O-tree 표현법을 이용한 개선된 플로어플랜 알고리즘

박재민° 허성우
동아대학교 컴퓨터공학과 설계자동화 실험실
parkings88°@dreamwiz.com, swhur@dau.ac.kr

Improved Floorplan Algorithm using O-tree Representation

Jaemin Park° Sungwoo Hur
Dong-A University Design Automation Lab.

요 약

본 논문은 기존의 O-tree 표현법을 이용한 플로어플랜 알고리즘의 결점을 보완한 새로운 알고리즘을 제안한다. 기존의 방법에선 플로어플랜의 변형을 처리하는 과정에서 몇 가지 변형을 간과하기 때문에 좋은 해를 놓치는 경우가 발생한다. 본 논문에서는 기존의 방법을 수정하여 변형을 처리하는 과정에서 블록이 들어갈 수 있는 모든 위치를 고려하였다. 그 결과 MCNC 벤치마크 회로를 이용한 실험에서 총면적이 이전의 방법에 비해 평균 3% 개선되었다.

1. 서 론

플로어플랜(Floorplan)은 VLSI회로를 설계하는 중요한 단계 중의 하나이다. 플로어플랜의 목표는 회로 내부의 사각형 모듈들을 겹치지 않게 하면서 각각의 목적 함수에 맞게 최소화하는 것이다. 여기서 말하는 각각의 목적 함수는 회로의 총 면적이나 총 배선길이 혹은 이 두 가지의 적절한 조합이 될 수가 있다.

플로어플랜을 위해 여러 가지 방법이 사용되는데 대표적으로 O-tree(ordered-tree) 표현법[1]을 이용한 방법, Sequence-Pair 표현법[4]을 이용한 방법, BSG 구조[5]를 이용한 방법, CBL(Corner Block List)[3]를 이용한 방법 등이 존재한다.

플로어플랜을 위한 방법 중 하나인 O-tree표현법을 이용한 결정적 알고리즘[1]은 순서화된 트리(Ordered-tree)를 이용하여 플로어플랜에 필요한 정보를 저장하며 결정적 알고리즘을 이용하여 회로의 전체 면적을 줄인다.

이후 O-tree표현법을 이용한 결정적인 알고리즘을 이용하는 방법은 동일하지만 연산시간을 줄인 방법이 등장했다. 결정적 알고리즘을 진행하는 과정에서 회로의 넓이를 바로 구하는 방법을 간소화하여 기존의 결정적인 알고리즘과 비슷하지만 처리 속도를 높힌 ENPA[2]가 바로 그것이다.

본 논문에서는 O-tree를 이용한 결정적 알고리즘[1]에서 탐색하지 못하는 해 공간을 찾아보는 보다 개선된 알고리즘을 소개한다.

본 논문은 다음과 같이 구성되어 있다. 2장에서 기존의 플로어플랜을 위한 O-tree표현법에 대한 결정적 알고리즘에 대해 간단한 소개를 하고 있으며, 3장에서는 기존의 결정적 알고리즘을 수정하여 새로 제안된 방법을 보여준다. 4장에서는 기존의 방법과 새로 제안된 방법을 비교하여 두 방법 대한 차이점을 실험 결과를 통해 보여준다.

2. O-tree표현법

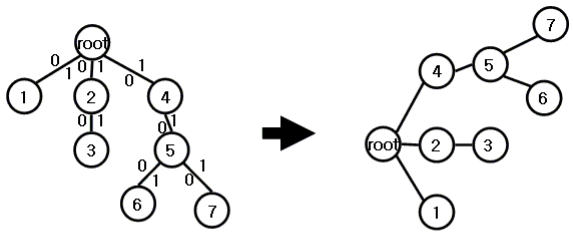
2.1 O-tree 인코딩

n-노드 O-tree는 n+1개의 노드들을 가지는 tree를 말한다. 이것은 1개의 root노드와 n개의 노드로 이루어진 O-tree라는 것을 의미한다. O-tree는 T와 π를 이용하여 O-tree의 정보를 유지할 수 있다. T는 2n개의 2진수를 저장하며, π는 n개의 노드 인덱스를 저장한다. O-tree를 T와 π의 정보로 변환하는 과정을 O-tree 인코딩이라 하며 O-tree 인코딩이 이루어지는 방법은 다음과 같다.

