韓國 컴퓨터情報學會 論文誌
第13卷 第4號, 2008. 7.

2008-13-4-3-4

케이블 TV 망에서 노드 선택을 위한 휴리스틱 연구

정 균 락*

Heuristics for Selecting Nodes on Cable TV Network

Kyun Rak Chong*

요 약

케이블 TV망은 분배센터에서 가입자에게 방송 신호를 내려 보내는데, 하이브리드 파이버 동축케이블(HFC)이 사용된 뒤로는 상향채널을 인터넷 같은 광대역 서비스로 확장해 활용하고 있다. 그런데 이 상향채널은 잡음에 취약한데 한 노드의 증폭기에 누적된 자식노드로부터의 잡음이 어떤 수준을 넘게 되면, 잡음이 더 이상 전파되는 것을 막기 위해 해당되는 노드를 분리하는 것이 필요하게 된다. 각 노드에 이익이 주어질 때 노드 선택 문제(NSP)는 각 노드에 누적된 잡음이 주어진 임계값을 넘지 않으면서 선택된 노드의 이익의 합이 최대가 되게 노드들을 선택하는 문제인데 NP-hard임이 증명되어 있다. 본 논문에서는 NSP의 근사해를 구하는 휴리스틱들을 제안하고 비교 분석하였는데, 구간 분할 휴리스틱이 greedy 휴리스틱보다 더 우수한 결과를 보였다. 이 휴리스틱들은 HFC 운영 시스템에 구현되어, 사용료를 더 많이 지불하는 우수 고객들에 해당하는 노드를 케이블 TV망에서 가능한 분리하지 않음으로써 더 좋은 질의 서비스를 제공하는 데 사용할 수 있다.

Abstract

The cable TV network has delivered downward broadcasting signals from distribution centers to subscribers. Since the traditional coaxial cable has been upgraded by the Hybrid Fiber Coaxial(HFC) cable, the upward channels has expanded broadband services such as Internet. This upward channel is vulnerable to ingress noises. When the noises from the children nodes accumulated in an amplifier exceeds a certain level, that node has to be cut off to prevent the noise propagation. The node selection problem(NSP) is defined to select nodes so that the noise in each node does not exceed the given threshold value and the sum of profits of selected nodes can be maximized. The NSP has shown to be NP-hard. In this paper, we have proposed heuristics to find the near-optimal solution for NSP. The experimental results show that interval partitioning is better than greedy approach. Our heuristics can be used by the HFC network management system to provide privileged services to the premium subscribers on HFC networks.

▸ Keyword : 케이블 TV 망(Cable TV network), Hybrid Fiber Coaxial, 노드선택(Node selection)

• 제1저자 : 정균락
• 접수일 : 2008. 4. 10, 심사일 : 2008. 5. 30, 심사완료일 : 2008. 7. 25.
* 홍익대학교 컴퓨터공학과 교수
※ 이 논문은 홍익대학교 교수 연구년 기간 (2006.9-2007.8) 중 연구되었음.

Ⅰ. 서 론

케이블 TV 망은 일반적으로 분배센터에서 가입자에게 방송 신호를 내려 보내는 데, 최근에는 가입자로부터 분배센터로 가는 상향 채널(upward channel)을 인터넷과 전화와 같은 광대역 서비스로 확장하여 그 효용성을 증가시키고 있다 [1, 3, 4, 5].

케이블 TV 망은 HFC(hybrid fiber coaxial) 기술을 사용하고 있으며, 750MHz를 커버하고 있는 데 65MHz이하의 대역이 상향 채널을 위해 사용되고 있다 [1]. 상향 채널은 제한된 대역폭을 가지고 있을 뿐만 아니라 잡음에도 매우 취약하다 [8][9]. HFC 망에서 ONU(optical node unit)들은 분배센터와 광케이블로 연결되어 있으며 스타 망 형태를 이루고 있다. ONU는 하향 전송 시에는 광 신호를 RF(radio frequency) 신호로 변환하고, 상향 전송 시에는 RF 신호를 광 신호로 변환한다. ONU와 가입자들은 트리 구조로 연결되어 있으며 트리 구조에서 각 노드는 증폭기나 분배기이다. 어떤 노드가 작동하지 않거나 그 노드를 연결하는 케이블이 끊어지면 ONU와 송수신을 할 수 없게 되며, 그렇지 않은 경우에도 잡음에 의해 송수신이 불가능할 수 있다.

