평균 지연 시간과 트래픽 용량이 제한되는 스패닝 트리 문제의 2단계 휴리스틱 알고리즘

이 용 진†

요 약

본 연구는 로컬 네트워크의 토폴로지를 설계하거나 루트 노드로부터 여러 개의 통신 경로를 구하는 데 사용될 수 있는 DCMST(Delay constrained Capacitated Minimum Spanning Tree) 문제를 다룬다. 기존의 CMST 문제는 루트 노드의 한 포트가 담당하는 트래픽의 용량에만 제한이 있는 데 비해 DCMST 문제에는 네트워크 평균 지연 시간의 제약 조건이 추가된다. 이 문제는 종단 노드의 트래픽 요구량과 네트워크의 평균 지연 시간을 만족시키는 스패닝 트리의 집합을 구하는 것으로 목적 함수는 전체 링크 비용을 최소로 하는 것이다. 본 연구에서는 트레이드-오프에 기준한 노드 교환 알고리즘과 노드 이동 알고리즘 그리고 평균 지연 알고리즘으로 구성되는 2 단계 휴리스틱을 제시한다. 실제 계산 경험과 성능 분석을 통해 제안한 알고리즘이 평균 지연을 고려한 기존의 CMST 알고리즘보다 비용 측면에 있어 더 우수한 해를 생성할 수 있음을 보였다.

Two Phase Heuristic Algorithm for Mean Delay constrained Capacitated Minimum Spanning Tree Problem

Yong-Jin Lee†

ABSTRACT

This study deals with the DCMST (Delay constrained Capacitated Minimum Spanning Tree) problem applied in the topological design of local networks or finding several communication paths from root node. While the traditional CMST problem has only the traffic capacity constraint served by a port of root node, the DCMST problem has the additional mean delay constraint of network. The DCMST problem consists of finding a set of spanning trees to link end-nodes to the root node satisfying the traffic requirements at end-nodes and the required mean delay of network. The objective function of problem is to minimize the total link cost. This paper presents two-phased heuristic algorithm, which consists of node exchange, and node shift algorithm based on the trade-off criterions, and mean delay algorithm. Actual computational experience and performance analysis show that the proposed algorithm can produce better solution than the existing algorithm for the CMST problem to consider the mean delay constraint in terms of cost.

키워드 : 토폴로지 설계(Topological Design), CMST(Capacitated Minimum Spanning Tree), DCMST(Delay Constrained Capacitated Minimum Spanning Tree), 휴리스틱 알고리즘(Heuristic Algorithm)

1. 서 론

최근의 네트워크 토폴로지는 다양한 응용 분야에 대처하기 위한 혼합형 구조로 진행되고 있으며, 특히 광역 네트워크는 여러 가지 토폴로지를 갖는 네트워크의 인터네트워킹을 요구하고 있는 추세이다. 이를 위해, 광역 네트워크의 구성을 백본과 그에 접속된 다양한 형태의 로컬 네트워크의 혼합형으로 보고, 백본 토폴로지 설계를 하나의 부문제로, 로컬 네트워크의 토폴로지 설계를 또 다른 부문제로 나누어서 해결한다. 여기서 로컬 네트워크의 개념은 보다 광의의 개념으로 LAN을 하나의 종단 노드로 간주할 수 있다[7].

대부분의 경우에 있어서 백본 토폴로지는 신뢰도 향상을 위해 여러 개의 통신 경로를 두는 메쉬형의 분산 토폴로지로 설계한다. 로컬 네트워크는 백본의 어느 한 노드에 접속되는 집중형 토폴로지로 설계하며, 그 구조는 트리나 스타의 형태를 갖는 것이 일반적이다. 이 때, 백본에 있는 노드나 스위치 등의 장비를 루트 노드라고 하면 그것의 포트 한 개가 처리할 수 있는 트래픽 용량에는 제한이 있으므로 로컬 네트워크의 토폴로지는 대개 제한된 수의 종단 노드들이 하나의 트리에 속하고, 이렇게 형성된 여러 개의 트리

† 정 회 원 : 우송대학교 컴퓨터전자정보공학부 교수
논문접수 : 2002년 11월 6일, 심사완료 : 2003년 4월 16일

들이 루트 노드에 연결되는 구조를 갖게 된다. 이러한 최소 연결 비용을 갖는 스패닝 트리의 집합을 구하는 문제를 그래프 용어로 CMST(Capacitated Minimum Spanning Tree) 문제라고 한다[10]. 이 문제는 루트 노드가 자신의 포트를 통해 최소 비용의 브로드캐스트 경로나 멀티캐스트 경로를 구할 때도 이용 가능하며, 종단 노드의 수가 많아지게 되면 계산 시간이 지수적으로 증가하는 NP-complete 문제로 알려져 있다[14].

CMST 문제를 해결하기 위한 알고리즘의 기법은 크게 최적해 기법과 휴리스틱 기법으로 나눌 수 있다. 최적해는 분기 한계(branch and bound) 기법을 사용한 알고리즘[2, 3]과 수리적 모델링을 통한 알고리즘[5, 6]이 있다. 일반적으로 최적해 기법은 노드의 수가 30개 이상인 경우에는 계산 시간이 엄청나게 증가하기 때문에 현실적으로는 휴리스틱 기법이 많이 사용된다. 휴리스틱 기법은 부분 트리에 한번에 하나의 링크를 더하여 스패닝 트리를 구해 나가는 FOGA (first order greedy algorithm)[1, 4]와 FOGA의 해를 개선해 나가는 SOGA(second order greedy algorithm)[4, 8, 9, 11, 13, 16]로 나눌 수 있다. FOGA의 알고리즘 [4]는 모든 종단 노드들을 루트 노드에 연결한 후, 이를 통해 비용이 가장 절감되는 노드 사이에 링크를 삽입하는 방법으로 해를 구하는데, 이 해는 근사 최적해(near optimal solution)로 대개 CMST 문제에 대한 벤치마크(benchmark)로 사용된다. SOGA는 FOGA의 하나에 의해 생성되는 초기 가능해로부터 시작하여 그 해에 특정 링크를 삽입하거나 제거하여 해를 개선해 나가는 알고리즘이다.

그러나, 링 토폴로지 설계에 있어 평균 지연 시간을 고려한 알고리즘[12]은 존재하지만, 위에서 살펴본 바와 같이 CMST 문제에 대한 알고리즘들은 평균 지연 시간을 고려하지 않았다. 따라서 본 연구에서는 노드 수가 많은 경우의 CMST 문제에 대해 네트워크의 평균 지연 시간이 특정 시간 이내가 되어야 한다는 제약 조건을 추가하고 이 문제를 DCMST 문제로 명명한다. 이 문제를 고려하는 이유는 네트워크가 커지는 경우 각 종단 노드들 사이의 통신 지연이 사용자에 대한 서비스 품질을 저하시킬 수 있기 때문이다. 제안하는 DCMST 알고리즘은 기존의 FOGA에 의해 생성되는 노드 집합을 초기해로 이용하는 일종의 SOGA로 2단계로 구성된다. 알고리즘은 트레이드-오프(trade-off) 기준에 의해 트리 사이의 노드를 교환 또는 이동하여 토폴로지를 변화시켜 나가면서 해를 개선해 나간다. 아울러 이 과정에서 요구되는 평균 지연 시간을 만족하도록 링크 용량을 최적으로 할당함으로써 최소 비용 스패닝 트리의 집합을 구한다. 전체 4장으로 구성되는 본 연구는 2장에서 모델링과 알고리즘을, 3장에서는 계산 경험 및 성능 분석을 그리고 4장에서는 결론을 제시한다.

