 Prim과 Kruskal 알고리즘이 널리 사용되고 있다. 역-삭제 알고리즘은 Kruskal 알고리즘을 역으로 수행하는 방식으로 현재에는 일반적으로 사용되지 않고 있다.[1] Borůvka와 역-삭제 알고리즘은 시각적으로 그래프를 보면서 MST를 찾는 그래픽 방법 (Graphical Method)이 적합하며, Prim과 Kruskal 알고리즘은 그래픽 방법 보다는 프로그램으로 구현한 계산 방법 (Computational Method)으로 MST를 보다 쉽게 찾을 수 있다.

원 그래프는 정점들이 간선들로 복잡하게 연결되어 있어 $|v|-1$개의 MST 간선을 찾는데 시간이 많이 요구되는 문제점을 갖고 있다. 이 문제점을 해결하기 위해 본 논문은 그래프의 간선 개수 $|e|$를 축소시키는 방법을 제안하고, 다음으로 각 알고리즘의 최적의 종료 시점을 제시한다.

2장에서는 Borůvka, Prim, Kruskal과 역-삭제 알고리즘 수행 방법을 고찰해보고, 실제 그래프에 적용하여 MST를 찾고 문제점을 고찰해 본다. 3장에서는 MST 알고리즘의 수행 횟수를 단축시키는 방법을 제시하고, 각 알고리즘의 종료 기준과 사이클을 확인하는 방법을 제안한다. 4장에서는 실제 그래프들을 대상으로 제안된 방법이 알고리즘 수행 횟수를 얼마나 단축시킬 수 있는지 평가한다.

II. 관련 연구와 연구 배경

1. MST 알고리즘

그래프 $G=(V,E)$에서 간선은 무방향으로 $\{i,j\}$로 표기한다.[7-9] 또한 간선의 가중치 $w\{i,j\}$를 편의상 $\{i,j\}$로 표기한다. 그래프의 정점 개수를 $|v|$, 간선 개수를 $|e|$라 하자. 신장트리는 $|v|$ 개의 정점을 사이클이 발생하지 않게 간선들로 모두 연결하는 경우이므로 간선의 수는 $|v|-1$개로 구성되어 있다.

Borůvka 알고리즘은 2개의 Stage를 수행한다.

(1st Stage) 그래프의 각 정점 i에서 중복에 무관하게 최소 가중치 간선을 선택한다. 선택된 간선들을 연결하여 부분신장트리 (Partial Spanning Tree, PSP) 그룹을 형성한다. 이 과정에서 사이클 (루프)이 발생하는 간선들을 삭제한다.

(2nd Stage) 그룹이 1개이면 알고리즘을 종료한다. 그렇지 않으면 이들 그룹들을 하나의 트리로 만들기 위해 그룹들을 상호 연결하는 최소 가중치 간선을 선택한다.

Prim 알고리즘은 다음과 같이 수행된다.

(1) 임의의 정점 i를 선택한다.

(2) 정점 i에 연결된 간선들 중에서 최소 가중치 간선 $\{i,j\}$를 선택한다.

(3) 정점 j에 연결된 간선들과 정점 i에서 선택하지 않고 남아 있는 간선 들 중에서 최소 가중치 간선 $\{j,k\}$를 선택한다. 단, 이 과정에서 정점 i,j를 연결하는 간선은 사이클이 발생하므로 무시한다.

(4) 기존에 방문한 정점들에서 선택되지 않고 남아 있는 간선들과 새로 선택된 정점 k에 연결된 모든 정점들에 대해 최소 가중치 간선을 선택한다. (단, 이 과정에서 사이클이 발생하는 간선은 무시한다.) 이 과정을 모든 정점을 방문할 때 까지 수행한다.

Kruskal 알고리즘은 다음과 같이 수행된다.

(1) E에 있는 모든 가중치를 갖는 간선들을 오름차순으로 정렬시킨다.

(2) 최소 가중치를 갖는 간선을 선택하여 부분 신장트리 집합 T를 구성한다.

(3) E에서 다음 최소 가중치 간선의 두 정점이 T의 한 집합에 존재하면 사이클이 발생하는 경우로 선택하지 않으며, 그렇지 않은 경우 선택한다.

(4) E에서 더 이상 선택할 간선이 없을 때까지 (3)을 반복적으로

수행하고 알고리즘을 종료한다.

역-삭제 알고리즘은 Kruskal 알고리즘을 역으로 수행하는 방식으로 최대 가중치 간선을 한번에 하나씩 제거하며 다음과 같이 수행된다.

(1) T는 모든 정점들과 간선들로 구성되어 있다.
(2) E에 있는 모든 가중치를 갖는 간선들을 내림차순으로 정렬시킨다.
(3) 최대 가중치를 갖는 간선을 한번에 하나씩 선택하여 해당 간선을 T에서 삭제하였을 때 T의 임의의 정점을 그래프에서 분리 (Disconnect)시키지 않으면 삭제한다.
(4) E에서 더 이상 선택할 간선이 없을 때까지 (3)을 반복적으로 수행하고 알고리즘을 종료한다.

Prim과 Kruskal 알고리즘은 계산 방법으로 그래프를 대상으로 시각적으로 해답을 얻는 그래픽 방법 보다는 컴퓨터 프로그램으로 구현하는 것이 보다 쉽게 해답을 얻을 수 있다. 반면에 Borůvka와 역-삭제 알고리즘은 계산방법보다는 시각적으로 그래프를 보면서 쉽게 해답을 찾을 수 있다. Borůvka의 1st Stage에서의 사이클 확인과 2nd Stage에서 부분신장트리들 간을 연결하는 최소 가중치 간선을 찾기 위한 방법은 결국 Kruskal 알고리즘을 적용해야 계산방법으로 구현할 수 있다. 또한, 역-삭제 알고리즘에서 해당 간선을 삭제하였을 때 임의의 정점을 그래프에서 분리시키지 않는지 확인하는 방법은 깊이우선탐색 (Depth-First Search, DFS)으로 사이클을 탐색하는 과정이 필요하다.

2. MST 알고리즘 적용

본 절에서는 MST 알고리즘을 그래픽 모델과 계산 모델로 분류하여 그림 1의 G_1 그래프를 대상으로 MST를 쉽게 찾는 방법을 고찰해 본다. G_1 그래프는 Wikipedia[1]에서 인용되었다. MST 알고리즘을 적용하기 위해 본 논문에서는 간선 가중치의 정방행렬을 이용한다.