동축 케이블을 사용하는 트리 네트워크에서 각 증폭기 노드는 자식 노드들의 신호들을 합병하게 되는 데, 합병된 신호는 증폭되어 부모 노드로 전달된다. 신호와 혼합된 잡음 역시 증폭되어 위쪽으로 전파된다. 이 때 잡음이 어떤 임계값을 넘게 되면 신호가 잡음과 구별되지 않게 되므로 이런 노드들은 네트워크에서 분리할 필요가 생기게 된다.

기존에 사용하고 있는 방법은 어떤 노드의 잡음의 합이 임계값을 넘으면 그 노드를 단순히 케이블 TV 망에서 분리하는 데, 이 방법은 일반 가입자에 해당하는 노드와 프리미엄 가입자에 해당하는 노드를 구별하지 않고 분리하는 단점이 있다 [7]. 본 논문에서 제안하는 방법은 각 노드의 잡음과 이익(프리미엄)을 함께 고려하여 이익의 합이 최대가 되게 서비스 할 노드를 선택하기 때문에, 프리미엄 가입자들이 가능한 최대로 선택되어 더 좋은 질의 서비스를 제공받을 수 있다 .

각 노드에 이익이 주어질 때, 선택된 모든 노드들의 잡음의 합이 각 노드들에 주어진 임계값을 넘지 않으면서 이익의 합이 최대가 되도록 트리 네트워크에서 노드들을 선택하는 노드 선택 문제(node selection problem, 이하 NSP)는 [7]에서 NP-hard임이 증명되었다 [2]. 그러므로 가입자의 수가 많은 케이블 TV 망에서는 실행시간이 많이 걸리기 때문에 NSP를 해결하기 위해 최적 알고리즘을 사용할 수 없게 된다.

본 연구에서는 NSP의 근사해(near optimal solution)를 구하기 위한 greedy 휴리스틱과 구간 분할(interval partitioning)을 이용하는 휴리스틱을 제안하고 실험을 통해 그 결과를 비교 분석하였다. 본 연구에서 개발된 휴리스틱들은 HFC 네트워크 운영 시스템이 장착되어 우수고객들에게 더 좋은 질의 서비스를 제공하는 데 사용할 수 있다.

Ⅱ. 노드 선택 문제

본 연구에서는 NSP를 다음과 같이 정의한다 [7]. NSP는 각 노드 i가 이익 p_i, 잡음 n_i, 임계값 c_i를 가질 때, 다음의 제한조건을 만족하면서 이익의 합이 최대가 되도록 노드들을 선택하는 문제이다.

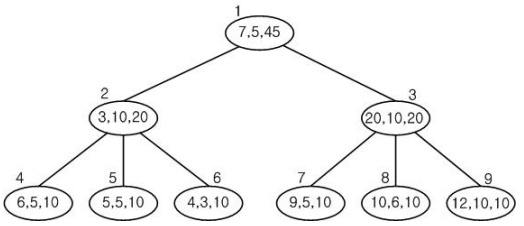

그림 1. 노드 선택 문제의 예
Fig 1. NSP Example

조건 1 : 노드 k가 루트인 서브트리를 S_k라 하고, S_k 안에서 선택된 노드들의 집합을 T_k라 할 때,
T_k 안에 있는 노드들의 잡음의 합은 c_k를 넘을 수 없다.
즉, 수식 (1)을 만족하여야 한다.

$$\forall k \quad \sum_{j \in T_k} n_j \leq c_k \quad \cdots\cdots (1)$$

조건 2 : 노드 k가 선택되지 않으면 S_k 안의 모든 노드는 선택되지 않는다. 그러므로 노드 k가 선택되기 위해서는 루트로부터 노드 k에 이르는 경로에 있는 모든 노드들이 선택되어야 한다.

〈그림 1〉에 NSP의 예가 나타나 있는데 노드 안의 숫자들은 (p_k, n_k, c_k)이고 노드 밖의 숫자는 노드의 번호이다. 선택된 노드의 번호는 1, 2, 3, 4, 5, 9이고, 이익의 합은 53이다. 조건 1을 만족시키는지 확인해 보면 먼저 모든 잎노드(leaf node)의 잡음은 임계값을 넘지 않음을 알 수 있다. 그 다

음 S_2를 보면 노드 2, 4, 5가 선택되었는데 각 노드의 잡음의 합 10+5+5는 20(c_2)를 넘지 않으며, S_3를 보면 노드 3과 9가 선택되었는데 각 노드의 잡음의 합 10+10은 20(c_3)를 넘지 않는다. S_1을 보면 노드 1, 2, 3, 4, 5, 9가 선택되었는데 각 노드의 잡음의 합 5+10+10+5+5+10은 45(c_1)을 넘지 않는다. 조건 2를 만족시키려면 노드 4와 노드 5가 선택되었으므로 노드 2와 노드 1이 선택되어야 하며, 노드 9가 선택되기 위해서는 노드 3과 노드 1이 선택되어야 하는 데 맞게 선택되었음을 알 수 있다.