2. DCMST 문제의 모델링 및 알고리즘

2.1 CMST 문제의 모델링

먼저 트리형 토폴로지 설계에 이용되는 기존의 CMST 문제의 모델링을 살펴본다. 임의의 그래프 $G = (V, A)$에서 $V = \{0, 1, 2, \cdots, n\}$이고, 각 노드에서의 트래픽 요구량을 $q_i (i \in V - \{0\})$, 링크 비용을 $(i, j) \in A$인 에지에 대해 d_{ij}라고 하면 CMST 문제는 (그림 1)과 같이 정식화된다[11].

$$\begin{aligned} &Minimize \sum_{i,j} d_{ij} x_{ij} \\ &S.T. \\ &\sum_{i \in R_j} q_i x_{ij} \le MAX \\ &where\ R_j\ specifies\ single\ tree,\ x_{ij} = 0 \text{ or } 1 \end{aligned} \qquad (1)$$

(그림 1) CMST 문제의 모델링

즉, 임의의 스패닝 트리에 소속된 노드들의 트래픽 요구량의 합이 하나의 포트가 지원할 수 있는 최대 트래픽(MAX) 이하가 되는 제약 조건을 만족하면서, 전체 링크 비용을 최소화하는 스패닝 트리의 집합을 구하는 문제이다. 식 (1)에서 R_j는 소속된 노드들 중에서 적어도 하나의 에지가 루트(root node, V = 0)에 연결되는 스패닝 트리이다. CMST 문제의 특별한 경우로 q_i의 값이 전부 1이면 문제는 어떠한 스패닝 트리에도 MAX 개 이상의 노드가 포함될 수 없다는 의미가 된다. 이를 하나의 포트가 고장나는 경우 해당 스패닝 트리에만 고장을 국한시키는 신뢰도 제약 조건이라고 한다.

2.2 DCMST 문제의 모델링

2.2.1 용어 및 기호의 정의

- n : 루트 노드를 제외한 전체 노드 수
- lcnt : 해에 포함되는 트리의 수
- kcnt(p) : 트리 p에 포함되는 노드의 수(p = 1, ···, lcnt)
- cost(p) : 트리 p에 대응되는 링크 비용(p = 1, ···, lcnt)
- q_i : 노드 i에서의 트래픽 요구량(i = 1, ···, n)
- node(p) : 트리 p에 대응되는 노드 집합
- path(p) : node(p)에 대응되는 MST(minimum spanning tree) 상의 링크 집합
- edge(p) : 트리 p에서 루트까지의 최소 연결 비용
- d_{ij} : 노드 쌍 (i,j) 사이의 거리 (i = 0, 1, ···, n ; j = 0, 1, ···, n)
- D : 거리 매트릭스
- d : 링크 용량의 단위 비용
- d_k : 임의의 토폴로지에서 링크 k의 순회 거리 (k = 1, ···, m)
- MAX : 하나의 스패닝 트리에 둘 수 있는 최대 트래픽 (또는 노드 수)

sub1 : 임의의 노드 i가 포함되어 있는 트리의 색인
sub2 : 임의의 노드 j가 포함되어 있는 트리의 색인
ks_{ij} : 노드 교환 알고리즘에서의 노드 쌍 (i, j)의 트레이드-오프(i = 1, ···, n ; j = 1, ···, n)
ks'_{ij} : 노드 이동 알고리즘에서의 노드 쌍 (i, j)의 트레이드-오프 (i = 1, ···, n ; j = 1, ···, n)
T : 네트워크의 평균 지연 시간(초)
T_k : 임의의 토폴로지에 포함된 링크 k의 평균 지연 시간(초)
υ : 전체 트래픽 비율
$1/\mu$: 평균 패킷 길이(bits/packet)
C_k : 임의의 토폴로지에 포함된 링크 k의 용량(bits/초)
λ_k : 임의의 토폴로지에 포함된 링크 k의 평균 도착율(packets/초)
Delay : 원하는 네트워크의 평균 지연 시간(초)
D_{cost} : DCMST 알고리즘의 각 단계에서 얻는 전체 링크 비용
NEW_{cost} : 임의의 토폴로지에 대한 전체 링크 비용

2.2.2 가정

① 패킷의 도착은 포아송 분포(poisson distribution)를 따른다.
② 패킷의 서비스 시간은 지수 분포(exponential distribution)를 따른다.

2.2.3 모델링

이 절에서는 기존의 용량 제한 조건을 만족하는 동시에 네트워크의 전체 평균 지연 시간이 특정한 값보다 작아야 한다는 제약 조건을 추가시켜 DCMST 문제를 모델링하고자 한다. 일반적으로 네트워크의 평균 지연 시간(T)은 식 (2)와 같이 주어진다[17, 18]. 식에서 링크의 개수가 n인 것은 DCMST 문제의 해는 스패닝 트리의 집합이고, 루트를 제외한 노드의 수가 n일 때 스패닝 트리 상의 전체 링크의 수 역시 n이기 때문이다.

$$T = \frac{1}{\upsilon} \sum_{k=1}^{n} \lambda_k T_k \tag{2}$$

가정에 의해 임의의 링크 k는 독립적인 M/M/1 큐로 간주될 수 있으므로 큐잉 이론으로부터 링크 k의 평균 지연 시간(T_k)은 식 (3)으로 유도된다.

$$T_k = \frac{1}{(\mu C_k - \lambda_k)} \tag{3}$$

식 (3)을 식 (2)에 대입하면 식 (4)를 얻는다.

$$T = \frac{1}{\upsilon} \sum_{k=1}^{n} \lambda_k [\frac{1}{(\mu C_k - \lambda_k)}] \tag{4}$$

네트워크의 전체 비용은 선형 함수를 가정하여 식 (5)로 정의한다.

$$D_{cost} = \sum_{k=1}^{n} dC_k d_k \tag{5}$$

이상으로부터 평균 지연 시간을 고려한 DCMST 문제의 모델링은 (그림 2)와 같이 표현된다.

$$Minimize\ D_{cost} = \sum_{k=1}^{n} dC_k d_k$$

$$S.T.$$

$$\frac{\lambda_k}{\mu} \le C_k \tag{6}$$

$$T = \frac{1}{\upsilon}[\sum_{k=1}^{n} \lambda_k \frac{1}{(\mu C_k - \lambda_k)}] \le Delay \tag{7}$$

$$\sum_{i \in R_j} q_i x_{ij} \le MAX \tag{8}$$

where R_j specifies single tree, x_{ij} = 0 or 1

(그림 2) DCMST의 모델링

DCMST 문제는 특정 링크에 흐르는 평균 흐름량(λ_k/μ)이 그 링크의 용량보다 작아야 하고(식 (6)), 네트워크의 평균 지연 시간(T)이 원하는 지연 시간(Delay)보다 작아야 하며(식 (7)), 하나의 트리(R_j)에 둘 수 있는 최대 트래픽이 MAX 이하(식 (8))가 되어야 하는 제약 조건하에서 전체 비용을 최소로 하는 스패닝 트리의 집합을 구하는 것이다. 만일 하나의 트리가 담당하는 노드의 개수가 특정 값보다 작게 하려면 식 (8)을 q_i = 1, ∀i로 바꾸면 된다.