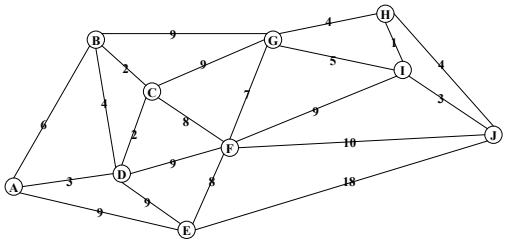
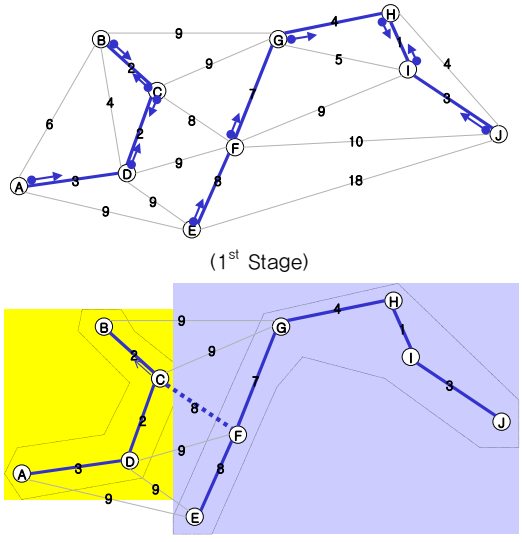


그림 1. G_1 **그래프**
Fig. 1. Graph G_1

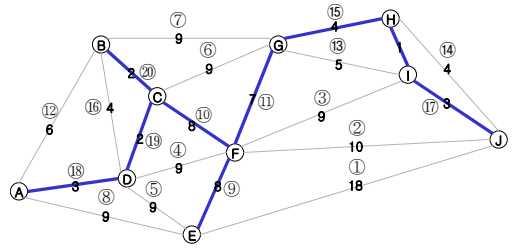
G_1 그래프에 대해 그래픽 방법으로 MST를 찾는 Borůvka와 역-삭제 알고리즘을 적용한 결과는 그림 2에, 계산 방법으로 MST를 찾는 Prim과 Kruskal 알고리즘은 표 1에 제시하였다.



(1st Stage)

(2nd Stage)

(a) Borůvka 알고리즘



수행 순서	간선 내림차순 정렬	그래프 정점 분리	정점 분리 여부 확인 (사이클)	남은 간선 수
1	{E,J}=18	X	{E,J}, {J,F}, {F,E}	20
2	{F,J}=10	X	{F,J}, {J,I}, {I,F}	19
3	{F,I}=9	X	{F,I}, {I,G}, {G,F}	18
4	{D,F}=9	X	{D,F}, {F,C}, {C,D}	17
5	{D,E}=9	X	{D,E}, {E,F}, {F,C}, {C,D}	16
6	{C,G}=9	X	{C,G}, {G,F}, {F,C}	15
7	{B,G}=9	X	{B,G}, {G,F}, {F,C}, {C,B}	14
8	{A,E}=9	X	{A,E}, {E,F}, {F,C}, {C,D}, {D,A}	13
9	**{E,F}=8**	O	-	13
10	**{C,F}=8**	O	-	13
11	**{F,G}=7**	O	-	13
12	{A,B}=6	X	{A,B}, {B,D}, {D,A}	12
13	{G,I}=5	X	{G,I}, {I,H}, {H,G}	11
14	{H,J}=4	X	{H,J}. {J,I}, {I,H}	10
15	**{G,H}=4**	O	-	10
16	{B,D}=4	X	{B,D}, {D,C}, {C,B}	9
17	**{I,J}=3**	O	-	9
18	**{A,D}=3**	O	-	9
19	**{C,D}=2**	O	-	9
20	**{B,C}=2**	O	-	9
21	**{H,I}=1**	O	-	9

(b) 역-삭제 알고리즘

그림 2. G_1 **그래프의 Borůvka와 역-삭제 알고리즘 적용**
Fig. 2. Application of Borůvka and reverse-delete algorithm of Graph G_1

표 1. G_1 **그래프의 Prim과 Kruskal 알고리즘 적용**
Table 1. Application of Prim and Kruskal algorithm of Graph G_1

(a) Prim 알고리즘의 계산 방법

V	T	S				e_{mst}
		Remained (S_R)	Add (S_A)	Delete (S_D)	$S_R+S_A-S_D$	
{A,B,C,D,E,F,G,H,I,J}	{ ⌀ }	{ ⌀ }	{ ⌀ }	{ ⌀ }	{ ⌀ }	{ ⌀ }
{B,C,D,E,F,G,H,I,J}	{**A**}	{ ⌀ }	{A,B}=6 {A,D}=3 {A,E}=9	{ ⌀ }	{A,B}=6, **{A,D}=3** {A,E}=9	{A,**D**}=3
{B,C,E,F,G,H,I,J}	{A,**D**}	{A,B}=6, {A,E}=9	{D,A}=3 {D,B}=4 {D,C}=2 {D,E}=9 {D,F}=9	{D,A}=3	{A,B}=6, {A,E}=9 {D,B}=4, **{D,C}=2** {D,E}=9, {D,F}=9	{D,**C**}=2
{B,E,F,G,H,I,J}	{A,D,**C**}	{A,B}=6, {A,E}=9 {D,B}=4, {D,E}=9 {D,F}=9	{C,B}=2 {C,D}=2 {C,F}=8 {C,G}=9	{C,D}=2	{A,B}=6, {A,E}=9 {D,B}=4, {D,E}=9 {D,F}=9, **{C,B}=2** {C,F}=8, {C,G}=9	{C,**B**}=2
{E,F,G,H,I,J}	{A,D,C,**B**}	{A,B}=6, {A,E}=9 {D,B}=4, {D,E}=9 {D,F}=9, {C,F}=8 {C,G}=9	{B,A}=6 {B,C}=2 {B,D}=4 {B,G}=9	{A,B}=6 {D,B}=4 {B,A}=6 {B,C}=2 {B,D}=4	{A,E}=9, {D,E}=9 {D,F}=9, **{C,F}=8** {C,G}=9, {B,G}=9	{C,**F**}=8
{E,G,H,I,J}	{A,D,C,B,**F**}	{A,E}=9, {D,E}=9 {D,F}=9, {C,G}=9 {B,G}=9	{F,C}=8 {F,D}=9 {F,E}=8 {F,G}=7 {F,I}=9 {F,J}=10	{D,F}=9 {F,C}=8 {F,D}=9	{A,E}=9, {D,E}=9 {C,G}=9, {B,G}=9 {F,E}=8, **{F,G}=7** {F,I}=9, {F,J}=10	{F,**G**}=7
{E,H,I,J}	{A,D,C,B,F,**G**}	{A,E}=9, {D,E}=9 {C,G}=9, {B,G}=9 {F,E}=8, {F,I}=9 {F,J}=10	{G,B}=9 {G,C}=9 {G,F}=7 {G,H}=4 {G,I}=5	{C,G}=9 {G,B}=9 {G,C}=9 {G,F}=7	{A,E}=9, {D,E}=9 {B,G}=9, {F,E}=8 {F,I}=9, {F,J}=10 **{G,H}=4**, {G,I}=5	{G,**H**}=4
{E,I,J}	{A,D,C,B,F,G,**H**}	{A,E}=9, {D,E}=9 {B,G}=9, {F,E}=8 {F,I}=9, {F,J}=10 {G,I}=5	{H,G}=4 {H,I}=1 {H,J}=4	{B,G}=9 {H,G}=4	{A,E}=9, {D,E}=9 {F,E}=8, {F,I}=9 {F,J}=10, {G,I}=5 **{H,I}=1**, {H,J}=4	{H,**I**}=1
{E,J}	{A,D,C,B,F,G,H,**I**}	{A,E}=9, {D,E}=9 {F,E}=8, {F,J}=9 {F,J}=10, {G,I}=5 {H,J}=4	{I,F}=9 {I,G}=5 {I,H}=1 {I,J}=3	{F,I}=9 {G,I}=5 {I,F}=9 {I,G}=5 {I,H}=1	{A,E}=9, {D,E}=9 {F,E}=8, {F,J}=10 {H,J}=4, **{I,J}=3**	{I,**J**}=3
{E}	{A,D,C,B,F,G,H,I,**J**}	{A,E}=9, {D,E}=9 {F,E}=8, {F,J}=10 {H,J}=4	{J,E}=18 {J,F}=10 {J,H}=4 {J,I}=3	{F,J}=10 {H,J}=4 {J,F}=10 {J,H}=4 {J,I}=3	{A,E}=9, {D,E}=9 **{F,E}=8**, {J,E}=18	{F,**E**}=8
{ ⌀ }	{A,D,C,B,F,G,H,I,J,**E**}	-	-	-	-	-

