4-레이어 채널 배선을 위한 네트리스트 분할 유전자 알고리즘

Netlist Partitioning Genetic Algorithm for 4-Layer Channel Routing

송호정, 송기용
Ho-Jeong Song, Gi-Yong Song
충북대학교 컴퓨터공학과

요 약

최근 VLSI 회로 설계는 자동 레이아웃(automatic layout) 툴을 사용하여 효과적으로 이루어지고 있다. 자동 레이아웃은 VLSI 칩 상에 모듈들의 위치를 결정하는 배치와 각 모듈간을 상호 연결하는 배선 두 가지의 중요한 기능으로 구성되어 있다. VLSI 칩의 성능과 면적은 이 두 가지의 기능을 수행하는 알고리즘의 성능에 따라 크게 좌우된다. 채널 배선은 VLSI 설계 과정중의 하나로, 글로벌 배선을 수행한 후 각 배선 영역에 할당된 네트들을 트랙에 할당하여 구체적인 네트들의 위치를 결정하는 문제이며, 네트들이 할당된 트랙의 수를 최소화하는 문제이다.

본 논문에서는 4-레이어 채널 배선 문제를 해결하기 위한 네트리스트 분할 문제에 대하여 유전자 알고리즘(genetic algorithm; GA)을 이용한 해 공간 탐색(solution space search) 방식을 제안하였으며, 제안한 방식을 여러 문제들에 대해 시뮬레이티드 어닐링 알고리즘과 비교, 분석한 결과 최적, 최악 및 평균 비용 측면에서 더 좋은 결과를 얻을 수 있었다.

Abstract

Current growth of VLSI design depends critically on the research and development of automatic layout tool. Automatic layout is composed of placement assigning a specific shape to a block and arranging the block on the layout surface and routing finding the interconnection of all the nets. Algorithms performing placement and routing impact on performance and area of VLSI design. Channel routing is a problem assigning each net to a track after global routing and minimizing the track that assigned each net.

In this paper we propose a genetic algorithm searching solution space for the netlist partitinoing problem for 4-layer channel routing. We compare the performance of proposed genetic algorithm(GA) for channel routing with that of simulated annealing(SA) algorithm by analyzing the results which are the solution of given problems. Consequently experimental results show that out proposed algorithm reduce area over the SA algorithm.

Key words : genetic algorithm, channel routing, simulated annealing

I. 서론

최근 VLSI 회로 설계는 전자설계자동화(EDA; Electronic Design Automation) 툴을 사용하여 효과적으로 이루어지고 있다. 이러한 EDA 툴은 회로의 레이아웃을 위하여 분할(partition), 배치(placement), 배선(routing) 등의 여러 단계의 과정을 수행하게 된다.

배치 단계에서 회로 블록과 핀들의 위치가 결정되면, 각 블록들 사이에 배선 영역이 생성된다. 배치 단계에서 생성된 배선 영역에 네트들의 위치를 결정하는 것을 배선이라 한다.

배선 문제는 연결선의 구체적인 위치를 결정하지 않고 각 네트들을 배선 영역에 할당시키는 글로벌 배선(global routing)과 각각의 배선 영역에 할당된 네트들의 구체적인 위치를 결정하는 디테일드 배선(detailed routing)의

두 단계로 나눌 수 있으며, 디테일드 배선은 채널 배선(channel routing)과 스위치박스 배선(switchbox routing)으로 구분된다[1][2].

채널 배선 문제는 VLSI 회로의 물리적 설계(physical design) 단계에서 모든 네트들을 최소한의 트랙에 할당하여 배선 영역을 최소화하는 문제이다. 즉 최적의 배선은 모든 네트들을 연결하였을 때 사용된 트랙의 수가 최소가 되도록 연결시키는 것으로써, 배선 결과에 따라 VLSI 칩의 면적에 영향을 미치므로, 배선 영역의 최소화는 VLSI 회로 레이아웃을 위한 EDA 툴에서 매우 중요하다[3][4][5][6].