먼저 이 문제를 해결하기 위해 필요한 용어를 정의하기로 한다 [7]. D_i를 S_i 안에 있는 노드들이 선택될 수 있는 모든 경우에 대한 쌍(pair) (p ,n)의 집합이라 하자. 여기서 p는 선택된 노드들의 이익의 합이고 n은 이 때의 잡음의 합이다. 즉, 수식 (2)에 의해 계산된다.

$$D_i = \{(p,n) \mid \forall T_i \subseteq S_i \quad p = \sum_{j \in T_i} p_j, n = \sum_{j \in T_i} n_j \le c_i \} \quad \cdots\cdots (2)$$

다음에는 D_i를 계산하기 위한 연산들을 정의한다.

정의 J(D_i, D_j, c)

$$J(D_i, D_j, c) = \{(p, n) \mid (p, n) \in D_i, n \le c\} \cup \{(p, n) \mid (p, n) \in D_j, n \le c\} \cup \{(p+p', n+n') \mid (p, n) \in D_i, (p', n') \in D_j, n+n' \le c\} \quad \cdots\cdots (3)$$

$J(D_i, D_j, c)$는 S_i와 S_j로부터 노드들이 선택될 수 있는 모든 경우에 대한 (p, n)의 집합 중 잡음의 합이 c를 넘지 않는 쌍들의 집합을 나타내고 있다. 만약에 $p_j \le p_k$이고 $n_j \ge n_k$이면 (p_k, n_k)가 (p_j, n_j)를 지배한다(dominate)고 한다. 지배되는 쌍은 이익은 적은데 잡음이 많은 경우이므로 삭제할 수 있다, 앞으로 $J(D_i, D_j, c)$의 계산에서는 지배되는 쌍들은 모두 제거되는 것으로 가정한다.

정의 F(D_1, D_2, ... , D_k, c)

(1) $F(D_i, D_j, c) = J(D_i, D_j, c)$ ····· (4)

(2) $F(D_1, D_2, \ldots\ldots, D_k, c)$
$= F(F(D_1, D_2, \ldots\ldots, D_{k-1}, c), D_k, c)$ ···· (5)

$F(D_1, D_2, \ldots\ldots, D_k, c)$는 $S_1, S_2, \ldots\ldots, S_k$로부터 노드들이 선택될 수 있는 모든 경우에 대한 (p, n)의 집합 중 잡음의 합이 c를 넘지 않는 쌍들의 집합을 나타낸다.

노드 i가 있을 때 노드 i의 자식을 i_1, i_2, ..., i_k라 하고, 대응되는 쌍의 집합을 D_{i1}, D_{i2}, ..., D_{ik}라 하면 D_i를 다음과 같이 구할 수 있다.

$$D_i = \{(p_i, n_i)\} \cup \{(p+p_i, n+n_i) \mid (p, n) \in F(D_{i1}, D_{i2}, \ldots\ldots, D_{ik}, c_i-n_i) \} \cdots (6)$$

즉 노드 i의 자식들이 선택되기 위해서는 조건 2에 의해 노드 i도 선택되어야 하므로 S_{i1}, S_{i2}, ... , S_{ik}에서 선택된 노드들의 잡음의 합은 c_i-n_i를 넘을 수 없다. 그러므로 $F(D_{i1}, D_{i2}, \ldots\ldots, D_{ik}, c_i-n_i)$를 구하고 여기에 속한 모든 쌍 (p, n)에 각각 p_i와 n_i를 더해주면 된다.

주어진 NSP 문제에서 최대 이익을 알기 위해서는 잎노드로부터 루트 노드에 이르기까지 집합 D_i를 계산하고, 루트가 노드 1이라 하면 D_1에서 이익이 최대가 되는 (p, n)을 찾으면 된다.