(b) Kruskal 알고리즘의 계산 방법

S	V	T	e_{mst}
{H, I}=1	{A,B,C,D,E,F,G,**H**,**I**,J}	{ ⌀ }	{H, I}=1
{B,C}=2	{A,**B**,**C**,D,E,F,G,J}	{H,I}	{B,C}=2
{C,D}=2	{A,**D**,E,F,G,J}	{H,I}, {B,**C**}	{C,D}=2
{A,D}=3	{**A**,E,F,G,J}	{H,I}, {B,C,**D**}	{A,D}=3
{I, J}=3	{E,F,G,**J**}	{H,**I**}, {A,B,C,D}	{I, J}=3
{B,D}=4	{E,F,G}	{H,I,J}, {A,**B**,C,**D**}	-
{G,H}=4	{E,F,**G**}	{**H**,I,J}, {A,B,C,D}	{G,H}=4
{H, J}=4	{E,F}	{G,**H**,I,**J**}, {A,B,C,D}	-
{G, I}=5	{E,F}	{**G**,H,I,J}, {A,B,C,D}	-
{A,B}=6	{E,F}	{G,H,I,J}, {**A**,B,C,D}	-
{F,G}=7	{E,**F**}	{**G**,H,I,J}, {A,B,C,D}	{F,G}=7
{C,F}=8	{E}	{**F**,G,H,I,J}, {A,B,**C**,D}	{C,F}=8
{E,F}=8	{**E**}	{A,B,C,D,**F**,G,H,I,J}	{E,F}=8
{A,E}=9	{ ⌀ }	{**A**,B,C,D,**E**,F,G,H,I,J}	-
{B,G}=9	{ ⌀ }	{A,**B**,C,D,E,F,**G**,H,I,J}	-
{C,G}=9	{ ⌀ }	{A,B,**C**,D,E,F,**G**,H,I,J}	-
{D,E}=9	{ ⌀ }	{A,B,C,**D**,**E**,F,G,H,I,J}	-
{D,F}=9	{ ⌀ }	{A,B,C,**D**,E,**F**,G,H,I,J}	-
{F, I}=9	{ ⌀ }	{A,B,C,D,E,**F**,G,H,**I**,J}	-
{F,J}=10	{ ⌀ }	{A,B,C,D,E,**F**,G,H,I,**J**}	-
{E,J}=18	{ ⌀ }	{A,B,C,D,**E**,F,G,H,I,**J**}	-

3. MST 알고리즘 분석과 연구 배경

그래프의 간선 $|e|$개에서 MST에 기여하는 간선들인 $|v|-1$를 선택하기 위해 기존의 MST 알고리즘은 모두 모집단을 $|e|$개 사용한다는 공통점이 있다. 이로 인해 Borůvka 알고리즘을 계산 방법으로 구현하기 위해서는 1st Stage에서 얻은 간선들의 사이클 확인과 2nd Stage에서 부분신장트리들 간을 연결하는 최소 가중치 간선을 찾기 위해 결국 Kruskal 알고리즘을 적용해야 한다. 따라서 Borůvka 알고리즘은 계산모델로 알고리즘을 구현하는 것보다 그래프를 시각적으로 보는 그래픽 방법으로 최소 가중치 간선을 선택하는 것이 보다 쉽다.

역-삭제 알고리즘은 임의의 간선을 삭제시 정점을 그래프에서 분리시키는지 확인하기 위해서는 그래프의 복잡한 사이클 발생 여부를 모두 탐색해야 하는 구현이 어렵다. 즉, Borůvka와 역-삭제 알고리즘은 그래프를 시각적으로 확인하는 방법이 계산 방법으로 구하는 방법보다 이해가 쉽다.

Prim 알고리즘은 하나의 정점을 선택하는데 있어 추가, 삭제와 선택 과정을 거치며, 추가에는 $2|e|$개의 간선을, 삭제에는 약 $1.5|e|$개의 간선을 대상으로 하기 때문에 알고리즘 수행 시간이 많이 소요된다. Kruskal 알고리즘은 항상 $|e|$개의 간선을 대상으로 한다. 결국, MST 알고리즘의 수행 횟수를 단축시키는 최선책은 간선 모집단의 개수를 감소시키는 방법이며, 차선책은 사이클이 발생하는 간선을 제거하면서 $|v|-1$개를 찾았을 때 알고리즘의 종료시키는 방법이 될 수 있다. 3장에서는 이 두 가지 해결 방법을 적용하여 MST를 쉽게 구하는 기법을 제안한다.