2.3 DCMST 알고리즘

본 연구에서는 CMST 문제를 해결하기 위해 대부분의 휴리스틱 알고리즘이 성능 평가의 척도로 삼고 있는 기존 알고리즘[4]에 의해 생성되는 해를 초기해로 이용한다(앞으로 이 알고리즘을 초기해 알고리즘으로 부르겠다. 물론 임의의 초기해를 사용해도 본 연구에서 제안하는 DCMST 알고리즘의 진행에는 문제가 없다). 초기해 알고리즘은 루트를 제외한 노드의 수가 n일 때 메모리 복잡도는 $O(n^2)$, 시간 복잡도는 $O(n^2 \log n)$으로 표현되는 효율적인 알고리즘으로 CMST 문제에 대한 근사 최적해를 제공한다. DCMST 알고리즘은 먼저 초기해의 각 트리에 포함되는 노드의 집합과 각 트리에서 루트까지의 최소 연결 비용 등을 입력으로 하여 각 노드간의 트레이드-오프를 계산한다. 다음에 트레이드-오프가 큰 순서대로 노드를 교환하여 해를 향상시키는 노드 교환 알고리즘(NEA : node exchange algorithm)과 그 결과 얻어지는 노드의 집합에 대해 또 다른 트레이드-오프를 이용하여 노드를 이동해 가는 노드 이동 알고리즘(NSA : node shift algorithm)을 실행한다. 두 개의 알고리즘은 평균 지연 시간의 제약 조건을 만족하기 위해 링크 용량을 할당하고 전체

비용을 계산하는 평균 지연 알고리즘(MDA : mean_delay_algorithm)을 공통으로 이용한다. DCMST 알고리즘의 전체 과정은 (그림 3)과 같다.

1. *Execute the initial algorithm*
2. **Phase 1** : *Execute node exchange algorithm*
   (*call mean_delay_algorithm*)
   If it is impossible to extend for all spanning trees, then goto 4.
   else
   Phase 2 : *Execute node shift algorithm*
   (*call mean_delay_algorithm*)
3. *If node set is changed, goto 2*
4. *Algorithm is terminated*

(그림 3) DCMST 알고리즘의 진행 과정

2.3.1 노드 교환 알고리즘

노드 교환 알고리즘은 초기해 알고리즘에 의해 얻어진 스패닝 트리의 노드 집합에서 시작하여 노드를 교환함으로써 해를 개선해 나가는 단계이다. 먼저 두 개의 최소 스패닝 트리(sub1, sub2)에 대응되는 노드 집합을 각각 node(sub1), node(sub2)라 하고, 대응되는 링크 비용을 cost(sub1), cost(sub2)라고 하자. 이제 node(sub1)의 한 원소와 node(sub2)의 한 원소를 서로 교환하여 얻은 새로운 최소 스패닝 트리를 (sub1′, sub2′), 대응되는 노드 집합을 node(sub1′), node(sub2′)라하고 링크 비용을 cost(sub1′), cost(sub2′)라고 하면, 그 비용의 차이는

$$sav = cost(sub1) + cost(sub2) - cost(sub1') - cost(sub2') \quad (9)$$

가 된다. 이 때 sav의 값이 양수이면 이전의 해가 개선될 수 있다. 그러나 식 (9)는 각 단계에서 MST 알고리즘의 적용이 많아지게 되므로 본 연구에서는 노드 i와 노드 j의 트레이드-오프를 식 (10)으로 정의한다. 식 (10)에서 edge(sub1)은 노드 i가 소속된 스패닝 트리(sub1)의 노드들 중에서 루트까지의 링크 비용이 최소가 되는 값이다. 즉, edge(sub1) = $\mathrm{Min}\{d_{m0}\}$, $m \in$ node(sub1)이고 마찬가지로 edge(sub2) = $\mathrm{Min}\{d_{m0}\}$, $m \in$ node(sub2)이다.

$$\begin{aligned} &\text{if } sub1 \neq sub2 \text{ and } \sum_{m \in node(sub1)} q_m + q_j - q_i \leq MAX \\ &\qquad \text{and } \sum_{m \in node(sub2)} q_m + q_i - q_j \leq MAX, \\ &\quad ks_{ij} = edge(sub1) + edge(sub2) - d_{ij} \\ &\text{else} \\ &\quad ks_{ij} = -\infty \end{aligned} \quad (10)$$

sub1과 sub2가 같은 경우에는 노드 i와 노드 j가 동일한 스패닝 트리에 소속된 것을 의미한다. 이 경우에는 노드를 교환해도 비용이 같으므로 해에서 배제하기 위해 트레이드-오프의 값을 $-\infty$로 놓는다. 또한, 노드의 트래픽 요구량이 서로 다른 경우에는 sub1에 속한 노드들의 트래픽 요구량의 합과 sub2에 속한 노드들의 트래픽 요구량의 합이 노드 쌍 (i, j)를 교환하더라도 최대 트래픽(MAX)보다 작아야 하므로 $\sum_{m \in node(sub1)} q_m + q_j - q_i \leq MAX$이고 동시에 $\sum_{m \in node(sub2)} q_m + q_i - q_j \leq MAX$이어야 한다. 그렇지 않은 경우에는 $-\infty$로 놓는다.

노드 교환 알고리즘은 먼저 초기해 알고리즘에서 얻은 각 트리의 노드 집합에 대해 평균 지연 알고리즘(MDA)을 호출하여 각 트리의 노드 집합에 대해 요구되는 지연 시간(Delay)을 만족하도록 최적의 링크 용량 할당을 수행하여 초기 비용(NEW_{cost})을 구한다. 그 후에 서로 다른 트리의 노드 집합에 소속된 노드 쌍 (i, j) (i < j)에 대해, 식 (10)을 이용하여 ks_{ij}를 구한다. 동일한 트리의 노드 집합에 소속된 노드 쌍 (i, j)에 대해서는 $ks_{ij} = -\infty$로 놓는다. ks_{ij}의 값이 양의 최대 값을 갖는 노드 쌍 (i, j)부터 시작하여 노드 i와 노드 j를 교환하면 새로운 트리의 노드 집합을 얻는다. 이 변화된 두 개의 트리의 노드 집합에 대해 MST 알고리즘[15]을 적용하여 MST 경로를 구한다. 여기서 두 개의 트리의 노드 집합에만 MST 알고리즘을 적용하는 이유는 나머지 트리의 노드들은 이미 MST 경로 상에 존재하기 때문이다.