〈그림 1〉의 예에서 보면 먼저 $D_4 = \{(6, 5)\}$, $D_5 = \{(5, 5)\}$, $D_6 = \{(4, 3)\}$이다. D_2를 계산하려면 $J(D_4, D_5, 10) = \{(6, 5), (5, 5), (11, 10)\}$인데 쌍 (6, 5)가 쌍 (5, 5)를 지배하므로 (5, 5)는 삭제되고 $J(D_4, D_5, 10) = \{(6, 5), (11, 10)\} = F(D_4, D_5, 10)$이 된다. $F(D_4, D_5, D_6, 10) = \{(4, 3), (6, 5), (10, 8), (11, 10)\}$이고 따라서 $D_2 = \{(3, 10), (7, 13), (9, 15), (13, 18), (14, 20)\}$이 된다. 같은 방법으로 D_3를 계산하면 $D_3 = \{(20, 10), (29, 15), (30, 16), (32, 20)\}$이 되고, D_1을 계산하면 $D_1 = \{(7, 5)\}, (27, 15), (36, 20), (37, 21), (39, 25), (40, 31), (43, 33), (44, 34), (45, 35), (46, 36), (49, 38), (50, 39), (51, 41), (52, 43), (53, 45)\}$가 된다. D_1에서 제일 마지막에 있는 쌍이 (53, 45)이므로 최대 이익은 53이 되고, 그 때 잡음의 합은 45가 된다.

III. Greedy 휴리스틱과 구간 분할을 이용한 휴리스틱 제안

NSP가 NP-hard이므로 D_i에 속한 (p, n) 쌍의 개수가 기하급수적으로(exponential) 증가할 수가 있다. 그러면 이 문제를 해결하는 데 필요한 시간과 메모리의 양도 역시 기하급수적으로 증가하기 때문에 D_i의 크기를 제한해야 한다. D_i

의 최대 크기를 r이라고 하면 r개의 쌍들이 선택되는 방법에 따라 휴리스틱들의 성능이 달라지게 된다. 본 연구에서는 다음의 두 가지 휴리스틱을 제안하였다. 첫 번째는 구간 분할(interval partitioning)을 이용하는 방법이고, 두 번째는 greedy 스타일의 휴리스틱이다 [10]. 구간 분할은 0/1 knapsack과 같은 문제에 사용되어 우수한 결과를 생성한 휴리스틱이고, greedy 휴리스틱은 여러 NP-hard 문제에 적용되어 왔으며 실행 시간이 매우 빠르면서 결과가 좋은 휴리스틱으로 알려져 있다 [10].

3.1 구간별 선택 (휴리스틱 I)

D_i에 속한 (p, n) 쌍 중에서 최대 이익이 P라고 하면 구간 0과 P 사이를 r개의 등구간으로 나눈 뒤 각 구간에서 이익이 최대가 되는 쌍을 하나씩 선택하는 방법이다. 예를 들어 r = 3이고 D_i = {(3, 10), (7, 13), (9, 15), (13, 18), (14, 20)}이라고 하면 P = 14이므로 세 구간은 (0, 4.66], (4.66, 9.33], (9.33, 14]가 되고 첫 번째 구간에서 (3, 10)이 선택되고, 두 번째 구간에서 (7, 13)과 (9, 15) 중 이익이 더 큰 (9, 15)가 선택되고, 세 번째 구간에서 (13, 18)과 (14, 20) 중 (14, 20)이 선택되어, 이 휴리스틱을 적용했을 때 선택되는 쌍들은 {(3, 10), (9, 15), (14, 20)}이 된다.

만약에 어떤 구간에 속한 쌍이 하나도 없는 경우에는 선택되지 않은 쌍중에서 이익이 최대인 쌍을 선택한다. 예를 들어 D_i = {(20, 10), (29, 15), (30, 16), (32, 20)}이면 P = 32가 되므로 세 구간은 (0, 10.66], (10.66, 21.33], (21.33, 32]가 되고 첫 번째 구간에는 속해 있는 쌍이 하나도 없으므로 아무 것도 선택되지 않고, 두 번째 구간에서 (20, 10)이 선택되고, 세 번째 구간에서 (32, 20)이 선택된다. 그런데 첫 번째 구간에서 아무 것도 선택되지 않았기 때문에 한 개가 더 선택될 수 있고, 선택되지 않은 쌍 중에서 이익이 제일 많은 (30, 16)이 선택되서 최종 결과는 {(20, 10), (30, 16), (32, 20)}이 된다.