Ⅲ. 모집단 축소 기법 적용 빠른 MST 결정

본 장에서는 MST 알고리즘의 수행 횟수를 단축시키는 방법에 기반하여 사이클이 발생하는 간선들을 제거하고 알고리즘을 빨리 종료시키는 기준을 적용한 방법을 제안한다.

1. 모집단 축소 기법

그래프의 간선 모집단의 개수를 축소시키는 방법을 고찰해 보자. MST는 절단 속성 (Cut Property)과 사이클 속성 (Cycle Property)을 갖고 있다. 절단 속성은 부분신장트리 T에 최소 가중치를 갖는 간선 e의 어느 한 정점이 속해 있으면 e는 MST에 포함된다는 속성이며, 사이클 속성은 임의의 사이클 C에 존재하는 최대 가중치 간선 e는 MST에 포함되지 않는다는 속성이다.

Borůvka, Prim과 Kruskal 알고리즘에서 사이클이 발

생하지 않는 최소 가중치 간선을 선택하는 방법은 절단 속성을 이용하는 방법이며, 역-삭제 알고리즘에서 최대 가중치 간선이 정점을 그래프에서 분리시키지 않으면 선택하는 기법은 절단과 사이클 속성 모두를 적용하는 방법이다.

결국, 절단 속성이나 사이클 속성을 적용하여 얻은 MST는 그래프의 사이클에서 최대 가중치 간선은 MST로 선택되지 않는다. 이러한 최대 가중치 간선들을 사전에 가능한 많이 식별하기 위해 본 논문에서는 그래프의 결합가 (Valency, δ) 개념을 도입한다. 결합가는 어느 한 정점 i에 연결된 간선의 수로 계산된다. 예로, 그림 1의 G_1 그래프에서 정점 A는 $\delta(3)$, 정점 D는 $\delta(5)$, 정점 C는 $\delta(4)$이다. MST에 속하지 않는 임의의 정점에서의 최대 가중치 간선들을 가능한 많이 사전에 제거하는 다음의 방법을 간선 모집단 (Population)의 대상을 축소하는 전처리 과정 (Preprocessing)이라 하자.

(1) 가중치 간선들 정방행렬에서 결합가 $\delta(i) \geq 3$인 행 정점 i에 대해 최대 가중치 간선 $e_{\max}\ \{i,j\}$를 선택한다. (단, 동일한 가중치는 모두 선택한다.)

	A	B	C	D	E	F	G	H	I	J	$\delta(i)$
A		6		3	9						3
B	6		2	4			9				4
C		2		2		8	9				4
D	3	4	2		9	9					5
E	9			9		8				18	3
F			8	9	8		7		9	10	6
G		9	9			7		4	5		5
H							4		1	4	3
I						9	5	1		3	4
J					18	10		4	3		4

(2) $\delta(i) - |e_{\max}| \geq 2$이면 $e_{\max}$인 최대 가중치 간선들 $\{i,j\}$를 삭제한다.

	A	B	C	D	E	F	G	H	I	J	$\delta(i)$	$\delta(i) - \lvert e_{\max} \rvert$
A		6		3	9						3	3-1 = 2
B	6		2	4			9				4	4-1 = 3
C		2		2		8	9				4	4-1 = 3
D	3	4	2		9	9					5	5-2 = 3
E	9			9		8				18	3	3-1 = 2
F			8	9	8		7		9	10	6	6-1 = 5
G		9	9			7		4	5		5	5-2 = 3
H							4		1	4	3	3-2 = 1
I						9	5	1		3	4	4-1 = 3
J					18	10		4	3		4	4-1 = 3

(3) $\{i,j\} = \{j,i\}$이므로 $\{j,i\}$를 삭제한다. 만약, 이 과정에서 특정 정점 i의 $\delta(i) = 0$이 되면 정점 i의 최소 가중치 간선은 삭제하지 않는다.

	A	B	C	D	E	F	G	H	I	J
A		6		3						
B	6		2	4						
C		2		2		8				
D	3	4	2							
E						8				
F			8		8		7			
G						7		4	5	
H							4		1	4
I							5	1		3
J								4	3	

이와 같이 전처리 과정을 거치면 모집단인 간선의 수는 기존의 21개에서 13개로 축소시킬 수 있다. 이들 13개의 간선을 모집단으로 하여 MST 알고리즘들을 적용하여 $|v|-1=9$개의 간선을 선택하거나 4개의 간선을 삭제하였을 때 알고리즘을 종료시키는 방법이 최적의 종료시점이 된다.

2. 축소된 모집단 대상 MST 결정

본 절에서 적용되는 변수들 중 V는 그래프의 모든 정점들, S는 간선들, T는 사이클 발생 여부 확인을 위해 MST로 되는 정점들, e_{mst}는 MST로 결정되는 간선들, $|e_{mst}|$는 MST로 결정되는 간선의 수를 의미한다.

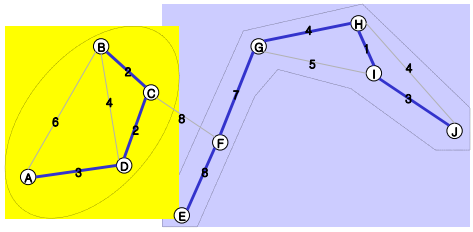
가. 그래픽 방법 MST 알고리즘

Borůvka 알고리즘을 그래픽 모델로 구현하면 다음과 같다.

(1) 전처리로 얻어진 모집단 간선들을 대상으로 정방행렬의 각 행 정점에 대해 최소 가중치 간선 $e_{\min}$을 선택한다.

	A	B	C	D	E	F	G	H	I	J
A		6		3						
B	6		2	4						
C		2		2		8				
D	3	4	2							
E						8				
F			8		8		7			
G						7		4	5	
H							4		1	4
I							5	1		3
J								4	3	

(2) 선택된 간선들을 연결하여 부분신장트리를 구성한다.



(3) 두 부분 신장트리를 연결하는 최소 가중치 간선을 선택한다. $\{C,F\} = 8$

참고로 Borůvka 알고리즘을 계산 모델로 구현하면 다음과 같다.

(1) 모든 변수에 초기값을 설정한다.
$S = \{\varnothing\}$, $e_{mst} = \{\varnothing\}$, $T = \{\varnothing\}$, $|e_{mst}| = 0$

(2) 정방행렬의 상삼각행렬에 있는 모든 간선들을 S에 저장하고 오름차순으로 정렬한다.

(3) 정방행렬의 각 행 정점에 대해 최소 가중치 간선을 선택하여 $\{i,j\}$에 대해 $\{j,i\}$를 선택한다. 상삼각행렬에서 선택된 간선들에 대해 e_{mst}에 저장, $|e_{mst}| = |e_{mst}| + 1$로 하고, S에서 삭

제한다.

