Sequence-Pair 모델 기반의 블록 배치에서 압축과 배치 역변환을 이용한 Simulated-Annealing 개선 기법

성영태○ 허성우
동아대학교
saint@donga.ac.kr○, swhur@dau.ac.kr

Simulated-Annealing Improvement Technique Using Compaction and Reverse Algorithm for Floorplanning with Sequence-Pair Model

YoungTae Seong○ SungWoo Hur
University Dong-A

요 약

Sequence-Pair(SP)는 플로어플랜을 표현하는 모델 중 하나로써, 일반적으로 SP 모델을 사용하는 플로어프래너(floorplanner)는 Simulated-Annealing(SA) 알고리즘을 통해 해 탐색 과정을 수행한다. SP 모델을 이용한 다양한 논문에서 플로어플랜 성능 향상을 위해 평가함수의 개선과 스케줄링 기법 향상을 모색하였으며, 평가함수의 경우 *O(nlogn)* 시간 알고리즘이 존재한다. 본 논문에서는 SP 모델을 이용한 SA 기법에서 SA의 해 탐색 과정 중 초기 해 탐색 시점에서 좋은 해를 빠르게 찾을 수 있는 방법을 제안한다. 제안 기법은 기존의 SA 프레임웍을 수정한 2단계 SA 알고리즘으로써 SP에 대응하는 배치를 압축하고 압축한 배치를 역변환하는 과정으로 구성된다. 실험과 결과를 통해 제안기법의 효과를 보이며, 평균적으로 동일한 SA 환경 하에서 제안기법이 최종결과 면에서 우수함을 보인다.

1. 서 론

VLSI 설계 과정에서 중요한 단계 중의 하나가 플로어플랜(floorplan) 설계 단계이다. 플로어플랜은 평면상에서 직사각형 모듈들을 중첩되지 않게 배치하면서 원하는 목적함수를 최적화 시키는 것을 목표로 한다. 목적함수로는 플로어플랜의 총 면적, 배선길이 또는 면적과 배선길이에 가중치를 부여한 합 등이다.

플로어플랜 문제에서 블록간의 중첩을 가지지 않는 새로운 플로어플랜을 표현하기 위해서 다양한 모델이 제시되었다. 특히 비분할 구조를 표현 할 수 있는 다양한 모델이 제시되었는데 대표적으로 sequence-pair를 이용한 것[1,2,3], BSG(Bounded-Sliceline Grid) 구조를 이용한 것[4], O-tree를 이용한 것[5], CBL(Corner Block List)을 이용한 기법[6] 등이 있다.

Sequence-pair 모델은 n개의 모듈의 위상적 관계를 나타내기 위해 두 순열을 이용하기 때문에 해 공간의 크기는 $O((n!)^2)$이 되며, 한 쌍의 sequence-pair로부터 대응하는 배치를 얻는 시간은 $O(n^2)$이 걸린다[1]. BSG 구조에서는 $n \times n$ 격자(grid)를 이용하여 모듈을 배치하며, 배치를 얻는데 걸리는 시간은 $O(n^2)$이고, 해 공간의 크기는 $O(n!C(n^2,n))$이다. O-tree 모델에서는 해 공간이 $O(n!2^{2n-2}/n^{1.5})$로써 상대적으로 적으며, 한 트리로부터 배치를 얻는 시간은 $O(n)$이다. CBL 모델을 사용하는 기법의 해 공간의 크기는 $O(n!2^{3n-3}/n^{1.5})$이며 주어진 CBL로부터 배치를 구하는데 필요한 시간은 $O(n)$이다.

대부분의 플로어플랜 기법들은 Simulated-Annealing(SA) 알고리즘에 기반한 탐색(search) 구조를 가지며, 수행시간의 대부분을 현재 플로어플랜으로부터 새로운 플로어플랜(neighbor solution)을 찾고 후보 플로어플랜을 평가(evaluation)하는데 사용한다. 따라서 좋은 평가값을 갖는 플로어플랜을 빠른 시간에 찾기 위해서는 플로어플랜을 빠르게 수정하는 기법과 평가함수의 성능을 개선하는 것이 필요하다. Tang과 Wong은 [2,3]에서 Murata[1]가 제안한 $O(n^2)$ 평가함수를 $O(nlogn)$으로 개선시켰으며, Pang[7] 등은 O-tree 모델의 변형 함수를 개선하였다.

본 논문에서는 SA알고리즘을 사용하는 sequence-pair 기반의 개선된 플로어플랜 기법을 제안한다. 제안기법은 기존[2]의 $O(n^2)$ 평가함수와 변형함수(perturb function)를 그대로 사용하는 대신 SA 프레임웍을 수정함으로써 해 탐색 능력을 개선한다.

본 논문의 구성은 다음과 같다. 2장에서 SP모델에 대한 일반적인 내용을 소개하고, 3장에서 제안기법을 설명한다. 4장에서는 제안기법의 효과를 실험을 통해 보이며 5장에서 결론 및 향후과제를 말한다.

2. Sequence-Pair 기반의 Simulated Annealing 기법

이 장에서는 sequence-pair모델을 소개하고 기존의 SA 알고리즘을 이용한 해 탐색 기법에 관해 설명한다.