O-tree의 root노드를 최초의 노드로 보고 DFS(Depth-First-Search)순서에 따라 방문을 하며 노드를 선택한다. root노드를 기점으로 시작하여 다시 root노드로 돌아오기까지 DFS순서로 방문을 하는데 이 과정에서 노드들 사이의 관계를 조사하여 T에 저장한다. 다음 순서의 간선이 자식 노드를 가리킬 경우 '0'을 부모 노드를 가리킬 경우 '1'을 저장한다. 모든 노드들을 방문한 후 root노드에 도달하면 O-tree의 탐색이 끝나게 된다. T의 경우 처음 노드인 root노드에서 시작할 때는 다음 노드가 반드시 자식 노드이어야만 하고, 마지막 탐색에서는 항상 root노드가 마지막 노드의 부모 노드가 되기 때문에 T원소의 처음과 끝은 반드시 0과 1이 된다.

그림 1의 좌측 O-tree를 참고로 예를 들면, 이는 root노드를 제외하고 7개의 노드로 이루어진 O-tree이다. root노드를 시작점으로 DFS순서에 따라 노드를 탐색해보면 1번, 2번, 3번, 4번, 5번, 6번, 7번 노드의 순서대로 탐색이 이루어진다. 이 정보는 π에서 저장하고 있다. 다음 DFS순서에 의해 노드들을 탐색하는 과정에

서 자식노드로 이동하는 경우와 부모 노드로 이동하는 경우를 구분하여 T에 '0'과 '1'을 저장한다. 탐색의 시작점인 root노드에서 첫 번째 순서인 1번 노드로 이동하고 '0'을 저장한다. 자식 노드가 없는 1번 노드는 다시 부모 노드인 root노드로 복귀하고 이때 T에 '1'이 저장된다. 다음 순서로 2번 노드로 이동하고 마찬가지로 0을 저장한다. 같은 방법으로 모든 노드들을 탐색하고 마지막으로 root노드로 돌아오면 T의 원소들은 (01001100010111)이 된다. 탐색하는 과정에서 하나의 노드에 대해 반드시 '0'과 '1'이 발생한다. 따라서 O-tree의 노드가 n개 일 때 T는 2n개의 정보를 가진다.



T: 01001100010111

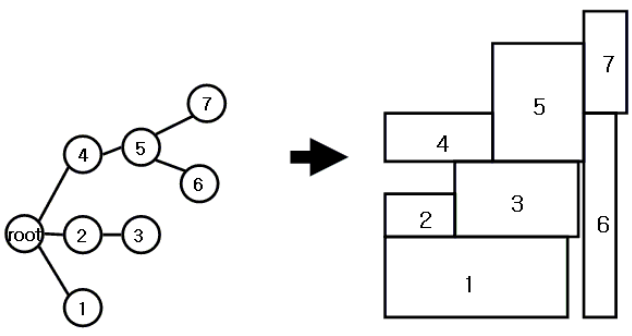
π: 1 2 4 3 5 6 7

그림 1. O-tree와 수평 O-tree

2.2 수평 O-tree와 배치

수평 O-tree(Horizontal-O-tree)는 그림 1의 좌측 O-tree를 그림 1의 우측 그림과 같이 기존의 O-tree를 시계 반대 방향으로 90° 회전시킨 모양이다.

O-tree를 수평 O-tree로 바꾼 후 노드들의 관계를 이용하여 배치를 얻는다. 원래의 O-tree기준으로 DFS 순서에 따라 노드를 선택하고 선택된 노드부터 하나씩 배치를 시작한다.