〈그림 2〉에 각 노드 i에 대해 D_i를 구하는 알고리즘이 나타나 있다. 구간별 선택 휴리스틱을 구현하기 위해서는 〈그림 2〉의 NSP 휴리스틱에서 SelectPairs(D_i) 함수를 〈그림 3〉과 같이 작성하면 된다. 여기서 Sort(D_i)는 D_i 안에 있는 쌍들을 이익에 대하여 오름차순으로 정렬하고 이 때 지배되는 쌍들을 모두 제거하는 함수이다.

```
NSP_Heuristics(i : node)
{
    if (n_i ≤ c_i)
        if (node i is a leaf)
            D_i = {(p_i, n_i)};
        else {
            let i_1, i_2, ……, i_k be the children of node i;
            for (j = 1 ; j <= k; j++) NSP_Heuristics(i_j);
            D_i = {(p_i, n_i)} ∪ {(p+p_i, n+n_i) |
                  (p, n) ∈ F((Di_1, Di_2, ……, Di_k, c_i-n_i) };
            D_i = SelectPairs(D_i)
        }
    else D_i = {};
}
```

그림 2. NSP 휴리스틱
Fig 2. NSP Heuristic

3.2 최대 이익 순으로 선택 (휴리스틱 G)

D_i에 속한 (p, n) 쌍 중에서 이익이 큰 순으로 r개를 선택하는 방법이다. 예를 들어 r = 3이고 D_i = {(3, 10), (7, 13), (9, 15), (13, 18), (14, 20)}이라고 하면 이 휴리스틱을 적용했을 때 선택되는 쌍들은 {(9, 15), (13, 18), (14, 20)}이 된다.

이 휴리스틱을 구현하기 위해서는 〈그림 2〉의 NSP 휴리스틱에서 SelectPairs(D_i) 함수를 〈그림 4〉와 같이 작성하면 된다.

```
SelectPairs (D_i)
{
    D_i' = Sort(D_i);
    let D_i' = {(p_1', n_1'), (p_2', n_2'), ..... , (p_t', n_t')}
    Δ = p_t' / r;
    D_i, = {};
    left = r;
    for (k = 1; k <= r; k++)
      if (the number of unexamined pairs
            in D_i' <= left) {
          move the unexamined pairs in D_i' to D_i;
          break;
      }
      else {
          find the max profit pair (p_j', n_j')
               in the interval ((k-1)*Δ , k*Δ];
          D_i = D_i ∪ {(p_j', n_j')};
          left--;
      }
    return ( D_i );
}
```

그림 3. 휴리스틱 I를 위한 SelectPairs(D_i)
Fig 3. SelectPairs(D_i) for Heuristic I

```
SelectPairs (Di)
{
    Di′ = Sort(Di);
    let Di′ = {p1′, n1′), (p2′, n2′), ..... , (pt′, nt′)}
    Di, = {};
    for (k = t-r+1; k <= t; k++)
        Di = Di ∪ {(pk′, nk′)};
    return ( Di );
}
```

그림 4. 휴리스틱 G를 위한 SelectPairs(D_i)
Fig 4. SelectPairs(D_i) for Heuristic G

IV. 실험 및 결과

제안된 휴리스틱들은 비쥬얼 C++ 6.0 언어를 사용해서 Intel(R) Core(TM) 2 CPU를 가진 PC에서 구현되고 실험되었다.

테스트 데이터에서 네트워크는 노드 수(n)가 40, 121, 364, 1093인 트리를 사용하였다. 또 각 트리마다 이익, 잡음, 임계값을 고려하여 여덟 종류로 나누고 각 종류마다 열 개의 데이터를 생성하였다. 그러므로 각 트리마다 총 80개의 데이터를 생성하여 실험하였다. 이익은 0과 1,000사이와 0과 10,000사이에서 임의(random)로 생성되었고, 잡음은 0과 1000(이하 m)사이에서 임으로 생성되었다. 임계값은 첫 번째 경우는 모든 노드가 상수인 $m*n/4$과 $m*n/2$ 두 종류를 사용하였고, 두 번째 경우에는 트리에서 노드의 레벨(level)을 고려한 값을 사용하였는데, 트리의 높이를 h, 노드의 레벨을 i라 하면 $m*2^{(h-i)}$와 $m*2^{(h-i)}/2$ 두 종류를 사용하였다.

휴리스틱의 효율성에 가장 영향을 주는 요소는 D_i의 크기(r)이다. r값의 변화에 따른 휴리스틱들의 결과를 비교하기 위하여 각 트리마다 다섯 개의 값을 사용하였는데, 초기 값을 트리 크기의 약 십분의 일을 사용하고 그 다음부터는 2배씩 증가 시켰다, 즉 초기 값을 d라고 하면 d, 2d, 4d, 8d, 16d의 다섯 개 값에 대해 실험하였다. 노드 수 (40, 121, 354, 1093) 각각에 대해 사용된 d값은 (5, 10, 50, 100)이다.