다음으로 변화된 전체 비용을 구하기 위해 MDA를 호출하여 반환되는 비용(D_{cost})과 기존의 비용(NEW_{cost})을 비교하여 D_{cost}가 NEW_{cost}보다 작은 경우에는 D_{cost}를 NEW_{cost}로 대치하고 해당되는 ks_{ij}를 $-\infty$로 놓는다. 또한 ks_{ij} 매트릭스를 다시 계산한 후에 위의 과정을 반복한다. 그렇지 않은 경우에는 해당되는 ks_{ij}를 $-\infty$로 놓은 후에 이전의 NEW_{cost}에 대응되는 ks_{ij} 중에서 양의 최대 값을 갖는 노드 쌍 (i, j)에 대해 위의 과정을 반복한다. 즉, 전체 비용이 개선된 경우에는 새로운 토폴로지에 대해 ks_{ij} 매트릭스를 다시 계산하고, 그렇지 않은 경우에는 ks_{ij} 매트릭스를 다시 계산하지 않는다. 노드 교환 휴리스틱은 다음과 같다.

Algorithm : node exchange algorithm (NEA)
input : lcnt, kcnt (p), node (p), path (p) obtained in the initial algorithm
output : NEW_{cost}, lcnt, kcnt (p), node (p), cost (p), path (p)
variable : flag /* flag = 1 ; (initialization), flag = 0 ; (otherwise) */
process :
  step 1 : /* initialization */
    ① flag ← 1 ;
      using path(p) (p = 1, 2, ···, lcnt),
      mean_delay_algorithm( ) ;
    ② NEW_{cost} ← D_{cost} ; /* initial total cost */

  step 2 : /* trade-off (ks_{ij}) computation */
    ① for node pair (i, j) (i < j, ∀(i, j)),
      if (i ∈ node(p)) sub1 ← p ; if (j ∈ node(p))
        sub2 ← p ;
    ② using equation(10), compute ks_{ij} ;
    ③ if ($ks_{ij} \leq 0$) ∀(i, j), algorithm is terminated ;
    ④ sort ks_{ij} with descending order ;

  step 3 : /* new topology generation */

```
while( ksij > 0) {
① for node pair (i, j) with the largest ksij (i, j),
   /* node exchange */
   if (i ∈ node(p)) sub1 ← p ; if (j ∈ node(p))
                    sub2 ← p ;
      node(sub1) ← node(sub1) ∪ j ;
      node(sub1) ← node(sub1) - i ;
      node(sub2) ← node(sub2) ∪ i ;
      node(sub2) ← node(sub2) - j ;
② for node(sub1) and node(sub2),
     by applying MST algorithm, find path(sub1)
     and path(sub2) ;
③ using path(sub1) and path(sub2),
   flag = 0 ;
   mean_delay_algorithm() ;
④ /* total cost comparison */
   if (Dcost ≥ NEWcost) {      /* node restoration */
      node(sub1) ← node(sub1) ∪ i ;
      node(sub1) ← node(sub1) - j ;
      node(sub2) ← node(sub2) ∪ j ;
      node(sub2) ← node(sub2) - i ;
      ksij ← -∞ ; repeat step 3 ; }
   else {
      NEWcost ← Dcost ;
      ksij ← -∞ ; goto step 2 ; }
} ;
```

(그림 4) 노드 교환 알고리즘

2.3.2 평균 지연 알고리즘

이 알고리즘은 노드 교환 및 이동 알고리즘이 중간에서 호출하는 알고리즘이다. 노드 교환 단계에서 호출되는 평균 지연 알고리즘은 초기해에 대해서는 전체 트리에 소속된 링크들에 대해 수행되고, 노드 교환이 일어난 경우에는 교환된 두 개의 트리에 소속된 링크들에 대해서만 수행된다. 알고리즘은 먼저 전체 또는 두 개의 노드 집합에 대응되는 MST 경로상의 링크들에 대해 평균 도착율(λ_k)을 구한다. 다음에 원하는 지연 시간(Delay)을 만족시키기 위해 링크 용량을 할당한다. 이것은 (그림 5)와 같이 정식화 된다.

$$Minimize\ D_{cost} = \sum_{k=1}^{n} dC_k d_k$$

$$S.T.$$

$$\frac{1}{v}\sum_{k=1}^{n}\lambda_k\left[\frac{1}{(\mu C_k-\lambda_k)}\right] = Delay$$

(그림 5) 최적 링크 용량 할당의 정식화

라그랑지 승수(lagrange multiplier) 기법을 사용하여, 최적해를 구하면 식 (11)을 얻는다.

$$C_k = \frac{\lambda_k}{\mu}\left[1+\frac{1}{v\cdot Delay}\frac{\sum_{j=1}^{n}\sqrt{d\lambda_j d_j}}{\sqrt{d\lambda_k d_k}}\right] \qquad (11)$$

마지막으로 알고리즘은 식 (5)를 이용하여 전체 비용(D_{cost})을 계산하여 NEA 또는 NSA로 반환한다. 이상을 정리하면 다음과 같다.

Algorithm : mean_delay_algorithm (MDA)

/* link capacity allocation and total cost computation algorithm */
input : path(p) obtained in NEA or NSA
output : D_{cost}, T_k
variable : flag /* flag = 1 ; (initialization), flag = 0 ; (otherwise) */
temp : vector to save link set (path(p)) on MSTs
process :
{

step 1 : /* traffic flow computation */
if (flag == 1) temp ← path(p), p = 1, 2, ···, lcnt
else temp ← path(p) p = sub1, sub2
for temp,
$\lambda_k \leftarrow \sum q_i$ [for $i \in node(p)$ *corresponding to temp* ;]

step 2 : /* link capacity allocation and mean delay time computation */
for temp {

$$C_k \leftarrow \frac{\lambda_k}{\mu}\left[1+\frac{1}{v\cdot Delay}\frac{\sum_{j=1}^{n}\sqrt{d\lambda_j d_j}}{\sqrt{d\lambda_k d_k}}\right]$$

$$T_k = \frac{1}{(\mu C_k-\lambda_k)}$$

}

$$T \leftarrow \frac{1}{v}\sum_{k=1}^{n}\lambda_k\left[\frac{1}{(\mu C_k-\lambda_k)}\right];$$

step 3 : /* total cost computation */
if (flag == 1) {

$$cost(p) \leftarrow \sum_k dC_k d_k\ (\forall\ k \in path(p));$$

$$D_{cost} \leftarrow \sum_{p=1}^{lcnt} cost(p);$$

}
else {

$$cost(sub1) \leftarrow \sum_k dC_k d_k\ (k \in path(sub1));$$

$$cost(sub2) \leftarrow \sum_k dC_k d_k\ (k \in path(sub2));$$

$$D_{cost} \leftarrow \sum_{p=1,\ p\neq sub1,\ sub2}^{lcnt} cost(p) + cost(sub1) + cost(sub2);$$

}

step 4 : /* control to NEA(or NSA) */
return(D_{cost}) ;

(그림 6) 평균 지연 알고리즘

2.3.3 노드 이동 알고리즘

이 알고리즘은 노드 교환 단계에서 얻어진 정보를 이용하여 특정 트리의 노드 집합에 포함되어 있는 노드를 다른 트리로 이동하는 알고리즘으로, 트레이드-오프 기준에 근거하여 해당 노드를 다른 트리의 노드 집합으로 이동시켜 해를 개선해 나간다. 만일 모든 노드 집합의 트래픽의 합이 MAX인 경우(또는 노드 수가 MAX인 경우)에는 알고리즘은 수행되지 않는다. 그렇지 않다면 서로 다른 트리에 있는 노드 쌍 (i, j)의 트레이드-오프 기준을 계산하고, 노드 j가 이동 가능하다고 하면 node(sub2)에 소속된 노드 j를 노드 i가 소속된 node(sub1)으로 이동한다. 동시에 node(sub2)에서 노드 j를 삭제하여 새로운 노드 집합을 구한다. 사용하는 트레이드-오프는 다음 식으로 정의한다.

