論文2000-37SD-5-10

효과적인 OTC 채널 라우터의 구현을 위한 최적화 기법의 성능 분석에 관한 연구

(A Study on Performance Analysis of Optimization Techniques for Efficient OTC(Over-The-Cell) Channel Router)

張 承 圭 *, 朴 在 興 *, 張 勳 *

(Seung Kew Jang, Jae Heung Park, and Hoon Chang)

요 약

지금까지 제시된 채널 라우팅 알고리즘의 복잡도는 공정기술의 발전에 따라 많은 어려운 부분이 해결되었지만 아직도 기존의 라우팅 알고리즘 수준에 머물러 있었다. 본 논문에서는 이러한 추세에 따라 개선된 3-레이어 OTC(Over-The-Cell) 채널 라우팅 알고리즘을 제시한다. 제안한 알고리즘은 채널 라우팅 문제를 일반적인 문제로 단순화시켜, 최적에 가까운 해에 수렴하기 위해서 시뮬레이티드 어닐링 기법을 이용한다. 그리고, 채널 라우팅에서 있어서 가장 어려운 요소로 알려진 cyclic vertical constraint를 제거하기 위한 방안을 제시하고, 알고리즘의 수행중 지역 해에 빠졌음을 감지하여 탈출을 보장하는 방안을 제시한다. 또한, 유전자 알고리즘을 이용하여 구현된 채널 라우터와 비교한다. 제안한 채널 라우팅 알고리즘은 리눅스 시스템에서 C++과 모티프를 이용하여 구현되었으며, 예제 회로에 적용하여 성능을 비교함으로써 제시한 알고리즘의 효율성을 찾을 수 있었다.

Abstract:

In this paper, we propose a Over-The-Cell channel routing algorithm for the advanced three-layer process. The proposed algorithm makes the channel routing problem to simplified one and makes use of simulated annealing technique to achieve the global optimal solution. And, a new method to remove the cyclic vertical constraints which are known to be the hardest element in the channel routing problem is proposed, and a way to detect the local minimal solution and escape from it successfully is presented. Futhermore, genetic algorithm based channel router is implemented and comparison is performed with the simulated annealing based one. All algorithms are written in C++ and GUI is made using Motif under Linux environment.

* 正會員, 崇實大學校 컴퓨터學科

(Dept. of Computing, Graduate School, SoongSil Univ.)

※ 이 논문은 1998년 한국 학술 진흥 재단의 학술연구비(과제번호 : 1998-016-E00054)에 의하여 지원되었음.

接受日字:1998年12月29日, 수정완료일:2000年4月25日

I 서 론

채널 라우팅 문제는 VLSI 기술과 더불어 오랫동안 연구되어 온 분야이지만, 대부분 2-레이어에 국한되어 왔고 라우팅의 해 역시 이론적으로 얻어진 최소 채널의 높이보다 하나 또는 두개 정도 증가하는 라우터들이 대부분이었다[1,10]. 그러나 90 년대부터 각광을 받기

시작한 새로운 표준셀 모델과 프로세스로 인해서 셀 위에서도 라우팅이 가능한 OTC(over the cell) 라우터가 등장하게 되었다.

지금까지의 대부분의 OTC 라우터들은[1,2] 셀 위에서 비아(via)를 허용하지 않는다는 가정하에 알고리즘이 연구되어 왔지만, 공정기술이 발전됨에 따라서 현재 셀 위에서도 자유로이 레이어를 변경하면서 라우팅이 가능하게 되었다. Holmes는 3-레이어를 사용한 라우팅 알고리즘으로 기존의 채널에 비해 65% 정도 감소하는 결과를 얻었다[2].

특히 라우팅 문제에 있어서 가장 힘든 요소가 CVC(cyclic vertical constraint)이고, 이로 인해 트랙이 증가하는 결과를 얻게 되지만 OTC 라우팅은 가능성 있는 네트를 셀 위로 올림으로써 효과적으로 라우팅을 하게 된다[3].

본 논문에서 제시하는 알고리즘은 최신의 기술인 개선된 3-레이어 공정을 위한 라우팅 알고리즘이기 때문에 기존의 OTC 채널 라우터에서 사용한 셀 위에서의 레이어의 평면성을 고려하지 않는다. 제시한 라우터는 시뮬레이티드 어닐링(SA: Simulated Annealing)과 유전자 알고리즘(GA: Genetic Algorithm)을 이용한 라우터로서 최적에 가까운 해를 얻을 수 있다. 기존에 연구된 많은 라우팅 알고리즘들은 대부분 그래프를 이용한 알고리즘으로 알고리즘이 매우 복잡하고 몇 가지 휴리스틱에 의존함으로써 최적의 해를 보장받을 수 없었다[4]. SA는 지금까지 많이 연구되어온 분야로써 전역 해를 찾기 위해 이용되고 있다. 또한 컴퓨팅 파워의 증가로 인해서 보다 빨리 해를 얻을 수 있게 되었다. 특히 분할(partioning), 배치(placement), 배선(routing) 등의 물리설계(physical design) 분야에서는 SA나 GA를 이용한 연구가 활발히 진행되고 있어 앞으로 이 분야에 많은 발전이 있을 것으로 보인다[5].

본 논문에서는 SA나 GA같은 방식을 이용하여 OTC 를 어떻게 구현할 것이가에 중점을 두었고, 그리고 이 해서 되어야할 것들을 다루었다. 또한 이러한 방식은 시간을 고려하지 않고, 좋은 해를 얻는 것에 보다 많은 의미를 두었다. 제시한 알고리즘의 장점은 라우팅 문제의 정의를 간단히 함으로써 쉽게 OTC 채널 라우터로 확장이 가능하다는 것과 지역해에서 탈출함으로써 빠르게 수렴할 수 있다는 점이다. 그리고, CVC을 제거하는 방법을 제시하였다. 결과적으로 기존의 알고리즘보다 간단히 구현할 수 있다. 또한 기존의 OTC 채널 라우터들은 셀 위에서 라우팅할 네트들을 선택한 뒤 라우팅을 하고 그런 다음 기존의 채널 라우터를 이용해서 채널 라우팅을 마치게 된다. 하지만 본 논문에서 제시하는 알고리즘은 채널과 셀의 라우팅 문제 자체를 하나로 취급하므로 기존의 라우터는 필요하지 않다. 즉, 셀 위의 영역 역시 채널과 동시에 취급한다.