본 논문에서는 VLSI의 설계 과정 중 글로벌 배선을 수행한 후 각 배선 영역에 할당된 네트들을 트랙에 할당하여 구체적인 네트들의 위치를 결정하는 채널 배선 문제 중, 4-레이어 채널 배선 문제를 해결하기 위한 네트리스트 분할 문제에 대하여 유전자 알고리즘을 이용한 해공간 탐색 방식을 제안하였으며, 이 방식을 시뮬레이트 어닐링 알고리즘과 비교, 분석하였다.

본 논문의 구성은 다음과 같다. II장에서는 본 논문에서 제안한 네트리스트 분할 유전자 알고리즘의 데이터 표현 방법과 알고리즘에 대하여 설명하고, III장에서는 제안한 네트리스트 분할 유전자 알고리즘과 시뮬레이티드 어닐링 알고리즘을 비교한다. 마지막으로 IV장에서는 결론과 향후 연구 방향에 대하여 기술한다.

II. 네트리스트 분할 유전자 알고리즘

유전자 알고리즘은 진화 과정에서 유도된 탐색방법으로, 이 알고리즘은 염색체와 유사한 자료구조를 사용하여 해공간을 부호화하며, 부호화한 자료 구조에 재조합 연산자를 적용하여 염색체들을 진화시킨다.

유전자 알고리즘은 처음에 임의로 선택된 모집단(population)에서 시작하며, 이러한 염색체 집단 중에서 일정한 방식으로 부모 염색체를 선택하고 이들 부모 염색체를 교배시켜 자식 염색체를 생성한다. 새로 생성된 자식 염색체는 평가함수에 의해 평가되며 좋은 평가 결과를 가지는 염색체가 다음 세대에 살아 남을 확률이 높게 된다. 이와 같은 방식으로 유전자 알고리즘은 염색체 집단의 진화를 통하여 최적해에 근접할 수 있으므로, 최적해를 구하기 어려운 여러 NP-문제에 적용될 수 있다[7][8][9].

1. 염색체의 표현

네트리스트 분할 문제는 네트리스트의 네트들을 그룹화 하는 것으로 네트리스트 분할 문제를 유전자 알고리즘으로 표현하기 위해서는 각 네트가 두 그룹 중 어느 그룹에 속하는지를 나타낼 수 있어야 한다.

네트리스트 분할 문제에서 전체 네트의 개수를 n이라고 하면, 유전자 알고리즘의 염색체는 n개의 bit 배열, 즉 비트 스트링을 사용하여 나타낼 수 있다. 네트를 나타내는 염색체의 각 bit는 그룹 A에 속해있는 네트에 해당하는 bit를 '0'으로, 그룹 B에 속해있는 네트에 해당하는 bit를 '1'로 표시하여, 각 그룹으로 분리된 네트들을 그림 1과 같은 염색체로 표현 할 수 있다. 그림 1은 n이 16일 때 네트 0, 3, 6, 7, 11, 12, 15가 그룹 A에 속해있고, 네트 1, 2, 4, 5, 8, 9, 10, 13, 14가 그룹 B에 속해 있는 것을 나타내고 있다.

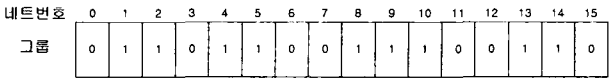

그림 1. 네트리스트 분할 문제의 염색체 표현
Fig. 1. Representation of chromosome for netlist partition problem.

일반적인 시스템에서의 정수형 워드의 크기는 16비트이다. 그러므로 모든 네트들을 표현하려면 그림 1과 같은 비트 스트링을 여러 개를 사용하여야 한다. 즉 그림 2와 같이 여러 개의 정수형 워드 배열을 사용하여 전체 네트들을 표현할 수 있다. β를 정수형 워드의 비트수라고 가정한다면, n개의 네트리스트를 나타내는데 필요한 unsigned integer의 개수 ω는 식(1)과 같이 구할 수 있다.

$$\omega = \lceil \frac{n}{\beta} \rceil \tag{1}$$

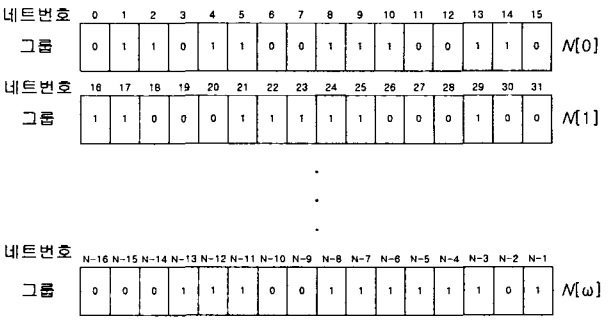

그림 2. n개의 네트리스트를 위한 염색체 표현
Fig. 2. Representation of chromosome for n netlist.