각 데이터에 대해 휴리스틱 I와 휴리스틱 G의 승패를 비교한 결과가 〈표 1〉에 나타나 있다. 〈표 1〉에서 (a, b, c)는 80개의 데이터 중 a는 휴리스틱 I가 휴리스틱 G보다 근사해가 더 우수했던 경우의 수이고, b는 휴리스틱 I와 휴리스틱 G가 근사해가 같은 경우의 수이고, c는 휴리스틱 G가 휴리스틱 I보다 근사해가 더 우수했던 경우의 수를 나타내고 있다. 〈표 1〉에 나타난 것 같이 모든 n에 대하여 휴리스틱 I가 휴리스틱 G에 비해 근사해가 우수한 데이터의 수가 더 많았으며 n이 커질수록 휴리스틱 I가 이긴 데이터의 수가 증가했다.

표 1. 휴리스틱 I와 휴리스틱 G의 승패 비교
Table 1. Number of wins, ties and loses between heuristic I and heuristic G

	n=40	n=121	n=364	n=1093
r=d	(63, 4, 13)	(75, 0, 5)	(79, 0, 1)	(80, 0, 0)
r=2d	(48, 18,14)	(68, 4, 8)	(68, 4, 8)	(79, 0, 1)
r=4d	(22, 36,22)	(54, 15,11)	(55, 19, 6)	(64, 10, 6)
r=8d	(3, 70, 7)	(40, 28,12)	(43, 33, 4)	(55, 19, 6)
r=16d	(0, 80, 0)	(17, 44,19)	(33, 40, 7)	(40, 38, 2)
합계	136,208,56	254,91,55	278,96,26	318,67,15

〈표 2〉에는 휴리스틱 G의 근사해를 1로 했을 때 휴리스틱 I의 근사해와의 비율이 나타나 있다. 〈표 1〉에서와 비슷한 결과를 보이고 있으며 평균적으로 휴리스틱 I가 휴리스틱 G보다 1.22배 더 좋은 결과를 산출하였다. 또 노드의 수(n)이 증가함에 따라 휴리스틱 I가 더 우수한 결과를 나타내고 있고, r이 작을수록 휴리스틱 I가 더 우수한 결과를 나타내고 있다.

〈표 3〉에는 휴리스틱 G의 실행시간을 1로 했을 때 휴리스틱 I의 실행시간과의 비율이 나타나 있다. 한 경우만 제외하고 휴리스틱 I의 실행시간이 휴리스틱 G의 실행시간 보다 더 걸리는 것으로 나타났으며 평균적으로 약 1.68배가 더 걸리는 것으로 나타났다. 또 노드의 수가 증가할수록 휴리스틱 I의 실행시간이 휴리스틱 G의 실행시간 보다 더 걸리는 것으로 나타났다.

표 2. 휴리스틱 G에 대한 휴리스틱 I의 근사해 비교
Table 2. Comparison of the near optimal solutions of heuristic I relative to the near optimal solutions of heuristic G

	n=40	n=121	n=364	n=1093
r=d	1.119	1.377	1.426	1.592
r=2d	1.105	1.224	1.419	1.481
r=4d	1.014	1.177	1.219	1.431
r=8d	1.001	1.115	1.160	1.415
r=16d	1.000	1.005	1.041	1.186
평 균	1.044	1.180	1.253	1.421

표 3. 휴리스틱 G에 대한 휴리스틱 I의 실행시간 비교
Table 3. Comparison of the execution times of heuristic I relative to the execution times of heuristic G

	n=40	n=121	n=364	n=1093
r=d	1.142	1.334	2.003	2.557
r=2d	1.293	1.277	1.804	2.711
r=4d	1.180	1.377	1.484	2.997
r=8d	0.986	1.591	1.814	2.342
r=16d	1.003	1.168	1.850	1.693
평 균	1.121	1.350	1.791	2.460

〈표 4〉와 〈표 5〉에는 r의 값이 증가함에 따라 휴리스틱 I의 근사해와 실행시간이 어떻게 변하는 지를 나타내고 있다. 즉 r=d일 때의 근사해와 실행시간을 1이라하고 r의 값을 2배씩 증가시키면서 그 근사해와 실행시간을 측정하였다. 표에서 보는 바와 같이 r이 2배씩 증가할 때 마다 근사해는 평균 1.01배 좋아졌지만 실행시간은 평균 3.69배 증가하였다. 〈표 6〉과 〈표 7〉에는 휴리스틱 G에 대해 근사해와 실행시간이 어떻게 변하는 지를 나타내고 있다. 표에서 보는 바와 같이 r이 2배씩 증가할 때 마다 근사해는 평균 1.08배 좋아졌지만 실행시간은 평균 3.82배 증가하였다.