(4) e_{mst}에 저장된 간선들을 T의 부분집합으로 구성한다.
만약, T의 부분집합의 개수가 1개이면서 $|e_{mst}|=|v|-1$이면 알고리즘을 종료한다. 만약 부분집합의 개수가 2개 이상이면 (5)를 수행한다.

(5) S에 있는 간선들을 최소 가중치 간선부터 선택한다.
(가) 만약, 간선 i와 j가 T의 한 부분집합에 모두 존재하면 생략한다.
(나) 만약, 간선 i와 j가 T의 두 부분집합에 분리되어 존재하면 두 집합을 하나로 연결하고, $\{i,j\}$를 e_{mst}에 저장, $|e_{mst}|=|e_{mst}|+1$로 설정한다.

(6) T의 부분집합의 개수가 1개이면서 $|e_{mst}|=|v|-1$이 될 때까지 (5)를 반복 수행한다.

여기서, (3)과 (4)는 1st Stage, 와 (5)는 2nd Stage이며, 이 단계에서 사이클이 발생하는 간선은 Kruskal 알고리즘 방법으로 확인할 수 있다. 결국, Borůvka와 Kruskal 알고리즘은 간선을 선택하는 방법에 차이가 있을 뿐, 사이클 여부를 확인하는 알고리즘 수행 과정은 동일함을 알 수 있다.

전처리 과정을 거친 간선 모집단에 대해 계산 모델 Borůvka 알고리즘을 적용한 결과는 표 2와 같다.

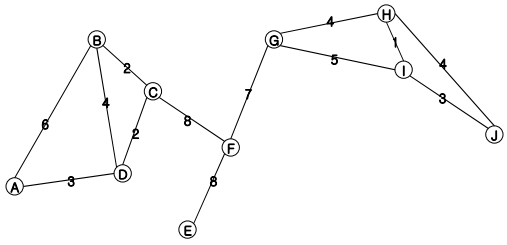
표 2. G_1 그래프의 축소된 모집단 대상 Borůvka 알고리즘 적용

Table 2. Application of Borůvka algorithm with the reduced population of Graph G_1

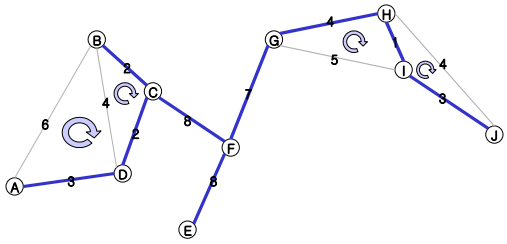
S	최소 가중치 간선	사이클 Check	e_{mst}	$\|e_{mst}\|$
{H, I}=1	**{H, I}=1**	{H,I}	**{H, I}=1**	1
{B,C}=2	**{B,C}=2**	{H,I}, {B,C}	**{B,C}=2**	2
{C,D}=2	**{C,D}=2**	{H,I}, {B,C,D}	**{C,D}=2**	3
{A,D}=3	**{A,D}=3**	{H,I}, {A,B,C,D}	**{A,D}=3**	4
{I, J}=3	**{I, J}=3**	{H,I,J}, {A,B,C,D}	**{I, J}=3**	5
{B,D}=4	-	-		
{G,H}=4	**{G,H}=4**	{G,H,I,J}, {A,B,C,D}	**{G,H}=4**	6
{H, J}=4	-	-		
{G, I}=5	-	-		
{A,B}=6	-	-		
{F,G}=7	**{F,G}=7**	{F,G,H,I,J}, {A,B,C,D}	**{F,G}=7**	7
{C,F}=8	-	-		
{E,F}=8	**{E,F}=8**	{E,F,G,H,I,J}, {A,B,C,D}	**{E,F}=8**	8
{B,D}=4	-	-	-	8
{H, J}=4	-	-	-	8
{G, I}=5	-	-	-	8
{A,B}=6	-	-	-	8
{C,F}=8	**{C,F}=8**	{A,B,C,D,E,F,G,H,I,J}	{C,F}=8	9

역-삭제 알고리즘을 그래픽 방법으로 구현하면 다음과 같다. 원래의 역-삭제 알고리즘이 간선을 내림차순으로 정렬시켜 최대 가중치 간선부터 선택 여부를 결정하는 방법을 취하고 있으나 본 논문에서는 랜덤하게 사이클을 확인하는 방법을 택한다.

(1) 전처리로 얻어진 모집단 간선들을 대상으로 상삼각행렬 간선들만으로 그래프를 구성한다.



(2) 그래프에서 사이클이 발생하는 부분을 랜덤하게 찾아 해당 사이클에서 최대 가중치 간선을 삭제한다.



참고로, 역-삭제 알고리즘을 계산모델로 구현하여 MST를 찾는 방법은 임의의 간선 $\{i,j\}$을 제거시 두 정점 i,j중에서 어느 한 정점이 그래프에서 분리되는지 여부는 사이클 발생 여부로 결정할 수 있다. 전처리 과정을 거친 간선 모집단에 대해 계산 모델 역-삭제 알고리즘을 적용한 결과는 표 3과 같다.

표 3. G_1 그래프의 축소된 모집단 대상 역-삭제 알고리즘 적용

Table 3. Application of reverse-delete algorithm with the reduced population of Graph G_1

S	Cycle	S	$\|S\|$
{E,F}=8	-	{E,F}=8	13
{C,F}=8	-	{C,F}=8	13
{F,G}=7	-	{F,G}=7	13
{A,B}=6	{**A**,B}=6, {B,D}=4, {D,**A**}=3	-	12
{G, I}=5	{**G**,I}=5, {I,H}=1, {H,**G**}=4	-	11
{H, J}=4	{**H**,J}=4, {J,I}=3, {I,**H**}=1	-	10
{G,H}=4	-	{G,H}=4	10
{B,D}=4	{**B**,D}=4, {D,C}=2, {C,**B**}=2	-	**9**
{I, J}=3	(알고리즘 종료)	{I, J}=3	
{A,D}=3		{A,D}=3	
{C,D}=2		{C,D}=2	
{B,C}=2		{B,C}=2	
{H, I}=1		{H, I}=1	

나. 계산 방법 MST 알고리즘

Prim 알고리즘은 다음과 같이 구현할 수 있다.

(1) 모든 변수에 초기값을 설정한다.
V에 모든 정점들 저장, $T=\{\varnothing\}$, $S=\{\varnothing\}$, $e_{mst}=\{\varnothing\}$.

(2) 정방행렬에서 임의의 정점 i를 선택하여 T에 저장하고, V에서 삭제한다.
(가) 정점 i에 연결된 간선들을 S에 저장한다.
(나) S에 저장된 간선들에 대해 $\{i,j\}$가 모두 T에 존재하

면 사이클이 발생하거나 기존에 이미 최소 가중치 간선이 연결되어 있으므로 삭제한다.