1. 사용된 표준셀 모델

OTC 채널 라우터는 표준셀의 라우팅 모델에 따라 알고리즘이 달라지게 된다[2]. 본 논문에서 제시한 알고리즘이 다루는 모델은 그림 1과 같은 HCVD-BTM (Horizontally Connected Vertically Divided-Boundary Terminal Model)이고 셀 위에서 비아를 허용하는 개선된 3-레이어 공정을 위한 라우팅 알고리즘이다. 일반적으로 산업계에서 가장 널리 사용되어 기술이 축적된 표준셀이 BTM이므로 쉽게 OTC 모델로 바꿀 수 있다. 그리고 기존의 대부분의 OTC 라우터는 OTC 위에서 비아가 허용이 되지 않았기 때문에 여기에 초점을 맞추어 셀 위에서의 평면 라우팅을 주로 다루어 왔지만[4], 현재 기술 수준이 충분히 개발되어 이미 OTC 위에서 여러 레이어와 비아를 허용하는 개선된 공정 기술을 지원하고 있다.

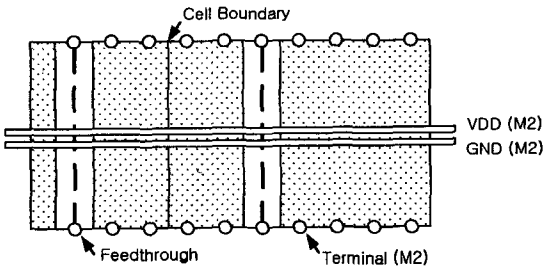

그림 1. HCVD-BTM
Fig. 1. HCVD-BTM.

II 시뮬레이티드 어닐링을 이용한 채널 라우터

1. 문제의 정의

라우팅 문제를 구성하는 기본요소는 채널의 길이, 상위(하위) 터미널 리스트 및 레이어의 개수이다. 제시한 알고리즘에서는 n-터미널 네트의 문제가 주어지면 이를 그림 2와 같이 2-터미널의 집합으로 바꾼다. 그러면, n-터미널 네트의 배선은 다소 쉬운 문제로 바뀌게 된

다. 모든 n-터미널 네트가 2-터미널 네트로 바뀌고 CVC가 제거되었다면 임의의 네트는 트렁크(trunk)라 불리는 하나의 수평 세그먼트와 두개의 수직 세그먼트를 가지게 된다. 수직 세그먼트의 열(column) 위치는 고정되고 그 길이만 변하게 되고, 수평 세그먼트는 길이는 고정되지만 트랙의 위치만 채널 안에서 수직 방향으로 움직일 수 있게 된다. 이를 수식으로 나타내면 식 1과 같다. 주어진 문제는 모든 네트들로 구성되는 네트 집합 N, 수직 세그먼트 집합 V와 수평 세그먼트 집합 H로 표현된다. 하나의 네트 N_i는 수직 세그먼트 vi와 수평 세그먼트 h_i로 구성되고, 하나의 수직 세그먼트 원소 v_i는 좌, 우 두개의 브랜치 $vleft,i$와 $vright,i$로 표현된다.

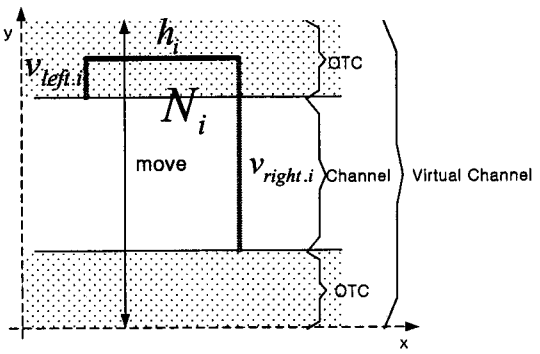

그림 2. 라우팅 문제
Fig. 2. Routing problem.

$$N=\{N_1, N_2, \ldots, N_n\},\quad N_i=\{v_i, h_i\}$$
$$V=\{v_1, v_2, \ldots, v_n\},\quad v_i=\{v_{left,i}, v_{right,i}\}$$
$$H=\{h_1, h_2, \ldots, h_n\} \tag{1}$$

$y(h_i)$는 h_i의 y 좌표의 값이고 hchannel은 채널의 높이이며 hOTC는 셀 위의 라우팅 영역이다. h_i는 채널 영역과 OTC 영역 내에 존재하므로 y(hi)는 식 2와 같이 주어진다.

$$0 \le y(h_i) \le h_{channel} + 2 \cdot h_{OTC} \tag{2}$$

N으로부터 V와 H를 얻을 수 있고, 만약 하나의 네트의 트랙이 변경되었을 경우, N을 변경한 뒤 V와 H를 같이 변경해야만 하다. 그리고 중첩(overlap) 비용을 계산할 때 N에서 바로 구하지 않고 V와 H를 사용하여 구하여 SA에서 평가함수의 값을 결정하는데 사용된다. 이렇게 네트의 정의를 간단히 하면 OTC 문제도 채널의 높이를 셀 위까지 확장함으로써 문제를 해결할 수 있다. 이처럼 확장된 채널을 가상 채널(virtual channel)이라고 정의한다[1].

2. 채널 라우팅 알고리즘

본 논문에서 제시하는 알고리즘은 전체적으로 그림 3과 같은 구조를 갖는다

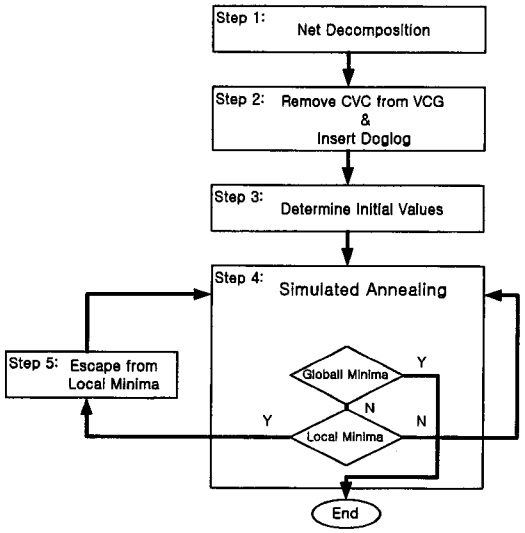

그림 3. 전체적인 알고리즘
Fig. 3. Routing algorithm.