표 4. r 값에 따른 휴리스틱 I의 근사해 비교
Table 4. Comparison of the near optimal solutions of heuristic I on various r values

	n=40	n=121	n=364	n=1093	평균
r=d	1.000	1.000	1.000	1.000	1.000
r=2d	1.037	1.036	1.006	1.004	1.021
r=4d	1.058	1.055	1.009	1.006	1.032
r=8d	1.067	1.059	1.010	1.006	1.036
r=16d	1.068	1.062	1.010	1.007	1.037

표 5. r 값에 따른 휴리스틱 I의 실행시간 비교
Table 5. Comparison of the execution time of heuristic I on various r values

	n=40	n=121	n=364	n=1093	평균
r=d	1.000	1.000	1.000	1.000	1.000
r=2d	1.789	3.633	4.567	5.815	3.951
r=4d	4.223	12.285	19.874	27.403	15.946
r=8d	9.111	29.847	55.568	118.562	53.272
r=16d	11.152	84.840	134.965	498.974	182.48

n = 40, 121, 364인 경우에는 그 크기가 작기 때문에 최적해를 구할 수 있지만 n이 더 커지면 시간이 많이 걸리기 때문에 최적해를 구할 수가 없다. 〈그림 5〉에는 n이 364인 경우에 대해 최적해를 1이라고 했을 때 최적해와 휴리스틱 I와 휴리스틱 G의 근사해와의 상대적 비율이 나타나 있다. 〈그림 5〉에서 보는 바와 같이 모든 r 값에 대하여 휴리스틱 I의 근사해는 최적해에 98%이상 근접하였다. 〈그림 6〉에는 n이 364인 경우에 대해 최적해를 구하는 데 필요한 시간을 1이라고 했을 때 휴리스틱 I와 휴리스틱 G의 근사해를 구하는 데 필요한 실행시간과의 상대적 비율이 나타나 있다. 휴리스틱 I의 경우 r = d(50)인 경우에 실행시간이 최적 알고리즘의 실행시간의 약 0.04배 이었고, r = 16d(800)인 경우에는 약 0.61배였다.

표 6. r 값에 따른 휴리스틱 G의 근사해 비교
Table 6. Comparison of the near optimal solutions of heuristic G on various r values

	n=40	n=121	n=364	n=1093	평균
r=d	1.000	1.000	1.000	1.000	1.000
r=2d	1.063	1.154	1.011	1.078	1.077
r=4d	1.177	1.238	1.184	1.119	1.180
r=8d	1.201	1.316	1.274	1.136	1.232
r=16d	1.203	1.463	1.434	1.397	1.374

표 7. r 값에 따른 휴리스틱 G의 실행시간 비교
Table 7. Comparison of the execution times of heuristic G on various r values

	n=40	n=121	n=364	n=1093	평균
r=d	1.000	1.000	1.000	1.000	1.000
r=2d	1.546	3.720	4.743	5.391	3.850
r=4d	4.216	11.131	25.346	19.275	14.992
r=8d	10.869	23.731	52.027	103.350	47.494
r=16d	12.842	94.715	121.242	596.909	206.427

실험 결과에 의하면 가입자의 수(n)가 커질수록 휴리스틱 I의 근사해가 휴리스틱 G의 근사해보다 더 우수하였고, 또 r이 작을 수록 휴리스틱 I의 근사해가 휴리스틱 G의 근사해보다 더 우수하였다. 그러나 실행시간을 보면 휴리스틱 I가 휴리스틱 G보다 더 많이 소요되는 것으로 나타났다. 일반적으로 케이블 TV 망의 가입자 수는 많고 휴리스틱 I는 휴리스틱 G에 비해 시간은 조금 더 걸리지만 작은 r값에도 우수한 결과

를 보이므로 휴리스틱 I을 사용하는 것이 더 유리한 것으로 판단된다.