$$\text{if } sub1 \neq sub2 \text{ and } \sum_{m \in node(sub1)} q_m + q_j \leq MAX,$$
$$ks'_{ij} = d_{ij} - d_{\max} \quad (12)$$
$$\text{else}$$
$$ks'_{ij} = \infty$$

식 (12)에서는 노드 i가 속한 스패닝 트리 sub1과 노드 j가 속한 스패닝 트리 sub2가 같지 않은 조건하에 스패닝 트리 sub1의 노드 개수에 1을 더한 값이 MAX보다 큰 경우(MAX : 최대 노드 수) 또는 스패닝 트리 sub1에 노드 j를 포함시킨 후의 총 트래픽의 합이 MAX보다 큰 경우(MAX : 최대 트래픽)에는 노드를 이동시킬 수 없으므로 트레이드-오프의 값을 ∞로 놓는다. 그렇지 않은 경우에는 $ks'_{ij} = d_{ij} - d_{max}$로 놓는다. 이 때, d_{max} = Max{edge(sub1), edge(sub2)}이다. 다음에 ks'_{ij}의 값이 음의 최소값을 갖는 노드 쌍 (i, j)부터 시작하여 노드 j를 노드 i가 속한 스패닝 트리로 이동하여 새로운 트리의 노드 집합을 얻는다. 이후의 MST 알고리즘을 적용하는 과정과 MDA를 호출하여 반환되는 비용(D_{cost})과 기존의 비용(NEW_{cost})을 비교하여 처리하는 과정은 NEA에서와 유사하다. 노드 이동 알고리즘은 다음과 같다.

```
Algorithm : node shift algorithm (NSA)
input : NEW_cost, lcnt, kcnt (p), cost (p), node (p), path (p) obtained
        in NEA
output : NEW_cost, lcnt, kcnt (p), cost (p), node (p), path (p)
variable : flag
process :
  step 1 : /* check whether node shift is possible */
           if ∀p, Σ_{j ∈ node(p)} q_j = MAX, algorithm is terminated ;
           else flag ← 0 ;

  step 2 : /* trade-off (ks'_ij) computation */
           ① for node pair (i, j) (i < j, ∀(i, j)),
             if (i ∈ node(p)) sub1 ← p ; if (j ∈ node(p))
                              sub2 ← p ;
           ② using Eq. (12), compute ks'_ij ;
           ③ if (ks'_ij ≥ 0) ∀(i, j), algorithm is terminated ;
           ④ sort ks'_ij with ascending order ;

  step 3 : /* new topology generation */
           while( ks'_ij < 0) {
            ① for node pair (i, j) with the smallest ks'_ij,
              /* node shift */
              if (i ∈ node(p)) sub1 ← p ; if (j ∈ node(p))
                               sub2 ← p ;
              kcnt (sub1) ← kcnt (sub1) + 1 ;
              node (sub1) ← node (sub1) ∪ j ;
              kcnt (sub2) ← kcnt (sub2) - 1 ;
              node (sub2) ← node (sub2) - j ;
           ② same as step 3, ② in NEA
           ③ same as step 3, ③ in NEA
           ④ /* total cost comparison */
              if (D_cost ≥ NEW_cost) {      /* node restoration */
                 kcnt (sub1) ← kcnt (sub1) - 1 ;
                 node (sub1) ← node (sub1) - j ;
                 kcnt (sub2) ← kcnt (sub2) + 1 ;
                 node (sub2) ← node (sub2) ∪ j ;
                 ks'_ij ← ∞ ; repeat step 3 ; }
              else {
                 NEW_cost ← D_cost ;  /* set the reduced cost to
                                         new total cost */
                 ks'_ij ← ∞ ; go to step 2 ; }
  } ;
```

(그림 7) 노드 이동 알고리즘

3. DCMST 알고리즘의 성질 및 성능 분석

3.1 DCMST 알고리즘의 성질

Lemma 1 : DCMST 알고리즘의 메모리 복잡도는 $O(n^2)$이다.

(증명) 메모리에 저장되는 2차원 데이터들은 d_{ij}, ks_{ij}, ks'_{ij} (i = 1, ···, n ; j = 1, ···, n) 등이다. 따라서 알고리즘의 NEA, NSA, MDA의 각 단계에서 메모리 복잡도는 $O(n^2)$이고 초기해 알고리즘의 메모리 복잡도 역시 $O(n^2)$이므로 DC MST 알고리즘의 전체 메모리 복잡도는 $O(n^2)$이다. ■

Lemma 2 : 평균 지연 알고리즘(MDA)의 시간 복잡도는 O(n)이다.

(증명) 생성된 트리를 이용하여 평균 지연 시간을 고려한 해를 구하기 위해서는 DCMST 알고리즘의 MDA만을 적용하면 된다. 임의의 CMST 알고리즘에서 얻는 MST 경로상의 최대 링크 수는 n이므로 MDA에서 계산해야 하는 링크수는 n이다. MDA에서 각 수식의 계산을 위한 시간 복잡도는 O(1)이고, 링크 부하의 최대 값을 얻기 위한 복잡도는 링크수와 동일하므로 O(n)이다. 따라서, 전체 실행 시간의 복잡도는 Maximum[O(1), O(n)] = O(n)이다. ■

Lemma 3 : 초기해 알고리즘의 해에 평균 지연 알고리즘을 적용하면 시간 복잡도는 $O(n^2 \log n)$이다.

(증명) 전체 수행 시간은 초기해 알고리즘의 수행 시간에 평균 지연 알고리즘의 수행 시간을 더한 것이므로 전체 시간 복잡도는 당연히 두 개의 시간 중에서 최대 값을 취해야 한다. 따라서 Maximum[$O(n^2 \log n)$, O(n)] = $O(n^2 \log n)$이다. ■

Lemma 4 : 하나의 트리에 둘 수 있는 최대 노드수가 MAX로 제한되는 DCMST 알고리즘의 시간 복잡도는 $O(n^2 \log n)$이다.