1) 단계 1: 다중-터미널을 2-터미널로 분할

n-터미널 네트를 2-터미널 네트로 바꿈으로서 문제를 간단히 한다. 네트 분할 방법은 여러 가지가 존재할 수 있지만, 여기서는 왼쪽의 터미널부터 순서대로 두개씩 묶어 새로운 2-터미널을 생성한다. 따라서 n-터미널 네트는 (n-1)개의 2-터미널 네트로 분할된다. 이렇게 함으로써 터미널이 있는 열에서만 삽입되는 제한 도그레그(restricted dogleg)를 사용할 수 있게 된다[7,8]. 그림 4는 제한 도그레그를 사용한 예를 보여준다. 네트 5에 대하여 네트 3 터미널이 있는 열에 제한 도그레그를 삽입함으로써 채널의 높이가 감소하였다.

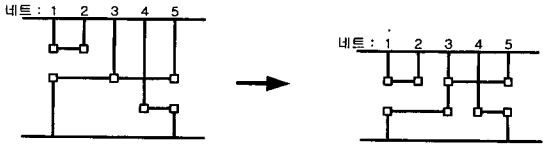

그림 4. 제한 도그레그를 사용한 예
Fig. 4. An example using restricted dogleg.

2) 단계 2: 제안된 CVC의 제거 알고리즘

2-터미널 네트로 바뀐 상태에서 LOB(locally optimal breaking)을 시도한다[3]. 이 과정을 거쳐야만 하는 이유는 처음에 주어진 문제에 CVC(cyclic vertical constraint)가 존재하면 곧바로 SA를 적용하기 힘들고, 여러 복잡한 휴리스틱을 적용해만 하기 때문이다. 그리고, 그래프에서 CVC를 제거했다면 최종해에는 오버랩이 없음을 알 수 있다.

본 알고리즘에서는 VCG(Vertical Constraint Graph)를 형성한 후에 CVC를 찾아 이를 없앨 수 있도록 하나의 네트를 두개의 네트로 분리하는 과정을 거치게 된다. 이 과정에서 터미널이 없는 열에서도 삽입될 수 있는 비제한 도그레그(unrestricted dogleg)가 필연적으로 삽입되고, 도그레그의 위치는 CVC가 전혀 발생하지 않도록 결정되어야 한다.

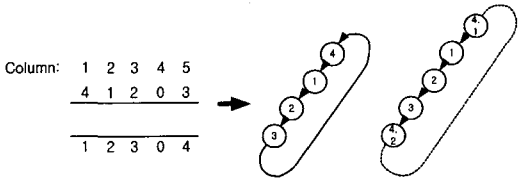

그림 5. CVC에서 네트의 선택과 도그레그의 위치선택
Fig. 5. Selection of net and dogleg position in CVC.

그림 5는 CVC가 존재하는 문제에 대해서 CVC를 제거한 VCG를 보여준다. 이 예에서는 CVC를 제거하기 위해 네트 4를 분리하였다. 그림 6은 이에 따라 네트 4에 비제한 도그레그 삽입된 경우를 보여준다.

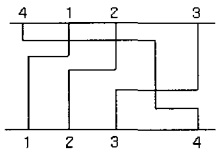

그림 6. CVC를 제거한 예
Fig. 6. An example removing CVC.

그림 7은 [9]에서 사용된 예로써 제시한 알고리즘의 동작을 확인하기 위해 보였다. 이 그림에서 보면 채널 경계에 터미널 번호가 한개 또는 두개 있는데, 이것은 이미 2-터미널 네트로 바뀌었기 때문이다. 터미널 번호와 열 번호는 편의상 0부터 시작한다. 주어진 문제로부터 그림 8과 같이 VCG를 만든 후에 탐색 알고리즘을 사용하여 CVC를 형성하는 네트 리스트 DC(directed circuits)를 얻을 수 있다. 이 그림에는 두개의 CVC를 형성하는 DC가 존재한다 {{0,3},{9,11}}. DC를 발견하는 효율적인 알고리즘과 이렇게 구한 DC 상의 네트중에서 CVC를 제거할 수 있는 네트를 찾아내는 효율적인 알고리즘이 요구된다.

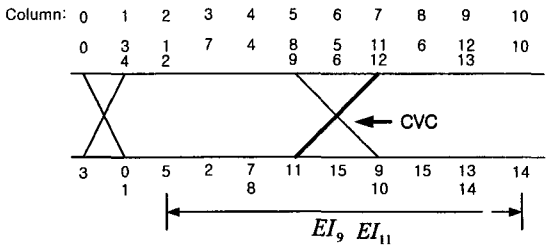

그림 7. CVC를 갖는 채널의 예
Fig. 7. Channel example with CVC.

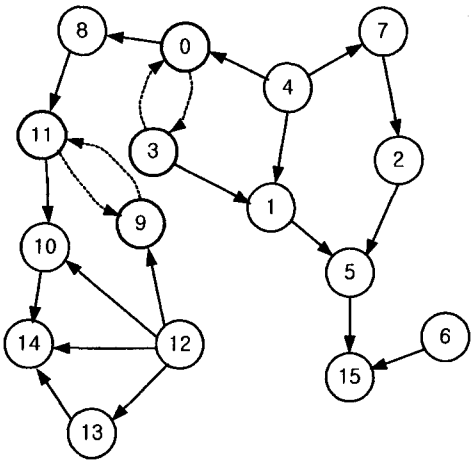

그림 8. 그림 7의 VCG
Fig. 8. VCG of Figure 7.

가) 제안된 DC 탐색 알고리즘

일반적인 그래프에 대해서 DC를 찾는 효과적인 알고리즘은 존재하지 않는다. 이런 알고리즘은 가능한 모든 DC를 찾아야 하기 때문이다. 하지만, 여기서의 VCG에서는 모든 가능한 DC를 찾을 필요는 없다. 하나의 DC가 다른 DC와 겹치는 부분이 있다면, 이중 하나만 찾으면 되므로 상대적으로 간단하다. DC 탐색의 목적은 DC가 전혀 존재하지 않도록 하는 것이기 때문이다. 그림 9는 본 알고리즘의 DC 탐색 알고리즘이다.