여기서 u번 네트가 위치하는 배열의 첨자 i와 i번째 배열에서 u번 네트가 위치하는 비트 j를 식(2), 식(3)과 같이 구할 수 있다.

$$i = \lfloor \frac{u}{\beta} \rfloor \tag{2}$$

$$j = \beta - 1 - (u \bmod \beta) \tag{3}$$

그림 3은 네트가 위치하는 배열의 첨자 i와 해당 배열에서의 비트 j를 표현하는 예를 보이고 있다.

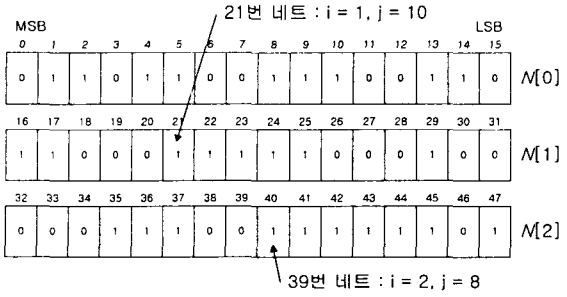

그림 3. 네트의 위치 표현 예

Fig. 3. Example of the location for nets.

만일 β=16이라고 가정한다면, 21번 네트는 다음 계산에 의해서 N[1].10번 비트에 위치하고 있다.

$$i = \lfloor 21/16 \rfloor = 1$$
$$j = 16 - 1 - (21 \bmod 16) = 10$$

또한 같은 방법으로 39번 네트는 N[2].7번 비트에 위치하고 있다.

$$i = \lfloor 39/16 \rfloor = 2$$
$$j = 16 - 2 - (39 \bmod 16) = 7$$

2. 초기 모집단의 생성

네트리스트 분할 유전자 알고리즘의 초기 모집단을 생성하기 위해서는 각 염색체의 각 비트를 해당 네트가 포함되는 그룹(A 또는 B)을 표시하는 '0' 또는 '1' 중의 하나로 결정해야 한다. 이러한 초기 모집단을 생성하기 위해서는 하나의 염색체마다 n번의 랜덤 값($\varepsilon \in [0,1]$)을 발생시켜 각 비트를 결정해야 한다.

본 논문에서는 초기 모집단을 생성하기 위하여 rank와 unrank라는 랭킹 함수를 도입하였다. *rank()* 함수는 n-bit의 염색체를 $0 \sim 2^n$사이의 정수 값을 반환하는 함수이며, *unrank()* 함수는 $0 \sim 2^n$사이의 정수 값을 입력받아 정수 값이 해당하는 염색체를 반환하는 함수이다. 즉 rank() 함수는 식(4)에 의하여 정수 값을 계산할 수 있다.

$$rank(N) = \sum_{i=0}^{n-1} x_i 2^i \tag{4}$$

표 1는 n=3일 때, 비트 스트링 N과 *rank()* 함수사이의 관계를 나타내고 있다. 즉 0~7 사이의 랜덤 값을 하나 발생시키면, 그 값에 해당하는 비트 스트링을 바로 얻을 수 있게된다. 전체 네트의 개수가 n일 때, 앞에서 ω개의 β-bit 비트 스트링을 사용하여 표현하였다. 그러므로 전체 n-비트의 염색체를 생성하기 위해서는 단지 ω번의 랜덤 값만을 발생시키면 된다.

표 1. 비트 스트링 N과 *rank()* 함수사이의 관계

Table 1. Relation with N and *rank()*.