T: 01001100010111

π: 1 2 4 3 5 6 7

그림 2. 수평 O-tree와 배치

구체적으로 배치되는 과정은 다음과 같다. 먼저 π로부터 노드를 하나씩 선택을 한 후 선택된 노드의 x좌표와 y좌표를 구한다. 노드의 너비와 높이는 노드와 대응되는 블록의 너비와 높이를 말한다. 노드의 x좌표는 부모 노드의 x좌표와 부모 노드의 너비를 합한 값이 된다. 여기서 root노드는 x좌표와 y좌표 그리고 너비와 높이를 0으로 둔다. root노드가 실제로 존재하는 노드가 아니고 단지 root의 자식 노드들의 x좌표와 y좌표를 계산하는 것에 도움을 주는 역할을 하기 때문에 root노드의 좌표값은 고정시킨다. root노드를 제외하면 부모 노드가 없는 노드는 없기 때문에 모든 노드의 x좌표를 구할 수 있다.

노드의 y좌표는 이미 배치된 블록과 겹치지 않는 범위 안에서 가장 낮은 y좌표를 선택한다. 이를 위해서 contour map이란 것을 사용한다.

contour map[1]이란 모든 x좌표에 대한 각각의 높이 정보를 유지하고 있는 구조로 이루어져있다. contour map의 초기값은 모든 x좌표에 대해 높이 0을 갖고 있으며, 하나의 블록이 추가될 때마다 블록과 겹치는 x좌표에 대한 높이를 추가한다. contour map에서 유지하는 높이의 정보는 블록에 관한 높이가 아니라 전체 배치에서 블록의 최상단에 이르는 높이를 말하는 것이다. 앞서 말했듯이 이 정보는 하나의 노드가 배치될 때마다 갱신되며 이 정보를 이용하여 차후 블록들의 y좌표를 모두 결정한다.

부모 노드를 이용하여 x좌표를 결정하고 contour map을 이용하여 y좌표를 결정하면 하나의 노드에 대한 x, y좌표가 모두 결정되어 진다. 이 과정을 통해 모든 블록은 바닥 쪽으로 밀착된다.

2.3 직교 제약 그래프(Orthogonal Constraint Graph)

직교 제약 그래프(Orthogonal Constraint Graph)는 실제 배치에서 다시 O-tree를 얻기 위한 과정이다. 2.2절에서 O-tree를 배치하기 위해 수평 O-tree를 사용했다. 직교 제약 그래프는 현재 배치된 블록들 기준에서 제일 밑바닥을 root로 보고 직접 맞닿은 블록들의 정보를 이용하여 그래프를 구성한다. 가장 밑바닥부터 그래프를 구성하는 이유는 수평 O-tree를 이용한 배치가 끝나면 모든 블록은 root혹은 다른 블록에 의해 사각형의 밑바닥을 공유하는 부분이 반드시 존재하기 때문이다. 직교 제약 그래프를 만드는 이유는 이것을 이용하여 O-tree를 얻은 후, 얻어진 O-tree를 다시 수평 O-tree로 변형하여 배치하면 수용할 만한 배치(admissible placement)가 만들어지기 때문이다. 수용할 만한 배치란 모든 블록이 왼쪽과 아래쪽 방향으로 더 이동할 수 없는 상태가 된 배치를 말한다. 수용할 만한 배치를 만드는 이유는 회로 전체의 면적을 최소로 하기 위함이다.

2.4 결정적 알고리즘

2.2절과 2.3절에서 O-tree를 배치하는 과정과 특정 배치에 대해 다시 O-tree로 만드는 과정을 살펴보았다. 이 과정을 반복함으로써 수용할 만한 배치를 만들 수 있다.

O-tree 표현법에서 이용되는 결정적 알고리즘의 기본 아이디어는 전체 회로에서 하나의 블록을 빼고 해당 블

록을 어떤 위치에 삽입하는 과정을 반복하는 것이다. 그리고 이 과정을 통해 바뀐 배치에 대해서 2.2와 2.3의 과정을 반복 실행함으로써 배치를 수용할 만하게 만든다. 이 과정을 반복함으로써 이 과정 중에 생성된 배치 중 최소 면적을 가지는 배치를 발견하는 것이 결정적 알고리즘의 최종 목표이다.