(다) S에 저장된 간선들 중 최소 가중치 간선 $\{i,j\}$를 선택하여 e_{mst}에 저장하고, 정점 i,j 중 V에 존재하는 정점에 대해 V에서 삭제하고 T에 저장한다.

(3) $V=\{\varnothing\}$가 될 때까지 T에 새로 저장된 정점 j에 대해 (2)를 반복적으로 수행한다.

전처리 과정을 거친 간선 모집단에 대해 Prim 알고리즘을 적용한 결과는 표 4와 같다.

표 4. G_1 그래프의 축소된 모집단 대상 Prim 알고리즘 적용
Table 4. Application of Prim algorithm with the reduced population of Graph G_1

V	T	S				e_{mst}
		Remained (S_R)	Add (S_A)	Delete (S_D)	S_R+S_A-	
{A,B,C,D,E,F,G,H,I,J}	{ ∅ }	{ ∅ }	{ ∅ }	{ ∅ }	{ ∅ }	{ ∅ }
{B,C,D,E,F,G,H,I, J}	{**A**}	{ ∅ }	{A,B}=6 {A,D}=3	{ ∅ }	{A,B}=6 **{A,D}=3**	{A,**D**}=3
{B,C,E,F,G,H,I,J}	{A,**D**}	{A,B}=6	{D,A}=3 {D,B}=4 {D,C}=2	{D,A}=3	{A,B}=6 {D,B}=4 **{D,C}=2**	{D,**C**}=2
{B,E,F,G,H,I,J}	{A,D,**C**}	{A,B}=6 {D,B}=4	{C,B}=2 {C,D}=2 {C,F}=8	{C,D}=2	{A,B}=6 {D,B}=4 **{C,B}=2** {C,F}=8	{C,**B**}=2
{E,F,G,H,I,J}	{A,D,C,**B**}	{A,B}=6 {D,B}=4 {C,F}=8	{B,A}=6 {B,C}=2 {B,D}=4	{A,B}=6 {D,B}=4 {B,A}=6 {B,C}=2 {B,D}=4	{C,F}=8	{C,**F**}=8
{E,G,H,I,J}	{A,D,C,B,**F**}	{ ∅ }	{F,C}=8 {F,E}=8 {F,G}=7	{F,C}=8	{F,E}=8 **{F,G}=7**	{F,**G**}=7
{E,H,I,J}	{A,D,C,B,F,**G**}	{F,E}=8	{G,F}=7 {G,H}=4 {G,I}=5	{G,F}=7	{F,E}=8 **{G,H}=4** {G,I}=5	{G,**H**}=4
{E,I,J}	{A,D,C,B,F,G,**H**}	{F,E}=8 {G,I}=5	{H,G}=4 {H,I}=1 {H,J}=4	{H,G}=4	{F,E}=8 {G,I}=5 **{H,I}=1** {H,J}=4	{H,**I**}=1
{E,J}	{A,D,C,B,F,G,H,**I**}	{F,E}=8 {G,I}=5 {H,J}=4	{I,G}=5 {I,H}=1 {I,J}=3	{G,I}=5 {I,G}=5 {I,H}=1	{F,E}=8 {H,J}=4 **{I,J}=3**	{I,**J**}=3
{E}	{A,D,C,B,F,G,H,I,**J**}	{F,E}=8 {H,J}=4	{J,H}=4 {J,I}=3	{H,J}=4 {J,H}=4 {J,I}=3	**{F,E}=8**	{F,**E**}=8
{ ∅ }	{A,D,C,B,F,G,H,I,J,**E**}	-	-	-	-	-

Kruskal 알고리즘은 다음과 같이 구현할 수 있다.

(1) 모든 변수에 초기값을 설정한다.
V에 모든 정점들 저장, $T=\{\varnothing\}$, $e_{mst}=\{\varnothing\}$, $|e_{mst}|=0$.

(2) 정방행렬의 상삼각행렬에 있는 모든 간선들을 S에 저장하고, 오름차순으로 정렬시킨다.

(3) S에서 최소 가중치를 갖는 간선 $\{i,j\}$을 선택하여 사이클 발생 여부를 확인한다.

(가) 만약, i,j 정점 모두 V에 존재하면 i,j를 T의 한 집합으로 생성, $\{i,j\}$를 e_{mst}에 저장, $|e_{mst}|=|e_{mst}|+1$, V에서 삭제한다.

(나) 만약, i,j 중 한 정점 (i)은 V에, 다른 한 정점 (j)은 T의 한 집합에 존재하면 V에 속한 정점 (i)을 T의 한 집합에 저장, $\{i,j\}$를 e_{mst}에 저장, $|e_{mst}|=|e_{mst}|+1$, 정점 i를 V에서 삭제한다.

(다) 만약, i,j 모두 V에는 없는 경우,
i,j가 T의 어느 한 집합에 모두 존재하면 사이클이 발생하는 경우이므로 선택하지 않는다.
i,j가 T의 서로 다른 집합에 존재하면 $\{i,j\}$를 e_{mst}에 저장, $|e_{mst}|=|e_{mst}|+1$, T의 두 집합을 하나의 집합으로 연결한다.

(4) S의 모든 간선에 대해 (3)을 수행하고 알고리즘을 종료한다.
그 결과 e_{mst}가 MST로 얻어지며, $|e_{mst}|=|v|-1$이 된다.

전처리 과정을 거친 간선 모집단에 대해 Kruskal 알고리즘을 적용한 결과는 표 5와 같다.

표 5. G_1 그래프의 축소된 모집단 대상 Kruskal 알고리즘 적용
Table 5. Application of Kruskal algorithm with the reduced population of Graph G_1

S	V	T	e_{mst}	$\lvert e_{mst}\rvert$
{H, I}=1	{A,B,C,D,E,F,G,**H,I,J**}	{ ∅ }	{H, I}=1	1
{B,C}=2	{A,**B,C**,D,E,F,G,J}	{H,I}	{B,C}=2	2
{C,D}=2	{A,**D**,E,F,G,J}	{H,I}, {B,**C**}	{C,D}=2	3
{A,D}=3	{**A**,E,F,G,J}	{H,I}, {B,C,**D**}	{A,D}=3	4
{I, J}=3	{E,F,G,**J**}	{**H,I**}, {A,B,C,D}	{I, J}=3	5
{B,D}=4	{E,F,G}	{H,I,J}, {A,**B**,C,**D**}	-	5
{G,H}=4	{E,F,**G**}	{**H,I**,J}, {A,B,C,D}	{G,H}=4	6
{H, J}=4	{E,F}	{G,**H,I,J**}, {A,B,C,D}	-	6
{G, I}=5	{E,F}	{**G**,H,I,J}, {A,B,C,D}	-	6
{A,B}=6	{E,F}	{G,H,I,J}, {**A,B**,C,D}	-	6
{F,G}=7	{E,**F**}	{**G**,H,I,J}, {A,B,C,D}	{F,G}=7	7
{C,F}=8	{E}	{**F**,G,H,I,J}, {A,B,**C**,D}	{C,F}=8	8
{E,F}=8	{**E**}	{A,B,C,D,**F**,G,H,I,J}	{E,F}=8	**9**
	{ ∅ }	(알고리즘 종료)		