N	A	B	rank(N)
[0 0 0]	{0, 1, 2}	ϕ	0
[0 0 1]	{0, 1}	{2}	1
[0 1 0]	{0, 2}	{1}	2
[0 1 1]	{0}	{1, 2}	3
[1 0 0]	{1, 2}	{0}	4
[1 0 1]	{1}	{0, 2}	5
[1 1 0]	{2}	{0, 1}	6
[1 1 1]	ϕ	{0, 1, 2}	7

다음 알고리즘 1은 ranking 알고리즘을 나타낸다.

알고리즘 1. *rank*(n, N)
Algorithm 1. *rank*(n, N)

```
s ← 0
for i ← 1 to n
   if i∈N then
      s ← s + 2^(n-i)
   end if
end for
return (s)
```

예를 들어 n=16이고 N=[0101101000010010] 이라면, 알고리즘 1에 의하여 다음과 같이 계산할 수 있다.

$$\begin{aligned} rank(n, N) &= 2^{14} + 2^{12} + 2^{11} + 2^9 + 2^4 + 2^1 \\ &= 16384 + 4096 + 2048 + 512 + 16 + 2 \\ &= 23058 \end{aligned}$$

다음 알고리즘 2는 unranking 알고리즘을 나타낸다.

알고리즘 2. *unrank*(n, s)
Algorithm 2. *unrank*(n, s)

```
N ← φ
for i ← 1 to n
   if s mod 2 = 1 then
      N[i] ← '1'
   else
      N[i] ← '0'
   end if
   s ← ⌊s/2⌋ =1
```

end for
return (N)

예를 들어, n=16이고, s=23058 이라면, 알고리즘 2에 의하여 그림 4과 같이 계산할 수 있다.

그림 4. *unrank*(n, s)의 계산 예

Fig. 4. Calculation example of the *unrank*(n, s).

3. 평가함수(Evaluation Function)

유전자 알고리즘이 진행되는 동안 현재 모집단의 개체들은 특정 평가함수에 의해 평가된다. 네트리스트 분할 문제에서의 평가함수는 분할된 각 그룹의 '사용 가능한 최소 트랙 수' 중 큰 그룹의 수를 나타내며, 이 값이 작을수록 더 좋은 적합도를 가지게 된다.

사용 가능한 최소 트랙 수는 그림 5과 같은 간격 그래프(interval graph)를 사용하여 구할 수 있다. 그림 5에서 (a)위치에서 최대로 5개의 네트가 겹치게 된다. 이것은 적어도 5개의 트랙이 필요하다는 것을 의미한다.

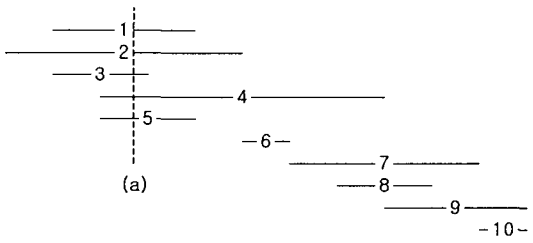

그림 5. 간격 그래프

Fig. 5. Interval graph.

4. 교배 연산자(Crossover Operator)

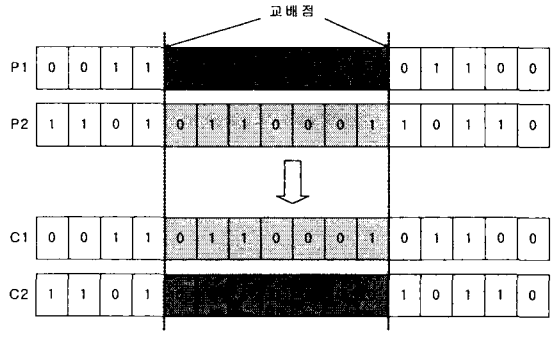

그림 6. 네트리스트 분할 2-점 교배 연산

Figure 6. 2-point crossover operation for netlist partition.

네트리스트 분할 유전자 알고리즘에서 사용한 교배 연산자는 2-점 교배 연산자를 사용하였다. 그림 6과 알고리즘 3은 네트리스트 분할 2-점 교배 연산의 구체적인 연산 방식을 기술한 것이다.