이 과정을 구체적으로 설명하면 다음과 같다. O-tree에서 특정한 블록에 대응되는 노드의 정보는 T와 π에 정의되어 있다. T에서 특정한 노드를 의미하는 '0'과 '1'을 삭제하고 π에서 삭제한 노드의 인덱스를 찾아 삭제하면 이후의 T와 π는 노드하나가 빠진 O-tree 정보를 갖게 된다. 삭제된 노드는 O-tree에 다시 들어가게 되는데 기존의 결정적 알고리즘에 의하면 O-tree에서 단말로 들어가는 공간만 찾는다. O-tree에 단말로 존재하는 노드는 반드시 T에서 '01'로 존재한다. 또한 T에서 '01'은 반드시 단말로 존재하는 노드를 나타낸다. 따라서 새로운 노드가 단말로 들어가기 위해서 T의 원소들 사이에서 '01'로 삽입되어야 한다. 그림 3에서 8번 노드가 단말로 삽입되는 예를 보이고 있다. 이 O-tree의 T의 정보는 (01001100010111)이다. 8번 노드가 단말로 삽입된다고 했을 때 T의 2번째 원소 이후 '01'이 삽입된다면 (01'01'001100010111)이 된다. 이렇게 된다면 1번 노드와 2번 노드 사이의 공간에 root노드의 자식으로 8번 노드가 단말로 들어오게 된다.

그림 3에서 보이는 단말이 되는 위치는 총 15개가 된다. 노드가 7개 이므로 T의 원소는 14개 이고 T의 원소 사이마다 들어갈 수 있는 모든 공간은 15개가 된다. 이처럼 O-tree에서 노드의 개수가 n개일 때 T의 원소는 2n이므로 2n+1만큼의 삽입될 수 있는 공간이 있다.

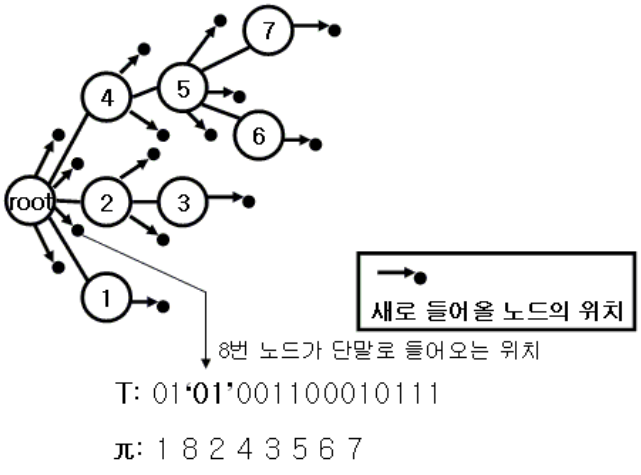


그림 3. 새로 들어갈 노드가 단말이 되는 위치

하나의 노드를 삭제하고 그것을 2n+1만큼의 공간에 삽입해보는 과정을 모든 노드에 대해 적용하려면 n(2n+1)번의 연산이 필요하게 된다.

3. 개선된 알고리즘

기존 결정적 알고리즘의 한계는 삽입되려는 노드가 O-tree의 가능한 모든 위치에 삽입되는 것이 아니라 단말이 되는 곳으로만 삽입된다는 점이다. 이것은 기존의 방법으로는 O-tree로 표현 가능한 모든 배치를 찾을 수 없다.

본 논문에서는 2.4절에서 말한 바와 같이 삽입되려는 노드가 단말이 되지 않는 경우도 고려하는, 즉 모든 경우를 고려하여 삽입하는 방법을 제안한다. 기존의 방법에서는 O-tree의 T에 모든 원소사이에 새로운 노드의 '01'을 넣어보았다. 모든 경우에 대해 노드를 삽입시켜 보기 위해 새로운 노드의 '0'과 '1'을 분리하여 넣는 방법을 소개한다.

그전에 O-tree의 T에 관련된 주요한 정리들을 살펴보자.