3. 알고리즘 성능 분석

본 절에서는 전처리 과정을 거치지 않은 전체 간선을 대상으로 하는 방법과 전처리 과정을 거쳐 모집단을 축소한 상태에서 MST를 얻는 방법의 성능을 비교하여 표 6에 제시하였다.

표 6. G_1 그래프의 모집단 크기에 따른 알고리즘 수행 횟수 비교
Table 6. Running time comparison of the algorithms with population size of Graph G_1

알고리즘	전체 모집단 이용	축소된 모집단 이용
Borůvka 알고리즘	1st Stage : 21개중 8개 선택 2nd Stage : 6개중 1개 선택	1st Stage : 13개중 8개 선택 2nd Stage : 1개중 1개 선택
역-삭제 알고리즘	21회 수행, 사이클 발생 간선 12개 삭제	4회 수행, 사이클 발생 간선 4개 삭제
Prim 알고리즘	간선 추가 38개, 간선 삭제 26개, 57개 간선 중 9개 선택 (121개)	간선 추가 26개, 간선 삭제 16개, 23개 간선 중 9개 선택 (65개)
Kruskal 알고리즘	간선 21개 수행,사이클 12개 발생, 9개 간선 선택	간선 13개 수행, 사이클 4개 발생, 9개 간선 선택

Borůvka 알고리즘은 27개 간선을 14개 간선으로 줄여 48%를, Prim 알고리즘은 46%, Kruskal 알고리즘은 34%, 역-삭제 알고리즘은 81%의 수행횟수 단축 효과를 얻었다.

Ⅳ. 알고리즘 적용성 평가

본 장에서는 그림 3의 다양한 8개 그래프에 대해 전체 모집단을 사용하는 경우와 모집단을 축소시켜 사용하는 경우 MST 알고리즘의 간선 비교 횟수 단축 효과를 고찰해 본다. 각 경우에 대해 MST를 구하는 과정은 G_1 그래프와 동일하므로 생략한다.

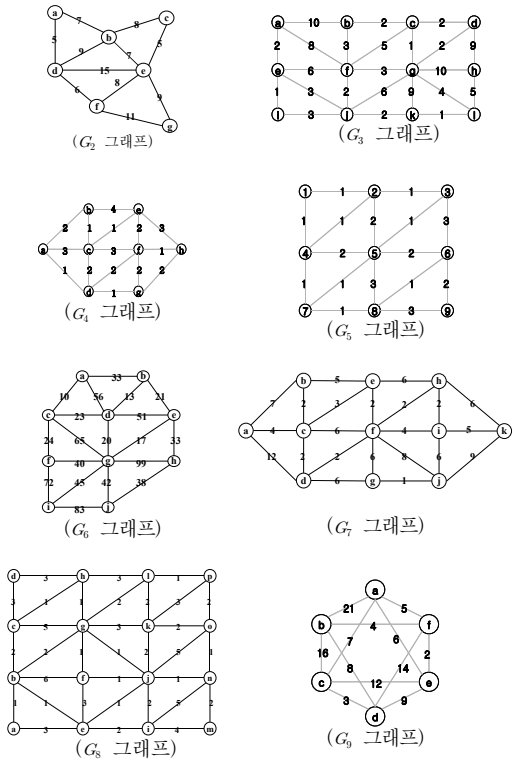


그림 3. 알고리즘 평가에 사용된 그래프
Fig. 3. Graph used for algorithm evaluation

G_1 그래프를 포함하여 그림 3의 8개 그래프에 대해 알고리즘 간선 비교 횟수를 비교한 결과는 표 7에 제시하였다.

전처리 결과 간선 모집단은 평균 35% 축소하는 결과를 얻었다. 축소된 모집단을 대상으로 알고리즘을 수행한 결과 간선 비교 횟수를 Borůvka 알고리즘은 38%, Prim 알고리즘은 37%, Kruskal 알고리즘은 39%, 역-삭제 알고리즘은 81%와 73%를 단축시키는 효과를 얻었다.

표 7. MST 알고리즘의 간선 비교 횟수 비교
Table 7. Comparison of edge comparison number of MST algorithm

(a) Borůvka 알고리즘

그래프	$\|v\|$	$\|e\|$	비교 간선 수/선택 간선 수/삭제 간선 수				축소 모집단의 간선 비교횟수 단축율(%)
			전체 모집단		축소 모집단		
			1st Stage	2nd Stage	1st Stage	2nd Stage	
G_1	10	**21**	21/ 8/0	6/ 1/0	**13/ 8/0**	**1/ 1/0**	**48%**
G_2	7	**11**	11/10/0	-	**8/ 6/0**	**-**	**27%**
G_3	12	**23**	23/10/1	13/ 2/0	**14/ 9/1**	**5/ 2/0**	**47%**
G_4	8	**15**	15/ 5/0	12/ 2/0	**9/ 5/0**	**4/ 2/0**	**52%**
G_5	9	**16**	16/10/2	-	**11/10/2**	**-**	**31%**
G_6	10	**19**	19/7/0	9/ 2/0	**12/ 7/0**	**4/ 2/0**	**43%**
G_7	11	**22**	22/ 9/0	5/ 1/0	**14/ 9/0**	**2/ 1/0**	**44%**
G_8	16	**33**	33/19/4	-	**23/14/1**	**6/ 2/0**	**12%**
G_9	6	**12**	12/ 4/0	6/ 1/0	**8/ 4/0**	**3/ 1/0**	**39%**
평균							**38%**