(증명) 먼저, DCMST 알고리즘의 노드 교환 알고리즘(NEA)을 고려해 보자. 알고리즘의 입력은 초기해 알고리즘에 의해 얻어지고 그 복잡도는 $O(n^2 \log n)$이다. step 1에서 입력되는 최대 링크 수는 n이고, Lemma 2에 의해 평균 지연 알고리즘(MDA)의 복잡도는 O(n)이므로 step 1의 복잡도는 O(n)이다. step 2에서 ks_{ij}는 ∀(i, j)(i ∈ node(sub1), j ∈ node(sub2) ; sub1 ≠ sub2)에 대해 계산되고 그 최대수는 1/2(n - MAX)(n + MAX - 1)이므로 시간 복잡도는 $O(n^2)$이다. step 3은 최악의 경우, 최대 ks_{ij} 개수만큼 MDA와 step 2를 반복 수행해야 한다. 먼저, ①의 계산 시간은 $O(n^2)$이다. ②에

서는 2개의 변화된 트리의 노드 집합에만 MST 알고리즘을 적용하면 된다. 사용된 MST 알고리즘[15]의 시간 복잡도는 O(E · log E)이고 E는 에지의 개수로 sparse graph의 경우에는 MAX이고, complete graph의 경우에는 1/2MAX(MAX + 1)이다. 그런데 최대 ks_{ij} 개수만큼 MST 알고리즘을 반복하더라도 MAX와 MAX^2은 상수이므로 시간 복잡도는 $O(n^2)$이 된다. ③에서 MDA는 2개의 변화된 트리에만 적용되므로 계산되어야 하는 링크의 최대 개수는 2MAX이다. 최악의 경우, 최대 ks_{ij} 개수만큼 MDA가 반복될 수 있으므로 시간 복잡도는 $O(n^2)$이다. 따라서 NEA의 step 3의 전체 시간 복잡도는 Maximum[$O(n^2)$, $O(n^2)$, $O(n^2)$] = $O(n^2)$이다. 이로부터 NEA의 전체 시간 복잡도는 Maximum[step 1의 복잡도, step 2의 복잡도, step 3의 복잡도] = Maximum[O(n), $O(n^2)$, $O(n^2)$] = $O(n^2)$이다. 동일한 방법으로 노드 이동 알고리즘(NSA)의 시간 복잡도 역시 $O(n^2)$이 된다. 그러므로 DCMST 알고리즘의 전체 실행 시간은 초기해 알고리즘의 실행 시간 + NEA 실행 시간 + NSA 실행 시간 = Maximam [$O(n^2 \log n)$, $O(n^2)$, $O(n^2)$] = $O(n^2 \log n)$이다. ■

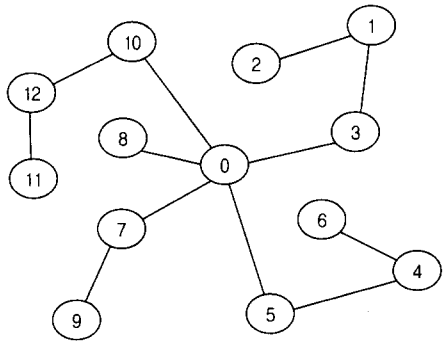
3.2 계산 예

DCMST 알고리즘의 진행 과정을 설명하기 위해 노드수(n)는 12, 각 노드에서의 트래픽 요구량은 1($q_i = 1$; i = 1, …, 12), 하나의 포트에 연결될 수 있는 최대 트래픽(MAX)은 3, 원하는 네트워크의 평균 지연 시간(Delay)은 1ms인 문제를 고려한다. 평균 패킷 길이($1/\mu$)는 1000 비트/패킷이고 편의를 위해 d와 C_k는 각각 1로 한다. 이 때 루트의 색인은 0이다. 거리 매트릭스(d_{ij})는 대칭형(symmetric)으로 <표 1>과 같다[11].

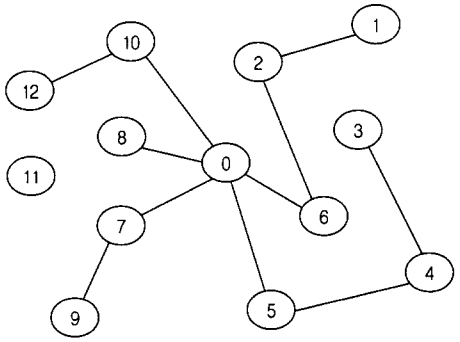
〈표 1〉 거리 매트릭스

	0	1	2	3	4	5	6	7	8	9	10	11	12
1	41												
2	40	2											
3	31	17	18										
4	24	19	19	8									
5	17	28	28	14	9								
6	23	23	21	23	16	20							
7	11	45	44	30	26	17	31						
8	7	48	47	38	31	24	29	13					
9	20	57	56	42	38	28	42	12	18				
10	22	63	61	53	46	39	42	25	15	22			
11	29	69	69	55	50	42	52	25	23	15	17		
12	31	73	72	61	55	47	53	32	24	24	11	12	

먼저, 초기해 알고리즘을 적용하면 (그림 8)을 얻으며, 그 스패닝 트리의 노드 집합은 {1, 2, 3}, {4, 5, 6}, {7, 9}, {8}, {10, 11, 12}이다. 이제 NEA의 step 1, ②에서 MDA를 호출하여 전체 비용을 계산하면 <표 2>를 얻는다.



(그림 8) 초기해의 토폴로지(비용 : 167)



(그림 9) NEA의 토폴로지(비용 : 155)

〈표 2〉 초기해에 평균 지연 알고리즘을 적용한 결과

tree(p)	cost(p)	path(p)	link cost	lambda (λ_k)	capacity (C_k : Mbps)	delay time (T_k : ms)
1	45.0	0-10 10-12 12-11	22.0 11.0 12.0	3.0 2.0 1.0	4.6 3.8 2.6	0.22 0.27 0.38
2	7.0	0-8	7.0	1.0	2.6	0.38
3	23.0	0-7 7-9	11.0 12.0	2.0 1.0	3.8 2.6	0.27 0.38
4	42.0	0-5 5-4 4-6	17.0 9.0 16.0	3.0 2.0 1.0	4.6 3.8 2.6	0.22 0.27 0.38
5	50.0	0-3 3-1 1-2	31.0 17.0 2.0	3.0 2.0 1.0	4.6 3.8 2.6	0.22 0.27 0.38
네트워크의 평균 지연 시간 : 1 ms 전체 비용 : 167.0						

이제, NEA의 step 2에서 서로 다른 트리에 속한 노드 쌍 (i, j)에 대해서만 트레이드-오프, ks_{ij} = edge(sub1) + edge(sub2) - d_{ij}를 계산한다. 여기서, edge(sub1)은 노드 i가 소속된 스패닝 트리 sub1의 노드 중에서 루트까지의 최단 거리, edge(sub2)는 노드 j가 소속된 스패닝 트리 sub2의 노드 중에서 루트까지의 최단거리, d_{ij}는 노드 쌍 (i, j) 사이의 링크 비용이다. 예를 들어, $ks_{1,4}$를 계산하려면, 먼저 위의 스패닝 트리의 노드 집합에서 노드 1이 소속된 스패닝 트리의 색인 sub1 = 1, 노드 4가 소속된 스패닝 트리의 색인 sub2 = 2를 발견한다. 다음으로 edge(sub1) = min{d_{01}, d_{02},

d_{03}} = min{41, 40, 31} = 31과 edge(sub2) = min{d_{04}, d_{05}, d_{06}} = min{24, 17, 23} = 17을 구한다. 따라서, $ks_{1,4}$ = 31 + 17 - 19 = 29가 된다. 동일한 스패닝 트리에 소속된 노드 쌍(i, j)에 대해서는 -∞로 놓는다. 이런 식으로 계산된 트레이드-오프 매트릭스(ks_{ij})는 <표 3>과 같다.