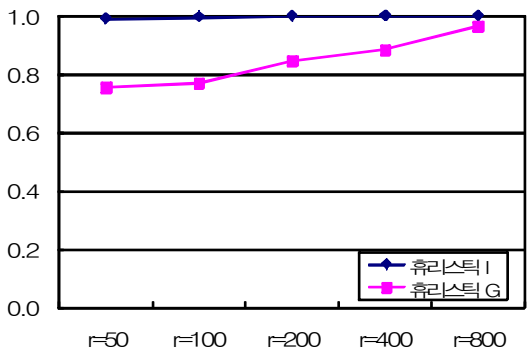

그림 5. n = 364일 경우 최적해와 근사해 비교
Fig 5. Comparison of near optimal solutions with the optimal solution for n = 364

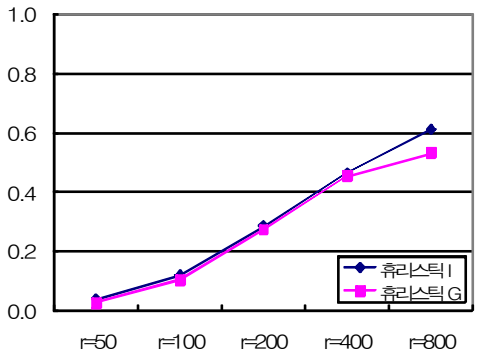

그림 6. n = 364일 경우 최적 알고리즘과의 실행시간 비교
Fig 6. Comparison of the execution times with optimal algorithms for n = 364

V. 결론

케이블 TV 망은 일반적으로 분배센터에서 가입자에게 방송 신호를 내려 보내는 데, 최근에는 사용자로부터 분배센터로 가는 상향 채널을 인터넷과 전화와 같은 광대역 서비스로 확장하여 그 효용성을 증가시키고 있다. 상향채널은 잡음에 매우 취약한데 잡음이 어떤 임계값을 넘게 되면 신호가 잡음과 구별되지 않게 되므로 이런 노드들은 네트워크에서 분리할 필요가 생기게 된다.

본 연구에서는 HFC 기술을 사용하는 케이블 TV 망에서, 선택된 모든 노드들의 잡음의 합이 각 노드들에 주어진 임계값을 넘지 않으면서 이익의 합이 최대가 되도록 노드들을 선택하는 휴리스틱을 개발하고 실험을 통해 비교 분석하였다. 본 연구에서 개발된 휴리스틱들은 HFC 네트워크 운영 시스템이 장착되어 잡음이 많은 노드들을 탐지하고 자동으로 분리하는 데 활용할 수 있으며, 우수고객들에게 더 좋은 질의 서비스를 제공하는 데 사용할 수 있다.

참고문헌

[1] Gary Donaldson and Doug Jones, "Cable Television Broadband Network Architecture," IEEE Communications Magazine, June, 2001.

[2] Michael R. Garey and David S. Johnson, "Computers and Intractability : A Guide to the Theory of NP-Completeness," W.H. Freeman and Company, New York, 1979.

[3] Andrew Paff, "Hybrid Fiber/Coax in the Public Telecommunications Infrastructure," IEEE Communications Magazine, April, 1995.

[4] Dennis Picker, "Design Considerations for a Hybrid Fiber Coax High-Speed Data Acess Network', Proceeding of COMPCON, IEEE, 1996.

[5] Srinivas Ramanathan and Riccardo Gusella, "Toward Management Systems for Emerging Hybrid Fiber-coax Access Networks," IEEE Network, September 1995.

[6] Gerry White, "Network Management Issues in Internet Works Based on Hybrid Fiber-Coax Cable Plants," Proceedings if the Second International Workshop on Integrated Multimedia Services to the Home, IEEE 1995.

[7] Jang Ho Lee, Changwoo Pyo, Kyun-Rak Chong, and Jun-Yong Lee, "An Optimal Algorithm for Maximal Connectivity of HFC Network," LNCS 2662, pp 868-875, 2003.

[8] H. Kobayashi, K. Hirai, and H. Ibe, "Cable Network System with Ingress Noise Suppressing Function," US Patent 6530087, 2003.

[9] Ed Pivonka, "Silence Noise on Your Network," Electronic Construction and Maintenance Vol. 104, No. 1, p18-21, January 2005.

〔10〕 E. Horowitz, S. Sahni. S. Rajasekaran, Computer Algorithms C++, Computer Science Press, 1996.

저 자 소 개

정 균 락

1980년 2월 : 한국과학기술원 전자계산학석사

1991년 2월 : 미네소타대학교 컴퓨터공학박사

1991년 ~ 현재 : 홍익대학교 컴퓨터공학과 교수

관심분야 : 알고리즘 설계 및 분석, VLSI 알고리즘