알고리즘 3. 네트리스트 분할 2-점 교배 연산

Algorithm 3. 2-point crossover operation for netlist partition.

1단계 : 교배를 하기 위한 2개의 교배점(cross point)을 랜덤 하게 선택한다.

2단계 : 부모 염색체 P1에서 두 개의 교배점에 의해 만들어진 가운데 부-염색체(sub-chromosome)를 자식 염색체 C2의 같은 위치에 복사한다. 자식 염색체 C1도 같은 방식으로 복사한다.

3단계 : 부모 염색체 P1에서 가운데 부-염색체를 제외한 나머지 유전자들을 C1의 같은 위치에 복사한다. C2도 같은 방법으로 생성한다.

5. 돌연변이(Mutation)

유전자 알고리즘에서 각 세대의 모집단은 진화를 진행하면서 얻고자 하는 해에 가까운 염색체들로 구성된 모집단으로 수렴하게 되지만, 그 결과가 최적해가 아닌 지역해로 수렴할 수도 있으며, 이러한 지역해로의 수렴을 막기 위하여 돌연변이(mutation) 연산을 수행하게 된다.

그림 7과 알고리즘 4는 네트리스트 분할 돌연변이 연산의 구체적인 연산 방식을 기술한 것이다.

알고리즘 4. 네트리스트 분할 돌연변이 연산

Algorithm 4. Mutation operation for netlist partition.

1단계 : 전체 염색체에서 각 유전자마다 0에서 1사이의 랜덤 값을 하나 발생시킨다.

2단계 : 발생시킨 랜덤 값이 돌연변이 율보다 크면 해당 유전자를 돌연변이 시킨다('0'→'1' 또는 '1'→'0').

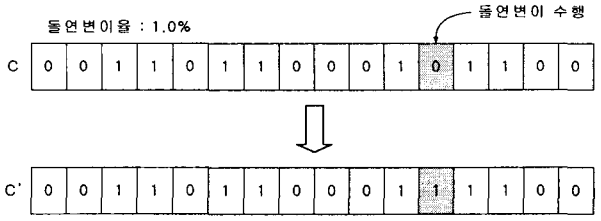

그림 7. 네트리스트 분할 돌연변이 연산

Figure 7. Mutation operation for netlist partition.

6. 네트리스트 분할 유전자 알고리즘

알고리즘 5는 앞에서의 기술한 방식을 사용한 네트리스트 분할 유전자 알고리즘을 나타낸다.

알고리즘 5. 네트리스트 분할 유전자 알고리즘
Algorithm 5. Genetic algorithm for netlist partitioning

단계 0 : 파라미터들의 설정
개체의 수를 나타내는 *pop_size*, 돌연변이 율을 나타내는 P_m, 그리고 최대 생성 횟수를 나타내는 *max_gen*을 설정한다.
단계 1 : 초기 모집단의 생성
각기 다른 *pop_size* 만큼의 객체 $S_i(i=1, \cdots, pop_size)$를 랜덤 하게 생성한다.
단계 2 : 모든 모집단의 비용을 계산하고 최대, 최소 객체를 기억한다.
단계 3 : 교배
3.1 : 모집단으로부터 두 개의 부모 염색체 P1, P2를 랜덤 하게 선택한다.
3.2 : 2-점 교배 연산자를 사용하여 새로운 자식 염색체 C1, C2를 생성
단계 4 : 돌연변이
랜덤 하게 $\varepsilon \in (0,1]$을 생성하고 돌연변이 율 P_m과 비교한다. 만일 $\varepsilon < P_m$이면 돌연변이 연산을 수행한다.
단계 5 : 만일 생성된 자식 염색체 C1 또는 C2가 모집단 내의 어느 객체와 같다면, C1 또는 C2가 같은 객체가 없는 새로운 객체가 될 때까지 돌연변이 연산을 수행한다.
단계 6 : 새로운 세대의 구성
모집단 내의 최대 비용을 갖는 두 개의 염색체를 삭제하고 생성된 자식 염색체 C1과 C2를 모집단 내에 추가한다.
단계 7 : 새로운 염색체의 생성 횟수가 *max_gen*이 될 때까지 단계 1부터 6까지를 반복한다. 최적해는 모집단 내에서 가장 작은 비용을 갖는 염색체이다.