(b) Prim 알고리즘

그래프	$\|v\|$	$\|e\|$	추가 간선 수/삭제 간선 수/비교 간선 수		축소 모집단의 간선 비교 횟수 단축율 (%)
			전체 모집단	축소 모집단	
G_1	10	**21**	38/26/ 57 (121)	**25/16/23 (65)**	**46%**
G_2	7	**11**	20/13/ 22 (45)	**14/ 8/13 (35)**	**22%**
G_3	12	**23**	43/30/ 61 (134)	**29/17/30 (76)**	**43%**
G_4	8	**15**	27/18/ 36 (81)	**16/ 8/18 (42)**	**48%**
G_5	9	**16**	29/20/ 27 (76)	**21/13/17 (51)**	**33%**
G_6	10	**19**	35/24/ 46 (105)	**23/14/27 (64)**	**39%**
G_7	11	**22**	40/27/ 54 (121)	**27/17/32 (76)**	**37%**
G_8	16	**33**	64/45/109 (218)	**45/30/84 (159)**	**27%**
G_9	6	**12**	20/12/ 26 (58)	**13/ 6/16 (35)**	**40%**
평균					**37%**

(c) Kruskal 알고리즘

그래프	$\|v\|$	$\|e\|$	대상 간선 수/수행 간선 수		축소 모집단의 간선 비교 횟수 단축율 (%)
			전체 모집단	축소 모집단	
G_1	10	**21**	21/21	**13/13**	**38%**
G_2	7	**11**	11/11	**8/ 8**	**27%**
G_3	12	**23**	23/23	**14/14**	**39%**
G_4	8	**15**	15/15	**9/ 7**	**53%**
G_5	9	**16**	16/16	**11/11**	**31%**
G_6	10	**19**	19/19	**12/12**	**37%**
G_7	11	**22**	22/22	**14/13**	**41%**
G_8	16	**33**	33/33	**23/22**	**33%**
G_9	6	**12**	12/12	**8/ 6**	**50%**
평균					**39%**

(d) 역-삭제 알고리즘

그래프	$\|v\|$	$\|e\|$	대상 간선 수/사이클 수/수행 간선 수			랜덤 선택 기준 간선 비교 횟수 단축율 (%)	
			전체 모집단		축소 모집단		
			$\|e\|$개 수행	$\|v\|-1$에서 종료	랜덤 선택	$\|e\|$개 수행	$\|v\|-1$에서 종료
G_1	10	**21**	21/ 8/21	21/ 8/16	**13/4/4**	**81%**	**75%**
G_2	7	**11**	11/ 3/11	11/ 3/ 6	**8/2/2**	**82%**	**67%**
G_3	12	**23**	23/ 9/23	23/ 9/16	**14/4/4**	**83%**	**75%**
G_4	8	**15**	15/ 6/15	15/ 6/ 8	**9/2/2**	**87%**	**75%**
G_5	9	**16**	16/ 5/16	16/ 5/13	**11/3/3**	**81%**	**77%**
G_6	10	**19**	19/ 8/19	19/ 8/15	**12/3/3**	**84%**	**80%**
G_7	11	**22**	22/ 8/22	22/ 8/15	**14/4/4**	**82%**	**73%**
G_8	16	**33**	33/10/33	33/10/27	**23/8/8**	**76%**	**70%**
G_9	6	**12**	12/ 4/12	12/ 4/ 8	**8/3/3**	**75%**	**63%**
평균						**81%**	**73%**

V. 결 론

본 논문은 최소신장트리를 구하는 대표적인 Borůvka, Prim, Kruskal과 역-삭제 알고리즘의 간선 비교 횟수를 획기적으로 감소시켜 MST를 빠르게 결정하는 방법을 제안하였다. 기존의 알고리즘들은 모두 그래프의 전체 간선을 모집단으로 하여 MST를 얻는 공통점을 갖고 있다. 또한, 모든 간선들을 비교한 시점에서 알고리즘이 종료된다. 반면에 제안된 방법은 그래프의 간선들 중에서 MST에 기여하지 못하는 가중치가 큰 간선들을 사전에 충분히 제거하는 모집단 축소 전처리 과정을 거쳤다. 전처리 과정에는 그래프의 결합가를 활용하는 개념을 적용하였다. 전처리 결과 간선 모집단의 수를 약 38% 축소시키는 효과를 얻었다. 또한, 최적의 알고리즘 종료시점 기준을 적용하였다. Borůvka와 Kruskal 알고리즘은 추가되는 간선 수가 $|v|-1$개 일 때 알고리즘을 종료시켰으며, 역-삭제 알고리즘은 남아 있는 간선 수가 $|v|-1$일 때 종료시켰다. 또한, 역-삭제 알고리즘에서 사이클을 랜덤하게 선택하여 최대 가중치 간선을 제거하는 기준을 적용하였다. 이 결과 간선 비교 횟수를 Borůvka 알고리즘은 38%, Prim 알고리즘은 37%, Kruskal 알고리즘은 39%, 역-삭제 알고리즘은 81%와 73%를 단축시키는 효과를 나타내었다.

References

[1] Wikipedia, "Minimum Spanning Tree," http://en.wikipedia.org/wiki/Minimum_spanning_tree, Wikimedia Foundation, Inc., 2007.

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

[3] 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, 2001.

[4] R. C. Prim, "Shortest Connection Networks and Some Generalisations," Bell System Technical Journal, Vol. 36, pp. 1389-1401, 1957.

[5] J. B. Kruskal, "On the Shortest Spanning Subtree and The Traveling Salesman Problem," Proceedings of the American Mathematical Society, Vol. 7, pp. 48-50, 1956.

[6] Wikipedia, "Reverse-Delete Algorithm," http://en.wikipedia.org/wiki/Reverse_delete_algorithm, Wikimedia Foundation, Inc., 2007.

[7] Wikipedia,"Graph(mathematics)," http://en.wikipedia.org/wiki/Graph_(mathematics), Wikimedia Foundation, Inc., 2007.

[8] M. B. Choi, S. U. Lee, "A Prim Minimum Spanning Tree Algorithm for Directed Graph," IWIT, v.12 no.3, 2012.

[9] Yong-Jin Lee, Dong-Woo Lee, "Minimum Cost Spanning Tree Problem in Wireless Sensor Network and Heuristic Algorithm," Journal of Advanced Information Technology and Convergence, pp. 275~282, vol. 7, no. 4, 2009. 8.

저자 소개

이 상 운(정회원)

• 2007.3 ~ 현재 : 강릉원주대학교 멀티미디어공학과 부교수

<주관심분야 : 소프트웨어 척도, 분석과 설계 방법론, 소프트웨어 신뢰성, 그래프 알고리즘>

• e-mail : sulee@gwnu.ac.kr

최 명 복(종신회원)

• 1997년~현재 : 강릉원주대학교 멀티미디어공학과 교수

• 2004년 1월~현재 : 한국인터넷방송통신학회 이사

<주관심분야 : 지능형 정보검색, 소프트웨어 공학, 알고리즘>

• e-mail : cmb5859@gmail.com