각 노드에서 방문 가능한 노드를 저장하기 위해 연결 리스트(linked list)를 이용하여 그래프를 표현하였다. 함수 cvc()에서는 모든 노드에서 함수 travel()을 호출하고 리턴 값은 그 노드에서 시작되는 DC가 있는지를 나타내는 이진 값이다. 라인 22에서 함수 travel()은 각 노드에서 깊이 우선 탐색을 위해 재귀적으로 호출된다. 인자 status는 각 노드가 방문되었는지를 라인 19에서 체크한다. 만약 어떤 노드가 CHECKED_DC로 체크되어 있다면, 이 노드는 DC를 형성하는 노드임을 의미하게 되어 이후에는 결코 방문되지 않는다. 이런 이유로 이 탐색 알고리즘은 기존의 DC 탐색 알고리즘 보다 효율적이다.

```
1   void vcg::cvc(){
2     num_cvc=0;
3     for(int i=0;i<num_nodes;i++){
4       if(nodes[i].next != NULL && nodes[i].found_dc != CEHECKED_DC){
5         if(travel(i,i)) num_cvc++;
6       }
7     }
8   }
9
10  int vcg::travel(int starting_node,int net){
11    if(!nodes[net].next) return false;
12    if(nodes[net].found_dc == CEHECKED_DC) return false;
13
14    if(nodes[net].found_dc == starting_node){
15      if(starting_node == net) return true;
16      return false;
17    }
18
19    nodes[net].found_dc=starting_node;
20    int num_dc=0;
21    for(node* n=nodes[net].next;n!=NULL;n=n->next)
22      dc+=travel(starting_node,n->n);
23
24    if(num_dc){
25      add_dc(num_cvc,net);
26      nodes[net].found_dc=CEHECKED_DC;
27      return true;
28    }
29    else return false;
30  }
```

그림 9. 구현된 DC 탐색 알고리즘
Fig. 9. Implemented DC search algorithm.

나) 도그레그 삽입을 위한 열 선택 알고리즘

구한 각각의 DC에서 네트를 선택하는 문제는 다음과 같은 휴리스틱으로 해결한다. 네트 Ni의 트렁크가 차지하는 구간을 (l_i, r_i)라 할 때, l_i와 r_i를 제외한 $(l_{i-k},\ r_{i-k})$의 구간을 확장된 구간 EI_i라 정의하자. k는 문제에 따라서 바뀔 수 있지만 본 알고리즘의 구현 시 3으로 고정하였다. 채널의 경계에 위치한 터미널 중 어떠한 네트에도 속하지 않기 때문에 전기적인 연결이 필요 없는 터미널을 무용(vacant) 터미널이라고 하고, 특정 열의 상하 모두 무용 터미널이 존재하는 경우 이를 인접 무용(vacant abutment)이다. 이때 다음 2개의 값을 정의할 수 있다.

- $N_{vacant}(i)$: EIi 안에서 무용 터미널의 개수
- $N_{abutment}(i)$: EIi 안에서 인접 무용의 개수

n-터미널 네트가 2-터미널 네트로 분할되었기 때문에 그 네트의 터미널 이외의 터미널에 분할 전의 같은 네트가 존재한다면, 그 터미널은 무용 터미널로 볼 수 있다.

하나의 CVC를 제거하기 위해서는 두개의 트랙이 소요되기 때문에 이러한 네트들은 채널의 밀도를 증가시키고 결국 높이를 증가시킬 소지가 크다. 따라서 이러한 네트들은 가능하면 OTC 영역으로 옮겨야 한다.

식 3은 도그레그 삽입을 위한 열 선택 함수이다. 여기에서 인접 무용과 무용 터미널을 기준으로 삼는 이유는 그림 5에서와 같이 이 부분에서 도그레그가 가장 쉽게 삽입될 수 있기 때문이다. 상용화된 칩들의 경우 일반적으로 무용 터미널과 인접 무용의 개수가 전체 터미널의 50-80%와 30-70% 정도 차지한다[10]. 따라서, 수식 3에 따라 DC에 속한 네트(n)들에 대해 EIi 평가함수를 적용해 그 중에서 최대의 값을 갖는 네트($n*$)를 식 4에 의해 정한다. 구현 시 α_1와 β_1는 각각 1.0과 2.0로 할당하였다. 무용 터미널이 있는 열 보다 인접 무용의 열에서 도그레그의 삽입이 보다 자유롭기 때문에 $\alpha_1 \le \beta_1$ 를 만족해야 한다.

$$Cost_{NS}(n) = \alpha_1 \cdot N_{vacant}(n) + \beta_1 \cdot N_{abutment}(n) \quad (3)$$

$$Cost_{NS}(n^*) = \max_{n \in DC} [Cost_{NS}(n)] \quad (4)$$

이렇게 선택된 네트는 두개의 서브네트로 분할되는데, 그 위치 즉 도그레그의 위치는 EI_n^*에 속한 열에 대해서 국부 채널 밀도(local channel density)의 값과 인접 무용과 무용 터미널의 개수에 따라 식 5와 같이 계산하고 식 6에 따라서 최대값을 갖는 열의 위치(c*)를 정하게 된다.

$$Cost_{SC}(c) = -\alpha_2 \cdot M_{local}(c) + \beta_2 \cdot M_{abutment}(c) + \gamma_2 \cdot M_{vacant}(c) \quad (5)$$

$$Cost_{SC}(c^*) = \max_{c \in EI_{n^*}} [Cost_{SC}(c)] \quad (6)$$

수식 5에서 $M_{local}(c)$ 은 열 c에서의 지역 채널 밀도를, $M_{abutment}(c)$는 인접 무용의 개수를, $M_{vacant}(c)$ 은 무용 터미널의 개수를 각각 나타낸다. $M_{vacant}(c)$ 와

$M_{abutment}(c)$ 는 0 또는 1을 갖게 된다. 구현 시 α_2, β_2 m γ_2는 실험에 따라 1.0, 2.0, 1.0으로 할당하였다. 그림 7의 네트 11에 대해 위의 식을 EI_n에 대해 적용하면 열 9에서 $M_{local}(9)$는 2, $M_{abutment}(9)$는 1, $M_{vacant}(9)$는 0이 되어 $Cost_{sc}(9)$의 값은 0이 되고, 이 때 이 열이 최대값을 갖게 되므로 열 9에 제한된 도그레그가 삽입된다.

지금까지의 결과로부터 CVC를 제거할 네트와 그 네트의 도그레그의 위치를 가지고 네트를 분할하여 새롭게 2-터미널 네트의 문제로 바꾼다.

3) 단계 3: 초기값 결정

이 단계에서는 SA를 적용하기 전에 초기값들을 정하게 된다. 제시한 알고리즘에서 각 네트의 초기 위치는 임의로 배치되고, 채널의 초기 높이는 이론적으로 최소인 값을 갖게 된다. 즉, 가상 채널의 초기 높이는 HCG(Horizontal Constraint Graph)의 최대 클릭(clique) 값이 된다[4].