정리 1: O-tree의 T에서 i번째 지점까지 '0'의 개수는 i번째 지점까지 '1'의 개수보다 반드시 크거나 같다.
증명: O-tree의 T는 root부터 시작하여 root로 끝나는 노드들의 부모 자식 관계를 나타내는 정보이다. 특정 지점에서 자식 노드로 가는 경로보다 부모 노드로 가는 경로가 더 많다고 한다면 최초 시작 노드인 root의 부모를 찾아 가야하는 일이 발생한다. 이것은 root가 최상위 노드라는 조건과 맞아 떨어지지 않는다. 결국 O-tree의 T에서 특정한 지점까지 '0'의 개수가 '1'의 개수보다 많거나 같아야 한다.□

정리 2: O-tree의 T에서 임의의 노드 i에 대응하는 '0'과 '1'이 사이에 있는 '0'과 '1'의 개수는 같다.
증명: T에서 임의의 노드 i에 대응하는 '0'과 '1'이 존재하는 것을 DFS의 성질에서 쉽게 볼 수 있다. 노드 i에 대응하는 '0'과 '1'사이에 또 다른 '0'과 '1'이 존재한다면 노드 i의 자식 노드가 하나 이상 존재함을 의미한다. 노드 i의 '0'과 '1'사이에 또 다른 '0'과 '1'이 없다면 노드 i는 단말로 존재하는 노드임을 의미한다. 노드 i가 단말이 아닐 경우 반드시 자식 노드를 갖는데 그럴 경우에 노드 i의 '0'과 '1'사이는 DFS의 성질에 의해 '0'과 '1'이 짝지어 존재한다.□

위의 2가지 조건을 만족시키면서 모든 경우를 고려해보는 방법은 다음과 같다.

먼저 T의 모든 원소들 사이에 언제나 '0'을 넣을 수 있다. 어디에 '0'이 들어가더라도 그것 때문에 해당지점까지의 '0'의 개수가 '1'의 개수보다 작아지는 일은 없기 때문에 첫 번째 조건에 어긋나지 않는다. T에서 삽입될 노드 i의 '1'은 노드 i의 '0'이 삽입된 위치 이후에 삽입된다. 노드 i의 '0' 이후로 시작되는 '0'과 '1'의 개수가 짝이 맞을 때 노드 i의 '1'이 들어갈 수 있다.

그림 4에서 8번 노드가 기존의 방법에 따라 단말로 삽입되는 경우의 O-tree를 보여준다. 8번 노드의 '0'이 T의 가장 앞에 삽입될 경우에 8번 노드의 '1'은 삽입된 '0'의 바로 뒤에 위치하는 방법만 고려하고 있다. 그 결과 O-tree에서 8번 노드는 DFS순서에서 가장 첫 번째

단말 노드로써 삽입된다.

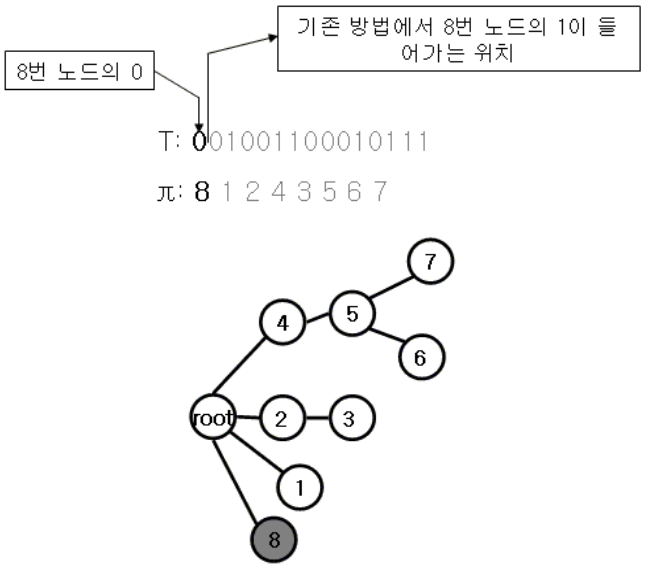


그림 4. 8번 노드가 단말로 들어가는 경우

그림 5에서는 8번 노드가 단말이 아닌 다른 경우로 삽입되는 경우를 보여준다. 8번 노드의 '0'이 T의 가장 앞에 삽입되는 점에서 그림 4와 동일하지만 그림 5에서 8번 노드의 '1'은 삽입된 '0'의 이후부터 '0'과 '1'의 개수가 같은 지점(i, j, k)을 찾아 각각의 경우 '1'을 삽입한다. 그림 5의 경우 삽입되는 경우가 단말이 아니며 기존의 방법에 비해 고려되는 경우가 더 많다. 그림 5의 <a>, <b>, <c>는 각각 i지점, j지점, k지점에 '1'이 삽입 되었을 때 O-tree를 나타낸 것이다.