III. 시뮬레이션

네트리스트 분할 유전자 알고리즘을 시뮬레이션 하기 위하여 각 8, 20, 40, 60, 80, 100, 120개의 네트를 갖는 네트리스트를 자동으로 생성하여 동일 네트리스트 시뮬레이티드 어닐링과 유전자 알고리즘을 각 100번씩 수행시켰고, 또한 벤치마크 네트리스트인 Deutsch's difficult problem중 deutsch52, deutsch72, deutsch218에 대하여 동일 알고리즘을 각 100번씩 수행시켜 그 결과를 분석하였다.

시뮬레이티드 어닐링은 조합 최적화 문제 해결에 적용되는 대표적 반복 휴리스틱 알고리즘으로 Timber-Wolf[10]등 실용 패키지뿐 아니라 알고리즘의 성능을 비교하는 벤치마크로 주로 사용된다.

네트리스트 분할 유전자 알고리즘의 시뮬레이션에서는 유전자 알고리즘의 돌연변이 율 P_m=0.01(1%), 모집단내의 개체의 수 *pop_size*=20, 최대 수행 횟수 *max_gen*=100, 시뮬레이티드 어닐링의 한 온도에서의 알고리즘수행 횟수 M=10, 최대 수행 횟수 *max_time*=1000, 초기 온도 T=10, 냉각 파라미터 α=0.9, 그리고 β=1.0의 값을 사용하였다.

표 2는 자동으로 생성된 네트리스트의 예를 보인 것이며, 그림 8은 표 2과 같은 네트리스트가 배선 영역에서 구성되어있는 실제 모양을 표현한 예이다.

표 2. 생성된 네트리스트 예
Table 2. Example of the generated netlist.

네트	터미널
1	T1, T5, B7, B9
2	T2, T8
3	B1, B3, B8
4	B0, B5
5	B2, T3, B4
6	T6, B10
7	T4, B6
8	T9, B11

T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
0	1	2	5	7	1	6	0	2	8	0	0

4	3	5	3	5	4	7	1	3	1	6	8
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11

그림 8. 채널 배선 영역
Figure 8. Routing region for the channel.

그림 9는 표2의 네트리스트에 대한 간격 그래프를 보이고 있다. 그림 9의 (a) 위치에서 6개의 네트가 겹치기 때문에 최소한 6개의 트랙이 필요하다는 것을 의미한다.

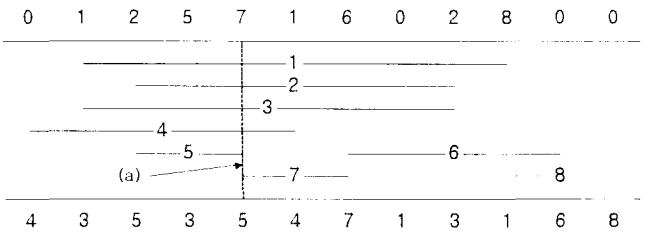

그림 9. 네트리스트의 간격 그래프

Figure 9. Interval graph for netlist.

그림 10은 그림 9과 같은 간격그래프를 가지는 네트리스트를 분할한 결과를 보이고 있다.

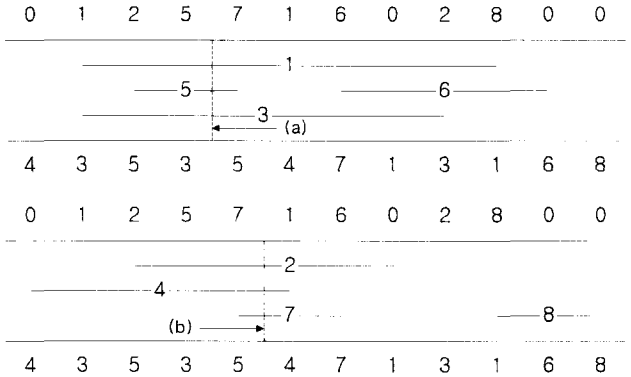

그림 10. 분할된 네트리스트의 간격 그래프

Figure 10. Interval graph for partitioned netlist.