4) 단계 4: 시뮬레이티드 어닐링 적용

이 단계에서는 SA를 적용하게 된다[11]. 이때 가장 중요한 것은 상태의 생성, 평가함수와 온도 T의 스케줄링이다.

가) 상태 생성 및 평가함수

상태를 생성하는 두개의 연산자(operator)를 정의하였다. 첫 번째는 임의로 하나의 네트를 선택하여 그 네트의 트랙을 제약 조건 안에서 임으로 이동시키는 이동(move) 연산자이고, 두 번째는 임의로 선택된 두개의 네트간의 트랙을 서로 교환하는 교환(swap) 연산자이다. 그리고, 두 연산자 중 선택은 식 7과 같이 확률적으로 선택하도록 하였다. 이때 함수 random(0,1) 은 0과 1사이의 실수를 임의로 생성한다. 실험적으로 이동 연산자의 발생 빈도수가 조금 우세할 때 수렴속도가 빨랐다.

$$\text{if}\,(random(0,1) < 0.6)\quad operator_{move} \\ else\quad operator_{swap} \tag{7}$$

제안한 SA 알고리즘에서는 식 8과 같은 평가함수를 사용하여 평가함수에 맞는 보다 좋은 전역해를 구한다. OL은 네트들의 중첩 빈도 수이고, NL은 네트들의 길이로 수평 세그먼트의 길이는 변함이 없기 때문에 수직 세그먼트의 총 길이의 합이다. 구현 시 α_3과 β_3은 많은 실험을 통하여 각각 0.9와 0.05를 배정하였다.

$$Cost_{SA} = \alpha_3 \cdot OL + \beta_3 \cdot NL \tag{8}$$

나) 스케줄링

일반적인 SA에서는 온도 T의 스케줄링이 새로운 상태의 이동시 상수 배만큼 변하게 된다. 그러나 본 논문에서는 지역해(local solution)를 검사하여 온도를 바꾸는 방법을 추가하였다. 알고리즘에서는 더 이상 비용이 감소하지 않는 지역해에 빠졌을 때 T_i 의 값을 증가시킴으로써 지역해에서 탈출하도록 하였다. 식 9와 같이 고정된 스케줄링이 아닌 동적인 스케줄링을 하게 된다. 구현 시 α_4는 0.97을, γ_4는 0.08을 사용하였을 때 효과적이었다.

$$T_{i+1} = \alpha_4 \cdot T_i \ : \text{지역해에 있지 않을 때} \\ T_{i+1} = T_i + \gamma_4 \ : \text{지역해에 있을 때} \tag{9}$$

5) 단계 5: 지역해의 탈출

제시한 알고리즘의 가장 중요한 부분으로서 SA 알고리즘의 수행 중에 지역해에 빠졌음을 감지하여 빠르게 탈출하게 만든다. 채널 라우팅 문제가 지역 최소해에서 쉽게 빠져 나올 수 있는 이유는 다른 문제와는 달리 주어진 문제 자체를 변화시킬 수 있기 때문이다.

매번 상태의 이동에서 평가함수의 변화를 관찰하다가 일정기간 변화가 없어지면 지역해로 판단을 내리게 되고 여기에서의 탈출을 위해 먼저 온도 T를 증가시켜본다. 이렇게 함으로서 지역해로부터 빠져 나올 가능성이 커진다. 그리고 일정기간 기다려 보다가 평가함수에 아무런 변화가 관찰되지 않으면 깊은 지역해로 판단하고, 이번에는 채널의 높이를 하나 증가시켜 탈출을 시도한다. 이렇게 두 계로 지역해를 구분하여 대처하도록 함으로써 새로운 채널의 삽입이 쉽게 이루어지지 않도록 하였다. 주어진 문제에는 CVC가 존재하지 않기

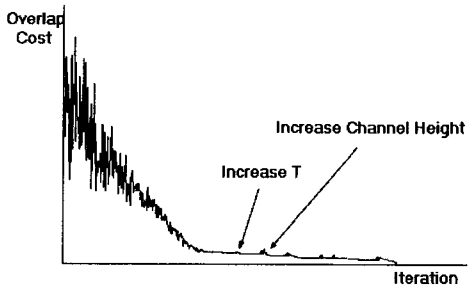

그림 10. 지역해의 탈출
Fig. 10. Escape of local minimum.

때문에 새로운 트랙의 삽입을 통하여 전역해로의 수렴을 보장할 수 있다. 그림 10은 지역해에서 탈출하기 위해서 시도되는 두 가지 방법을 보여준다.

3. 구현

```
1void simulated_annealing(){
 2  float T=100.0;
 3  float next_cost, current_cost, real_cost,
 4  float prev_costs[20], last_cost;
 5  bool check_minimum=1;
 6  int count=0, local_count=0;
 7
 8  for(int i=0;i<20;i++) prev_costs[i]=(float)i;
 9
10  initialize_cost();
11  for(int i=0;i<OUTER_LOOP;i++){
12     for(int k=0;k<INNER_LOOP;k++){
13        current_cost=tracks.overlap_cost()+columns.overlap_cost();
14        real_cost=0.9*current_cost+0.05*columns.length_cost();
15
16        if(rand_01() < OP){ generate(); op=0; }
17        else { generate2(); op=1; }
18
19        next_cost=0.9*cost_overlap()+0.05*cost_length();
20
21        if(!accept(next_cost,real_cost,T)){
22           if(!op) recover();
23           else recover2();
24           cost_overlap(); cost_length();
25        }
26     }
27     T=0.97*T;
28
29     if(current_cost == 0) break;
30     prev_costs[i%20]=current_cost;
31     if(check_minimum && is_localminimum(prev_costs)){
32        T+=0.08;
33        last_cost=current_cost;
34        check_minimum=0;
35     }
36     if(!check_minimum) count++;
37     if(count == 20){
38        if(last_cost == current_cost){
39           if(local_count++ == MAX_LOCAL){
40              cerr << "Too Many Local minimum₩n";
41              break;
42           }
43           tracks.incnum();
44           columns.inc_trunks();
45           check_minimum=0;
46           count=0;
47        }
48        else
49           check_minimum=1; count=0;
50     }
51  }
52}
```

그림 11. 구현된 SA 함수
Fig. 11. Implemented SA function.

그림 11은 실제로 구현된 SA 함수 부분을 보여준다. 먼저, 필요한 변수들을 정의하고 SA를 적용한다. 주목할 만한 점은 평가함수의 계산에서 상당히 많은 계산 시간이 소요되므로, 상태의 변화 시 변화하는 곳의 비용만을 계산하여 이전의 비용에 변화를 가하는 것이다. 마지막으로, 지역해에 빠졌는지를 검사하여 온도 T를 증가시키거나 채널의 높이를 증가시켜 탈출하게 된다.