본 논문에서 제안한 개선된 알고리즘은 다음과 같이 정리할 수 있다.

```
1. Best에 현재의 배치 넓이를 넣는다.
2. π에 있는 임의의 노드 i에 대해 다음을 반복한다.
{
  2.1 π에서 i를 삭제한다.
  2.2 노드 i에 대응하는 '0'과 '1'을 T에서 삭제한다.
      이때 T의 길이는 2(n-1)이다.
  2.3 노드 i에 대응하는 '0'을 삽입할 수 있는 2n-1
      개의 위치 x 각각에 대해 다음을 반복한다.
   {
     2.3.1 T에서 x지점에 '0'을 삽입한다. π에서 대
           응되는 지점에 노드 i를 삽입한다.
     2.3.2 방금 삽입된 '0'에 대응하고 '1'을 넣을 수
           있는 모든 위치y에 대해 다음을 반복한
           다.
      {
        2.3.2.1 T에서 y지점에 '1'을 삽입한다.
        2.3.2.2 T와 π에 대응하는 배치를 구한다.
        2.3.2.3 배치의 넓이를 계산하고 필요하면
                Best를 갱신한다.
      }
   }
}
```

기존의 결정적 알고리즘과 마찬가지로 개선된 알고리즘도 모든 노드에 대해 한번 실행한다. 기존의 알고리즘과 가장 큰 차이점은 기존 알고리즘이 삽입된 '0'이후 '1'의 위치를 바로 다음으로 결정해버리는 반면, 개선된 알고리즘에서는 삽입된 '0'이후 가능한 '1'의 위치를 모두 찾는다는 점이다.

모든 경우를 찾기 위해서 삽입될 노드가 단말로 들어가는 경우, 즉 0과 1이 연속으로 배치되는 경우도 포함해야한다.

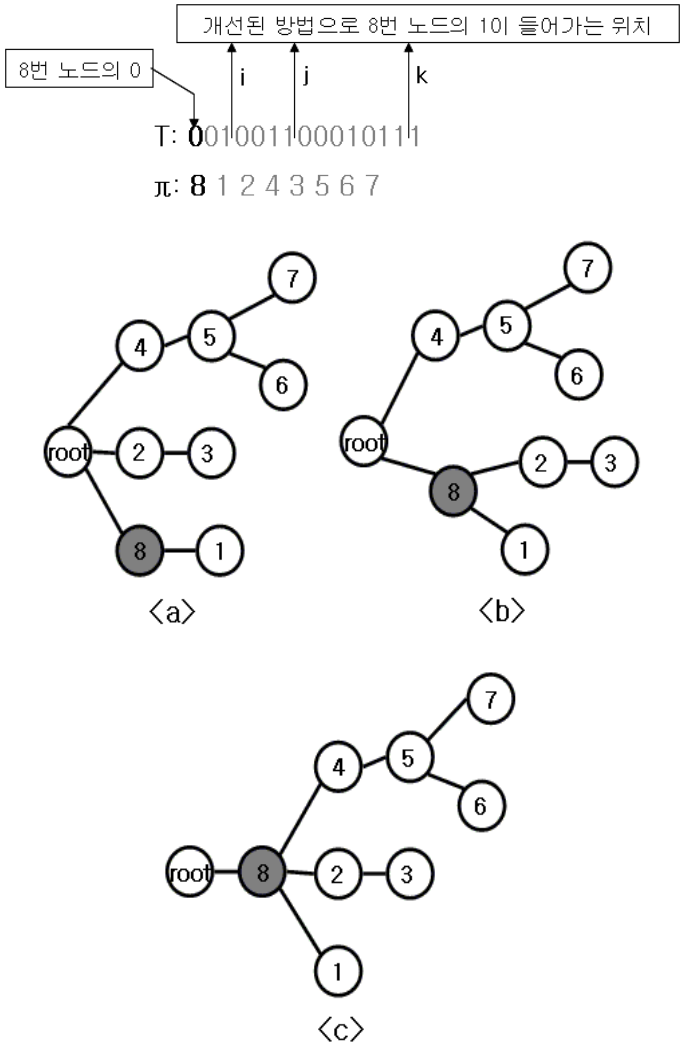


그림 5. 8번 노드가 단말이 아닌 경우

4. 실험 결과

제시된 알고리즘은 C로 구현되어 MS Window XP SP2, AMD 2800+(2.0Ghz) 1GB의 시스템에서 실험하였다. 플로어플랜의 목적함수는 배선 길이를 제외하고 회로의 면적에 대해서만 고려하였다.