네트리스트 분할 유전자 알고리즘의 성능을 분석하기 위하여 자동 생성된 네트리스트와 deutsch52, deutsch72, deutsch218에 대한 시뮬레이티드 어닐링과 유전자 알고리즘 적용 결과의 최적값, 최악값, 평균값을 구하여 각 알고리즘을 비교하였다.

표 3와 표 4에서 네트리스트 종류는 자동 생성된 네트리스트의 종류를 의미하며 네트 개수는 생성된 각 네트리스트의 네트 개수를 의미한다. 또한 최적, 최악, 평균은 생성된 각 네트리스트에 대해 두 개의 알고리즘을 100번씩 수행한 총 100개의 결과 중 최적 비용, 최악 비용 그리고 비용의 평균값을 의미한다.

그림 11, 12, 13은 최적 비용, 최악 비용, 평균 비용 시뮬레이션 결과를 그림으로 나타낸 것이다.

두 알고리즘의 구현 결과를 분석해보면 자동 생성 회로의 경우 최적 비용, 최악 비용, 평균 비용 모두 유전자 알고리즘이 시뮬레이티드 어닐링보다 빠르게 최적해에 수렴하는 것을 알 수 있으며, 이는 유전자 알고리즘의 효과적인 해공간 탐색 방법에 기인한다.

표 3. SA와 GA의 자동 생성 네트리스트 테스트 결과

Table 3. Generated netlist test result for SA, GA.

네트리스트 종류	네트 개수	시뮬레이티드 어닐링			유전자 알고리즘		
		최적	최악	평균	최적	최악	평균
ex01	8	4	4	4	4	4	4
ex02	20	11	12	11.5	11	12	11.5
ex03	40	16	16	16	14	16	15
ex04	60	17	19	18	16	18	17
ex05	80	18	22	20	18	21	19.5
ex06	100	26	31	28.5	25	29	27
ex07	120	30	36	33	27	33	29.5

표 4. SA와 GA의 벤치마크 테스트 결과

Table 4. Benchmark test result for SA, GA.

네트리스트 종류	네트 개수	시뮬레이티드 어닐링			유전자 알고리즘		
		최적	최악	평균	최적	최악	평균
deutsch52	52	15	19	17	15	18	16.5
deutsch72	72	18	23	20.5	17	21	19
deutsch218	218	47	56	51.5	43	55	49

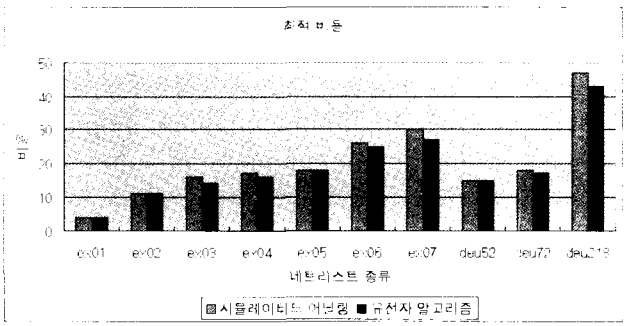

그림 11. 최적 비용 비교

Figure 11. Comparison of the best cost.

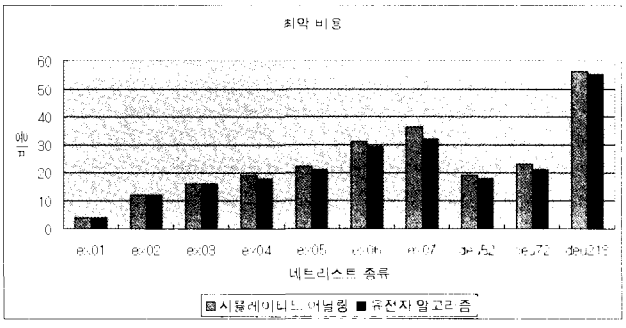

그림 12. 최악 비용 비교

Figure 12. Comparison of the worst cost.