이렇게 해서 얻어진 최종의 해는 레이어 2와 3으로 이루어지며, 프로세스와 전기적 특성을 고려해 볼 때 레이어 1을 사용할 수 있으면 향상된 특성을 얻게 된다. 따라서, 채널 부분에서의 라우팅은 그림 12와 같이 가능하다면 레이어 3을 레이어 1로 바꾸어 준다.

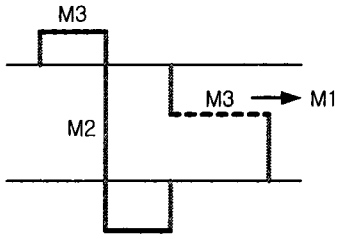

그림 12. 채널 영역의 레이어 변경
Fig. 12. Layer change in channel region.

Ⅲ 유전자 알고리즘을 이용한 채널 라우터

SA외에 최적 해를 찾기 위한 또 다른 방법으로 GA이 있다. SA는 하나의 상태를 변화해 가면서 해에 근접해 가는 Monte Carlo 기법인 반면, GA는 가능한 여러 상태들로부터 출발하여, 좋은 해에 대해서 이들 간의 규칙성에 근거하여 보다 낳은 해로 발전하는 방법으로 진화의 원리에 토대를 두고 있다.

본 연구에서는 OTC 채널 라우터를 앞의 SA 뿐만 아니라 GA를 이용해 구현해 보았다.

1. 구현 및 성능 분석

그림 13은 OTC 채널 라우팅을 구현한 GA의 주요 부분이다. 그리고, 그림 14와 같이 정수 배열을 사용하여 하나의 라우팅 해를 염색체로 표현하였다.

그리고, 각각의 개체에 대한 적합도는 식 10과 같이 계산하였으며, 다음 세대로 넘어갈 개체를 선별하는 선택 과정(selection process)은 roulette wheel 방법을 이용하였다[12]. 또한 실험적으로, 3점 교배를 사용하였고, 0.03 정도의 확률로 변이를 발생시켰다.

$$f=\frac{1}{Cost_{SA}} \tag{10}$$

그림 15는 쉬운 문제에 대해서 적용했을 때의 비용의 변화과정이다. 비용의 변화가 일정기간 없어지면 채널 증가 후 다시 시작하도록 하였기 때문에 해에 도달할 때까지 여러 개의 그래프가 존재하는 것이다.

대부분의 이보다 어려운 문제에 대해서 GA만으로는

전역해에 수렴하지 못하였다. GA는 교배와 변이가 적절히 수행될 때 그 효과를 얻을 수 있지만, 채널 라우팅 문제에서는 교배가 거의 이루어지지 않았다. 그 결과 해의 수렴이 어려웠고 만족할만한 결과를 얻지 못했다.

GA의 특징은 전역해의 인접한 구역까지는 접근하지만 최종 목적지(goal)까지의 도달은 상당히 어려운 것으로 알려져 있다. 따라서 GA는 문제 해결의 전 단계로 사용되는 경우가 많다고 보고되었으며, 이 실험을 통해서 파악할 수 있었다[12,13].

```
1 void genetic_algorithm(){
2   float cx;
3   float prev_costs[LOCAL_LEN];
4   bool check_minima=true;
5   int count=0, num_local=0;
6
7   for(int i=0;i<LOCAL_LEN;i++)
8     prev_costs[i]=(float)i;
9    evalute();
10  keep_the_best();
11
12  for(int i=0;i<MAX_LOOP;i++){
13      selection();
14      crossover();
15      mutate();
16      mutate2();
17      evalute();
18      elitist();
19
20      cx=data.cx;
21    prev_costs[i%LOCAL_LEN]=cx;
22
23    if(cx == 0.0) break;
24        if(num_local > MAX_LOCAL) continue;
25
26        if(check_minima &&is_localminima(prev_costs)){
27         increase_track();
28         evalute();
29         keep_the_best();
30         i=-1;
31         num_local++;
32         check_minima=false;
33      }
34    if(!check_minima) count++;
35    if(count == LOCAL_LEN){
37        check_minima=true; count=0;
38      }
39  }
40 }
```

그림 13. 구현된 GA 함수
Fig. 13. Implemented GA function.

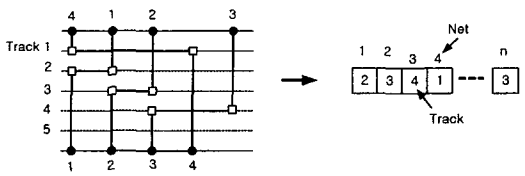

그림 14. 염색체 표현
Fig. 14. Genotype representation.

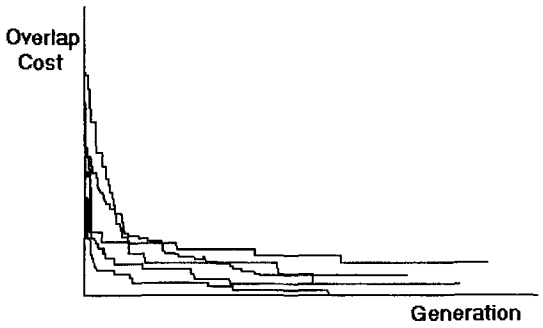

그림 15. 중첩 비용의 변화 과정
Fig. 15. Overlap cost graph.

Ⅳ. 실험결과

제시한 알고리즘은 리눅스의 X 윈도우 시스템에서 C++을 사용하여 구현하였고, 두 가지 구현에 대해서 각각 1600라인 정도 소요되었으며, 사용자와의 인터페이스를 위한 GUI는 그림 16과 같이 모티프(Motif) 라이브러리를 이용하였다. 다음의 결과들은 OTC 영역 위에 5개의 트랙이 있다고 가정하고 얻은 결과이다. Holmes는 150 λ의 높이를 갖고 하나의 OTC에 6개의 트랙이 가능한 셀을 가정하였다[14]. 따라서 제안된 알고리즘이 실제 레이아웃에 적용된다면 본 실험의 결과보다 더 향상된 결과를 얻을 수 있을 것이다.