〈표 3〉 초기 ks_{ij} 매트릭스

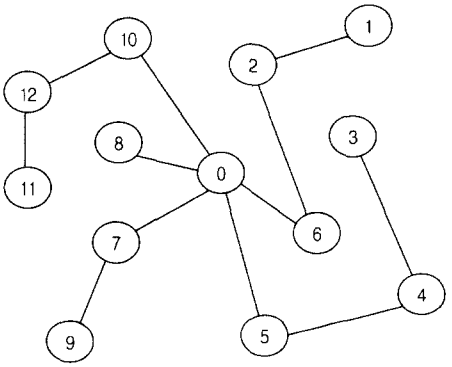
	1	2	3	4	5	6	7	8	9	10	11	12
1	-∞											
2	-∞	-∞										
3	-∞	-∞	-∞									
4	29	29	40	-∞								
5	20	20	34	-∞	-∞							
6	25	27	25	-∞	-∞	-∞						
7	-3	-2	12	2	11	-3	-∞					
8	14	15	24	17	24	19	29	-∞				
9	-15	-14	0	-10	0	-14	-∞	24	-∞			
10	-10	-8	0	-7	0	-3	8	38	11	-∞		
11	-16	-16	-2	-11	-3	-13	8	30	18	-∞	-∞	
12	-20	-19	-8	-16	-8	-14	1	29	9	-∞	-∞	-∞

표에서 제일 큰 값을 선택하면 노드 쌍 (3, 4)에 대해 $ks_{3,4}$ = 40이다. 노드 3과 노드 4를 교환하면 스패닝 트리의 노드 집합은 {1, 2, 4}, {3, 5, 6}, {7, 9}, {8}, {10, 11, 12}가 된다. 변화된 노드 집합 {1, 2, 4}와 {3, 5, 6}에 MST 알고리즘을 적용한 후에 NEA의 step 3, ③에서 mean_delay_algorithm을 호출하여 전체 비용을 계산하면 171이 된다. 이 값은 초기 해의 비용인 167보다 크므로 스패닝 트리의 노드 집합을 다시 원래대로 환원한 후에 가장 큰 트레이드-오프를 구하면 $ks_{8,10}$ = 38을 얻는다. 이런 식으로 알고리즘을 계속 적용한 결과는 (그림 9) 및 <표 4>와 같다.

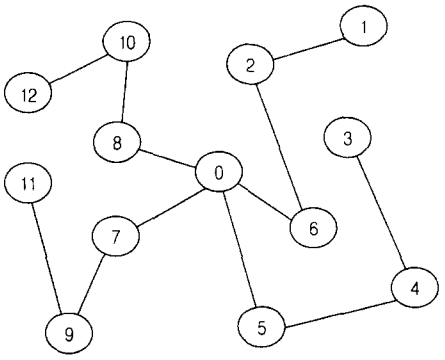
〈표 4〉 NEA와 NSA의 해

tree(p)	cost(p)	path(p)	link cost	lambda (λ_k)	capacity (C_k : Mbps)	delay time (T_k : ms)
1	45.0	0-10	22.0	3.0	4.6	0.22
		10-12	11.0	2.0	3.8	0.27
		12-11	12.0	1.0	2.6	0.38
2	7.0	0-8	7.0	1.0	2.6	0.38
3	23.0	0-7	11.0	2.0	3.8	0.27
		7-9	12.0	1.0	2.6	0.38
4	34.0	0-5	17.0	3.0	4.6	0.22
		5-4	9.0	2.0	3.8	0.27
		4-3	8.0	1.0	2.6	0.38
5	46.0	0-6	23.0	3.0	4.6	0.22
		6-2	21.0	2.0	3.8	0.27
		2-1	2.0	1.0	2.6	0.38
네트워크의 평균 지연 시간 : 1 ms 전체 비용 : 155.0						

NSA를 적용하면 비용은 155로 NEA와 동일하지만, NEA의 노드 집합의 개수는 5이고 그 집합은 {1, 2, 6}, {3, 4, 5}, {7, 9}, {8}, {10, 11, 12}인데 비해, NSA의 노드 집합의 개수는 4이고 그 집합은 {1, 2, 6}, {3, 4, 5}, {7, 8, 9}, {10, 11, 12}이다. 그런데 (그림 10)에서 루트에서의 에지 수가 5인 이유는 집합 {7, 8, 9}에 MST 알고리즘을 적용한 결과 (0-8), (0-7-9)인 2 개의 MST가 얻어지기 때문이다. 따라서 노드 집합이 변경되었고 이를 이용하여 다시 NEA를 적용하면 비용이 151인 (그림 11)과 <표 5>가 얻어지며, 모든 스패닝 트리 p (p = 1, 2, 3, 4)에 대해 더 이상 확장될 수 없으므로 알고리즘이 종료된다.



(그림 10) NSA의 토폴로지(비용 : 155)



(그림 11) 두 번째 NEA의 토폴로지(비용 : 151)

〈표 5〉 두 번째 NEA의 해

tree(p)	cost(p)	path(p)	link cost	lambda (λ_k)	capacity (C_k : Mbps)	delay time (T_k : ms)
1	33.0	0-8	7.0	3.0	4.8	0.21
		8-10	15.0	2.0	3.9	0.26
		10-12	11.0	1.0	2.8	0.36
2	38.0	0-7	11.0	3.0	4.8	0.21
		7-9	12.0	2.0	3.9	0.26
		9-11	15.0	1.0	2.8	0.36
3	34.0	0-5	17.0	3.0	4.8	0.21
		5-4	9.0	2.0	3.9	0.26
		4-3	8.0	1.0	2.8	0.36
4	46.0	0-6	23.0	3.0	4.8	0.21
		6-2	21.0	2.0	3.9	0.26
		2-1	2.0	1.0	2.8	0.36
네트워크의 평균 지연 시간 : 1 ms 전체 비용 : 151.0						

3.3 시뮬레이션

DCMST 알고리즘을 평가하기 위해 노드의 좌표를 x = [0, 100], y = [0, 100] 사이에서 일양 분포(uniform distribution)로 난수 발생하고, 대응되는 유클리디안 거리는 식 (13)을 사용한다. 즉, 임의의 2개의 노드 좌표 (x_1, y_1), (x_2, y_2)에 대해, 두 좌표 사이의 거리를

$$d_{ij} = \lfloor (\sqrt{(x_1-x_2)^2 + (y_1-y_2)^2} + 0.5) \rfloor \times 100 \quad (13)$$

으로 정의하고 거리 자체를 비용(d_{ij})으로 간주하며, 그 값은 대칭으로 가정한다.