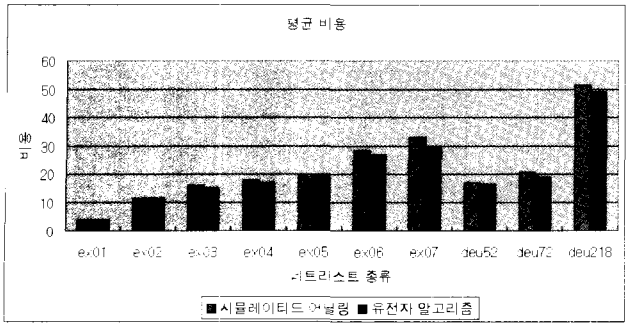

그림 13. 평균 비용 비교

Figure 13. Comparison of average cost.

두 알고리즘의 시뮬레이션 결과 네트리스트 분할 유전자 알고리즘이 시뮬레이티드 어닐링보다 더 좋은 결과를 얻는 것을 알 수 있다.

시뮬레이티드 어닐링은 초기에는 지역해에서 벗어날 수 있는 확률이 높으나, 냉각과정이 진행되면서 지역해에서 벗어날 수 있는 확률이 낮아져서 일정 횟수를 수행한 후에는 더 좋은 해를 찾을 가능성이 작아지게 된다. 반면에 유전자 알고리즘은 교배와 돌연변이 연산을 사용한 효과적인 해공간 탐색으로 시뮬레이티드 어닐링에 비해 더 좋은 최적해를 찾게 된다.

IV. 결론

본 논문에서는 VLSI 설계 과정 중 4-레이어 채널 배선 문제를 해결하기 위한 네트리스트 분할 문제에 대하여 유전자 알고리즘을 이용한 해 공간 탐색 방식을 제안하였으며, 이 방식을 시뮬레이트 어닐링 알고리즘과 비교, 분석하였다.

제안한 네트리스트 분할 유전자 알고리즘과 시뮬레이티드 어닐링 방식을 임의로 생성한 네트리스트와 벤치마크 네트리스트에 적용하여 비교, 분석한 결과 제안한 네트리스트 분할 유전자 알고리즘이 시뮬레이티드 어닐링 방식보다 더 효과적으로 최적해에 근접하는 것을 알 수 있었다.

접수일자 : 2002. 12. 02 수정완료 : 2003. 1. 15

참고문헌

[1] S. M. Sait, H. Youssef, *VLSI Physical Design Automation Theory and Practice*, World Scientific Publishing, 2001.

[2] Naveed A. Sherwani, *Algorithms for VLSI Physical Design Automation. 3rd Edition*, Kluwer Academic Publishers, 2001.

[3] A.Hashimoto and J.Stevens. "Wire routing by optimizing channel assignment within large apertures," *Proceedings of 8th Design Automation Conference*, pages 155-169, 1971.

[4] D.N.Deutch. "A dogleg channel router," *Proceedings of 13th Design Automation Conference*, pages 425-433, 1976.

[5] R.L.Rivest and C.M.Fiduccia. "A greedy channel router," *Proceedings of 19th Design Automation Conference*, pages 418-424, 1982.

[7] S. M. Sait, H. Youssef, *Iterative Computer Algorithms with Applications in Engineering*, Computer Society, 1999.

[8] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addision-Wesley Publisher Company, Inc., 1989.

[9] L. Davis. *Handbook of Genetic Algorithms*, van Nostrand Reinhold, New York, 1991.

[10] C.Sechen and A.L.Sangiovanni-Vincentelli, "Timberwolf3.2: A new standard cell placement and global routing package," *Proceedings of 23rd Design Automation Conference*, pp. 432-439, 1986.

송호정 (Ho-Jeong Song)
準會員
1994년 배재대학교 물리학과 이학학사
1996년 청주대학교 전자공학과 공학석사
2003년 충북대학교 컴퓨터공학과 공학박사
관심분야 : VLSI 설계, High-level Synthesis

송기용 (Gi-Youn Song)
正會員
1974~80년 서울대, 동대학원(전자공학)
1995년 Univ. of SouthwesternLouisiana 공학박사
1983년~현재 충북대학교 공과대학
컴퓨터공학과 재직중
관심분야 : 컴퓨터구조, VLSI 설계등