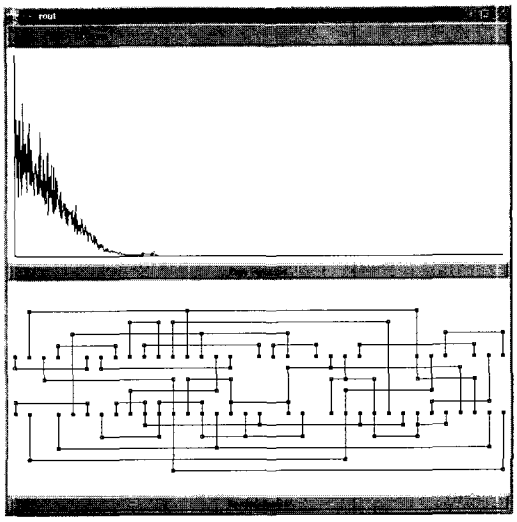

그림 16. 구현된 프로그램
Fig. 16. Implemented program.

1. SA를 이용한 채널 라우터의 실험 결과

그림 17와 18은 제안한 알고리즘을 구현하여 [9]와 [15]의 예제에 적용한 결과이다. [15]에서는 채널의 밀도가 6이었으나 제안된 OTC 라우터의 결과는 1로 크게 채널의 밀도가 감소함을 알 수 있다.

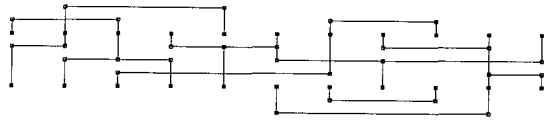

그림 17. OTC 채널 라우팅 결과 1
Fig. 17. OTC channel routing example.

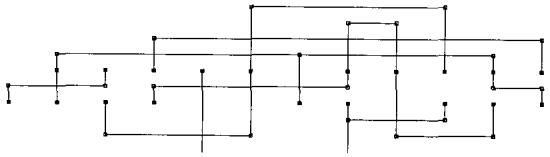

그림 18. OTC 채널 라우팅 결과 2
Fig. 18. OTC channel routing example 2.

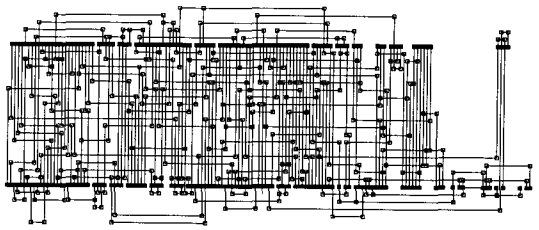

그림 19. OTC 채널 라우팅 결과 3
Fig. 19. OTC channel routing example 3.

그림 19은 deutsch's difficult 문제에 적용한 결과이고, 수행 중 평가함수의 변화는 그림 20이다. 중간에 지역해에 빠지게 되지만 채널의 높이를 증가시킴으로써 효과적으로 지역해에서 탈출함을 알 수 있다.

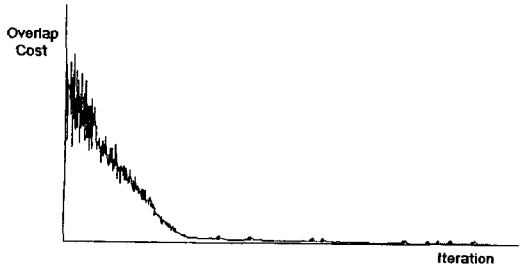

그림 20. 지역해의 탈출
Fig. 20. Escape of local minimum.

표 1은 [6], [8], [9], [15], [16]에서 사용되었던 예제에 대해서 적용해본 결과이다. 도그레그가 필요한 경우에만 삽입되므로 문제가 클수록 비아의 개수가 작고, OTC의 영역을 사용하므로 채널의 밀도가 감소됨을 알 수 있다. 두 번째 결과(Yoshimura)에서 비아의 수가 오히려 증가한 이유는 OTC 영역에서 비아가 존재하지 않는 평면 OTC 라우터의 결과와 비교되었기 때문이다.

표 1. 실험 결과
Table 1. Experimental result.

실험회로	기존 알고리즘		제시한 알고리즘	
	#density	#via	#density	#via
Brouwer	12	67	4	67
Yoshimura	12	56	5	68
Hwang	3	12	3	27
Wong	6	21	1	23
Burstein	19	347	18	326

2. GA과 SA의 통합 알고리즘

GA는 특성상 지역해를 찾는데 어려움을 보인다. 따라서 GA는 탐색의 전 단계로 사용되는 것이 효과적이다[12,13]. 본 논문에서는 GA의 단점을 극복하기 위해서 앞에서 사용되었던 SA를 같이 이용하는 통합 알고리즘을 구현하였다. 그림 21에서 보듯이 평가함수의 값이 감소되는 경향을 보이다가 더 이상 변화가 되지 않는 현상을 보인다. 이 때 SA를 사용하여 전역해에 근접하도록 만든다.

그림 21에서 각각의 그래프는 알고리즘의 수행 중에 채널을 증가시키기 전과 증가시킨 후 새로운 채널의 높이에서 다시 시도된 결과들이다.

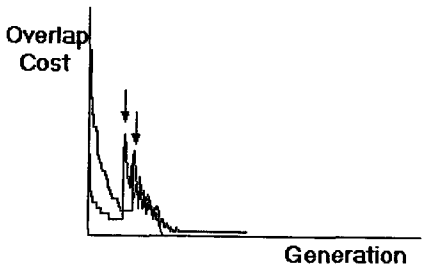

그림 21. 통합 알고리즘의 cost 변화 과정
Fig. 21. Cost graph of hybrid algorithm.

3. SA와 통합 알고리즘의 비교

표 2는 SA를 이용한 채널 라우터와 GA를 이용한 채널 라우터의 실험 결과를 비교한 표이다. SA를 이용한 구현이 GA를 이용한 구현 보다 향상된 결과를 보이고 있다. 이는 GA의 교배와 변이 연산이 채널 라우팅 문제에서는 효과적으로 적용되지 않았기 때문이다. 실험 결과 특히 교배는 초기의 단계에서만 효과를 보일 뿐 이후에는 거의 영향을 주지 못했다. 결과적으로, 여러 가지 해가 존재할 수 있는 채널 라우팅 문제의 특성상 하나의 해에서 출발해 나가는 SA는 만족할 만한 결과를 얻을 수 있었지만, 오히려 GA는 상당히 많은 계산 시간에도 불구하고 좋은 해를 얻지 못했다.

표 2. 라우터의 결과 비교
Table 2. Routing result comparison.