루트 노드의 위치에 따라 네트워크 장비가 좌표(0, 0)에 위치하는 것을 TC로, 좌표(50, 50)에 위치하는 것을 TE로 한다. 이제 지수 난수(exponential random number)를 X, 전체 네트워크의 평균 트래픽 비율을 υ라고 하면,

$$X = \frac{1}{\upsilon} \ln(\frac{1}{1-U}) \quad (14)$$

이다. 식 (14)에서 U는 구간 [0, 1]에서 일양 분포를 하는 확률 변수이다. 식 (14)를 이용하는 방법을 사용하여 n개의 포아송 난수(poisson random number)를 발생시키고, 그 값들을 각 노드에서의 트래픽 요구량으로 하였다. υ는 10, 20, 30, 40, 50, n은 30, 50, 70을 사용하고, 최대 트래픽(MAX)은 시뮬레이션 결과 얻어진 값 중 최대 값에서부터 n/4 까지를 사용하였다. 사용 언어는 FORTRAN-77과 C 언어이고 실행 환경은 PC이다. 식 (15)는 초기해 알고리즘에 의해 생성된 트리에 평균 지연 알고리즘을 적용하여 얻은 전체 비용에 대한 DCMST 알고리즘의 절감률을 표시한다. 식 (15)에서 V[CMST]는 초기해 알고리즘에 평균 지연을 고려한 비용이고, V[DCMST]는 DCMST 알고리즘에 의한 비용이다.

$$SV = \frac{V[CMST] - V[DCMST]}{V[CMST]} \times 100 \quad (15)$$

TC 및 TE 테스트에 대해 시뮬레이션을 수행하고 식 (15)에 대한 최대 값의 결과를 정리한 것이 <표 6>에 나타나 있다. 표에서 보듯이, 절감률은 TC 테스트의 경우가 최대 4.5%, TE 테스트의 경우가 최대 5.6%로 TE 테스트의 경우가 보다 우수하게 나타나고 있다. 실제 시뮬레이션의 수행 시간은 앞의 Lemma 4에서 이미 증명한 바가 있지만, 초기해 알고리즘의 수행 시간을 1로 했을 때 TC 테스트인 경우가 1.4~2.54, TE 테스트인 경우가 1.2~2.03이었다. 이 값에는 초기해 알고리즘의 실행 시간이 포함되어 있으므로 NEA, NSA, MDA만의 전체 실행 시간은 각각 0.4~1.54(TC), 0.2~1.03(TE)이 된다. 따라서 Lemma 4에서 증명한 시간 복잡도와 시뮬레이션의 수행 시간이 근사함을 알 수 있다. 한편 최대 트래픽(MAX)의 증가 및 노드수의 증가는 절감률에 아무런 영향을 주지 못하고 있다. 이러한 결과로부터 DCMST 문제는 노드수, 최대 트래픽 그리고 링크 비용 매트릭스와 해 사이의 관계식을 유도하기가 어려움을 알 수 있다.

<표 6> 최대 절감률

테스트	노드수 \ υ	10	20	30	40	50
TC	30	-	0.53	1.81	1.03	2.34
	50	0.18	4.45	1.81	1.17	2.62
	70	1.22	1.78	2.32	1.63	2.01
TE	30	-	0.94	1.86	2.24	2.20
	50	0.18	5.59	2.35	2.34	5.15
	70	1.01	1.23	1.67	2.72	3.79

4. 결 론

본 연구에서는 포트의 용량과 평균 지연 시간의 제약 조건을 만족하면서 루트 노드로부터 종단 노드들을 담당하는 최소 비용의 스패닝 트리의 집합을 얻는 알고리즘을 제시하였다. 제안한 알고리즘은 기존의 CMST 문제의 제약 조건에 평균 지연 시간의 제약 조건을 추가하여 확장한 DCMST 문제를 해결한다. 메모리 및 시간 복잡도는 기존의 CMST 벤치마크 알고리즘과 같으며 노드 교환 및 노드 이동의 2단계를 통하여 효율적으로 해를 개선해 나간다. 한편 알고리즘의 성능에 대한 몇 가지 특성이 유도되었으며, 시뮬레이션 결과 비교적 단시간 내에 개선된 해를 구할 수 있었다. 알고리즘은 로컬 네트워크의 트리 토폴로지 설계 분야와 스위치 등의 통신 장비로부터 단말 노드들까지의 브로드캐스트나 멀티캐스트 경로를 구하는 데 적용될 수 있다. 앞으로의 연구에서는 혼합형 토폴로지를 설계하는 경우의 알고리즘이 기대된다.

참 고 문 헌

[1] V. Ahuja, "Design and Analysis of Computer Communication Networks," McGraw-Hill, pp.115-149, 1985.

[2] K. M. Chandy and T. Lo, "The Capacitated Minimum Spanning Tree," Networks, pp.173-181, 1973.

[3] K. M. Chandy and R. A. Russell, "The Design of Multipoint Linkages in a Teleprocessing Tree Network," IEEE Trans. on Computers, Vol.21, No.10, pp.1062-1066, 1972.

[4] L. R. Esau and Williams, "On teleprocessing system design, part II," IBM Syst. J., Vol.5, No.3, pp.142-147, 1966.

[5] B. Gavish, "Formulations and Algorithms for the capacitated minimal directed tree problem," Journal of ACM, Vol. 30, pp.118-132, 1983.

[6] B. Gavish, "Augmented Based Algorithms for Centralized

Network Design," IEEE Trans. on. Comm., Vol.33, No.12, pp.1247-1257, 1985.

[7] B. Gavish, "Topological Design of Telecommunication Networks-Local Access Design Methods," Annals of Operations Research, Vol.33, No.1, pp.17-71, 1991.

[8] B. Gavish and K. Altinkemer, "Parallel Savings Heuristic for the Topological Design of Local Access Tree Networks," Proceedings IEEE-INFOCOM '86, pp.139-139, 1986.

[9] A. Kershenbaum, R. Boorstyn and R. Oppenheim, "Second-Order Greedy Algorithms for Centralized Teleprocessing Network Design," IEEE Trans. on Comm., Vol.28, No.10, pp.1835-1838, 1980.

[10] A. Kershenbaum, "Telecommunication Network Design Algorithm," McGraw-Hill, pp.183-200, 1993.

[11] Y. Lee and T. Kim, "An Algorithm for the Capacitated Minimum Spanning Tree in Local Area Network," Journal of the Korea Information Science Society, Vol.21, No.6, pp. 1097-1106, 1994.

[12] Y. Lee and T. Kim, "Two-Phase Heuristic Algorithm for Delay Constrained Minimal Cost Loop Problem," International Journal of Network Management, Vol.6, No.3, pp. 142-149, 1996.

[13] Y. Lee and D. Lee, "Two-Phase Algorithm for the Topological Design of Local Access Tree Networks," Proceedings of the IASTED International Conference- Networks, pp.67-70, 1996.

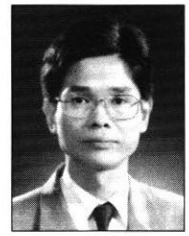
[14] C. H. Papadimitriou, "The Complexity of the Capacitated Tree Problem," Networks, Vol.8, pp.217-230, 1978.

[15] R. Sedgewick, "Algorithms," Addison-Wesley, pp.452-461, 1989.

[16] R. Sharma, "Design of an Economical Multidrop Network Topology with Capacity Constraints," IEEE Trans. on Comm., Vol.31, No.4, pp.590-591, 1983.

[17] A. S. Tanenbaum, "Computer Networks," pp.62-87, Prentice-Hall, 1981.

[18] Y. Zheng and S. Akhtar, "Networks for Computer Scientists and Engineers," Oxford University Press, pp.364-372, 2002.



이 용 진

e-mail : yjlee@woosong.ac.kr

1983년 고려대학교 산업공학과(학사, 석사)

1995년 고려대학교 전산과학과(박사)

1995년~현재 우송대학교 컴퓨터전자정보 공학부 부교수

관심분야 : 네트워크 성능평가, 인터넷 QoS, 분산 시스템