MCNC 밴치마크 회로의 각각에 대해 100번의 랜덤한 초기 배치에 대해 기존의 방법과 본 논문에서 제안된 방법을 실험했다. 표1의 실험결과가 보여 주듯이 기존의 방법에 비해 본 논문에서 제안된 방법은 전체적으로

3%정도 개선되었다. hp를 제외한 나머지 회로에 대해 최대면적, 최소면적, 평균면적은 본 논문에서 제안된 방법이 기존의 방법보다 좋은 결과를 보였다.

블록의 수가 많은 회로 일수록 기존의 방법과 본 논문에서 제안된 방법 사이의 실행시간 차이가 커지는데 이러한 이유는 결정적 알고리즘에서 탐색하는 해공간의 차이로 인해 발생되기 때문이다. 결정적 알고리즘만 보았을 때 기존의 방법은 해공간이 n(2n+1)이지만 본 논문에서 제안된 기법은 최악의 경우 해공간이 $n^2(2n+1)/2$가 된다.

표 1. 실험 결과

회로	적용 방법	면적(mm^2)			평균 개선율(%)	평균 시간(초)
		최대	최소	평균		
apte	기존 방법	51.69	47.31	49.19	1.63	0.089
	개선 방법	50.82	47.30	48.39		0.309
xerox	기존 방법	22.46	20.31	21.43	2.42	0.114
	개선 방법	21.88	19.91	20.91		0.453
hp	기존 방법	13.78	9.20	11.60	3.62	0.140
	개선 방법	13.02	9.22	11.18		0.576
ami33	기존 방법	1.36	1.29	1.33	3.88	1.894
	개선 방법	1.32	1.24	1.28		19.686
ami49	기존 방법	41.46	39.13	40.54	3.54	5.799
	개선 방법	40.25	38.07	39.10		87.069

5. 결 론

본 논문에서 제안된 방법은 기존의 방법에 비해 회로 면적의 최적화 성능이 뛰어나다. 대부분의 데이터에서 본 논문에서 제안된 방법은 기존의 방법 보다 좋은 결과를 보여주고 있다. 특히 블록의 개수가 많은 회로의 경우 기존의 방법에서 많은 경우를 간과 하는데 비해 본 논문에서 제시한 방법은 그런 경우를 모두 고려하기 때문에 해의 질이 높아질 가능성이 커지는 장점이 있다.

본 논문에서 제안된 방법의 단점인 CPU 시간을 많이 사용하는 문제를 ENPA[2]에서 사용했던 기법을 응용할 수 있다면 연산시간을 줄일 수 있을 것으로 보인다.

6. 참고문헌

[1] P. -N. Guo, C. -K. Cheng, T. Yoshimura, "An O-tree representation of non-slicing floorplan and its applications," Proc. 36th ACM/IEEE Design Automation Conf. ,pp. 268-273,June 1999.

[2] Y. Pang, C.K. Cheng T. Yoshimura, "An Enhanced Perturbing algorithm for Floorplan DesignUsing the O-tree Representation", ACM Proc. ISPD, pp. 168-173, 2000

[3] X. Hong et al. Corner block list: An effective and efficient topological representation of non-slicing floorplan, Proc. ICCAD, pp.8-12, 2000.

[4] H. Murata and E. Kuh, Sequence-pair based placement method for hard/soft/pre-placed modules, Proc. ISPD, pp.167-172, 1998.

[5] S. Nakatake, H. Murata, K. Fujiyoshi, Y. Kajitani, "Module placement on BSG-structure and IC layout application" in: Proc. of International Conference on Computer Aided Design, 484-490, 1996.