실험회로	SA를 이용한 알고리즘			통합 알고리즘		
	#density	#cost	수행시간(sec)	#density	#cost	수행시간(sec)
Brouwer	4	8.75	3.5	7	13.2	5.3
Wong	1	2.7	0.8	1	3.9	1.2
Burstein	18	115.7	32.4	18	164.4	68.7

V. 결론 및 향후 연구방향

지금까지 채널 라우팅 알고리즘의 복잡도는 공정기술의 발전에 따라 많은 부분이 해결되었다. 이러한 추세에 따라 본 논문에서는 개선된 공정기술에 기초한 OTC 채널 라우팅 알고리즘을 제시하였다.

제안한 알고리즘에서는 문제를 단순화시켜 최적에 가까운 해를 얻기 위해 SA과 GA을 이용하여 구현하였다. CVC를 제거하기 위한 휴리스틱을 제시하였고, 채널 높이의 점진적인 증가를 위해서 알고리즘의 수행 중 지역해에 빠졌음을 감지하여 탈출을 보장하는 방안을 제시하였다. 이로써, 기존의 라우터에 비해 상대적으로 간단한 알고리즘으로 만족할 만한 결과를 얻을 수 있었다.

두 가지 방법으로 구현했을 때 GA를 이용한 구현은 교배의 정의를 확실히 내리지 못하여서 일반적인 생각만으로는 이득이 거의 없었고 대부분 교배가 아닌 변이에 의해서 해에 근접해 갔다. 또한, 기하급수적인 계산 시간을 요구했기 때문에 한세대의 개체수를 크게 잡을 수가 없었다. 따라서 채널 라우팅 문제에 있어서 GA가 갖는 장점을 얻을 수 없었다. 실험 결과, SA를 이용한 구현보다 좋은 해를 얻지 못했고 다수의 개체에 대해서 계산을 하게 되므로 시간상 상당한 오버헤드를 나타내었다.

제시한 알고리즘의 성능을 보다 향상시키기 위하여 온도 T의 스케줄링 파라미터를 실험적으로 결정하고, 임의로 생성되는 상태보다는 효율적으로 상태를 생성할 수 있는 휴리스틱을 개발하여야 한다. 그리고 GA의 유전자 인코딩 방법과 적합도를 크게 고려한 알고리즘에 대한 연구를 수행하여야 한다. 또한, 평가함수에 추가되어야 할 요소로서 라우팅 와이어 간격 등의 전기적 요소를 추가시킴으로써 성능과 면적 면에서 보다 향상된 결과를 얻을 수 있을 것이다.

참 고 문 헌

[1] Jaewon Kim and Sung-Mo Kang, "A New Triple-Layer OTC Channel Router," IEEE Transactions on Computer-Aided Design, pp 1059-1070, 1996.

[2] Siddhart Bhingarde Naveed Sherwani and Anand Panyam, "Routing in the Third Dimension," IEEE Press, 1995.

[3] Anthony D, Johnson and Rongchung Sun, "A Genetic Algorithm for VLSI Channel Routing in the Presence of Cyclic Vertical Constraints," Midwest Symposium on Circuits and Systems, 1996.

[4] Naveed Sherwani, "Algorithms for VLSI Physical Design Automation," Toppan Kluwer, 1996.

[5] Volker Schnecke and Oliver Vornberger, "Genetic Design of VLSI-Layouts," IEEE International Conference on GAs, pp 430-435, 1995.

[6] Randall Jay Brouwer, "Experience with Serial and Parallel Algorithms for Channel Routing using Simulated Annealing," Master Thesis, Univ. of Illinois, pp 9-17, 1985.

[7] David N, Deutsch, "A DOGLEG Channel Router," Design Automation Conference, pp

425-433, 1976.

[8] Takeshi Yoshimura and Ernest S.Kuh, "Efficient Algorithms for Channel Routing," IEEE Transactions on Computer-Aided Design, pp 25-35, 1982.

[9] Chi-Yi Hwang Min-Siang Lin, Hourng-Wern Perng and Youn-Long Lin, "Channel Density Reduction By Routing Over The Cells," Design Automation Conference, pp 120-125, 1991.

[10] Naveed A. Sherwani, Nancy D. Holmes and Majid Sarrafzadeh, "Utilization of Vacant Terminals for Improved Over-the-Cell Channel Routing," IEEE Transactions on Computer-Aided Design, pp 780-792, 1993.

[11] William P. Swartz. Jr, "Automatic Layout of Analog and Digital Mixed Macro/Standard Cell Integration Circuits," PhD Thesis, pp 39-49, 1993.

[12] Zbigniew Michalewicz, "Genetic Algorithms + Data Structure = Evolution Programs," Springer-Verlag, 1992.

[13] L.M. Patnaik B.B, Prahlada Rao and R.C. Hansdah, "An Extended Evolutionary Programming Algorithm for VLSI Channel Routing," Proceedings of the Fourth Annual Conference on Evolutionary Programming, pp 521-544, 1995.

[14] Naveed A. Sherwani, Nancy D. Holmes and Majid Sarrafzadeh, "Algorithms for Three-Layer Over-the-Cell Channel Routing," International Conference on Computer-Aided Design, pp 428-431, 1991.

[15] D. F. Wong, Jingsheng Cong and C. L. Liu, "A New Approach to Three- or Four-Layer Channel Routing," IEEE Transactions on Computer-Aided Design, pp 1094-1104, 1988.

[16] Michael Burstein, "Hierarchical channel router," INTEGRATION, the VLSI Journal, pp 59-74, 1983.

저 자 소 개

張 承 圭(正會員)

1997년 숭실대학교 인공지능학과 졸업(B.S.). 1999년 숭실대학교 전자계산학과 졸업(M.S.). (현재) (주) 인터넷시큐리티 근무. 관심분야는 CAD, VLSI 설계, VLSI 테스팅

朴 在 興(正會員)

1999년 숭실대학교 컴퓨터학부 졸업. 1999년~숭실대학교 컴퓨터학과 석사과정. 관심분야는 CAD. VLSI 설계. VLSI 테스팅

張 勳(正會員)

1987년 서울대학교 전자공학과 졸업(B.S.). 1989년 서울대학교 전자공학과 졸업(M.S.). 1993년 University of Texas at Austin 박사학위 취득. 1991년 IBM Inc. 1993년 Motorola Inc. Senior Membger of Technocal Staff. 1994년~숭실대학교 컴퓨터학부 조교수. 관심분야는 컴퓨터 시스템, VLSI 설계, VLSI 테스팅