2.1 플로어플랜 표현을 위한 sequence-pair 모델

Murata[1]가 제안한 sequence-pair는 n개 블록들의 상대적인 위치 관계를 n개 블록에 대한 순열(permutation)의 쌍으로 표현한다. 주어진 sequence-pair (Γ+, Γ-)에 대하여 (Γ+ = <.., x_i, .., x_j, ..>, Γ- = <.., x_i, .., x_j, ..>) 일 때, x_i는 x_j의 왼쪽에 위치하며 (Γ+ = <.., x_j, .., x_i, ..>, Γ- = <.., x_i, .., x_j, ..>) 일 때, x_i는 x_j의 아래쪽에 위치한다. 이 때, x_i, x_j $\in$ *Block*, $1 \le i, j \le n$ 이다. 따라서 두 순열은 블록들의 수평, 수직 관계를 보여주며, 이 관계는 수평 제약 그래프 $G_h(V, E)$와 수직 제약 그래프 $G_v(V, E)$로 다음 과 같은 방법으로 구성 할 수 있다.

(a) $V = \{s_h\} \cup \{t_h\} \cup \{v_i \mid i = 1, .., n\}$,

(b) $E = \{(s_h, v_i) \mid i = 1, .., n\} \cup \{(v_i, t_h) \mid i = 1, .., n\}$
$\cup \{(v_i, v_j) \mid$ block i is left to block $j\}$,

(c) Vertex weight = *width* of block i for v_i, but 0 for s_h and t_h.

수직 제약 그래프도 수평 제약 그래프와 유사한 방법으로 구성 한다. 두 그래프 $G_h(V, E)$, $G_v(V, E)$는 각 정점이 가중치를 가지며 방향성이고 또한 사이클이 없는 DAG(Directed Acyclic Graph)이다. Sequence-pair 모델에서는 각 블록의 위치 좌표 (x, y)를 각각 $G_h(V, E)$, $G_v(V, E)$에서 최장 경로(longest path)를 구함으로써 결정할 수 있다. 이때 블록의 좌표 (x, y)는 그 블록의 왼쪽 아래 모서리 위치이다. $G_h(V, E)$, $G_v(V, E)$를 구성하는데 $\Theta(n^2)$ 시간을 소요하고 두 그래프로부터 최장경로를 구하는데 $\Theta(n+m)$, n과 m은 각각 정점과 간선의 수, 시간을 소요하므로 임의의 sequence-pair로부터 이에 대응하는 배치로 변환하는데 총 $\Theta(n^2)$ 시간을 가진다.

2.2 Simulated-Annealing알고리즘을 이용한 해 탐색

기존의 sequence-pair 모델을 이용한 SA 탐색 알고리즘은 그림1과 같다. SA 알고리즘은 입력으로 초기 플로어프랜 SP_0, 초기 온도 T_0, α, β, N_0, N_{max}를 받는다. 초기 플로어프랜을 위한 SP_0는 랜덤 알고리즘을 통해 임의로 생성하며, 나머지 값들은 SA의 스케줄링에 사용된다. 그림의 Metropolis 알고리즘 중 3번째 줄은 현재 해에서 다음 해를 구하는 변형(perturbing) 과정이다. 일반적으로 변형 과정은 sequence-pair의 한 sequence에서 임의의 두 모듈을 교환하거나, 교환하면서 각 모듈의 회전을 고려하여 그 중 가장 평가값이 좋은 경우를 채택한다.

sequence-pair 모델을 이용하는 대부분의 SA알고리즘은 Cost 함수 또는 Neighbor 함수의 개선에 집중하였다. SA 알고리즘의 개선을 위한 노력으로는 스케줄링 개선 기법 등이 있다.

```
Algorithm Simulated-Annealing
1. INPUT: SP0, T0, α, β, N0, Nmax
2. OUTPUT: A solution SP
3. Begin
4.      set N := N0, SP := SP0, T := T0; Ntotal :=0.
5.      while ((T>0.0)  and (Ntotal <= Nmax)) do {
6.           Call Metropolis(SP, T, N);
7.           Ntotal := Ntotal + N;
8.           T := T * α;
9.           N := N * β;
10.     }
11.     return SP;
12. End
```

```
Algorithm Metropolis(SP, T, N)
1. Begin
2.      while (N > 0) do {
3.           SPnew := Neighbor(SP);
4.           △h := Cost(SPnew) - Cost(SP);
5.           if ((△h < 0) or (random < e^(-△h/T))) then
6.                SP := SPnew;
7.                N := N - 1;
8.      }
8. End
```

그림 1. 기존의 Sequence-Pair 기반의 SA 알고리즘

3. 제안 기법

본 논문에서는 기존의 SA 알고리즘의 초기 반복에서 기존의 알고리즘보다 빠른 시점에 좋은 해를 구할 수 있는 기법을 제안한다. 이 장에서는 제안 기법의 전체 개요와 함께 두 가지 주요 알고리즘 Compact와 Reverse를 소개하고 SA의 초기 반복에서 Compact와 Reverse 알고리즘의 효과에 대해 설명한다.

3.1 개요

본 논문에서 제안한 기법은 기존의 [2]에 제시된 평가함수와 변형 함수를 그대로 사용하는 대신 SA 프레임웍을 그림2와 같이 수정한 것이다. 그림2에서 Simulated_Annealing 알고리즘은 기존의 Metropolis 알고리즘을 사용하지 않고 수정된 Metropolis_proposed를 사용하며 그 외 코드는 동일하다.

```
Algorithm Metropolis_Proposed(SP, T, N)
1. Begin
2.      while (N > 0) do {
3.           SP_new := Neighbor(SP);
4.           if (CPR <= Area_total/Cost(SP_new)) {
5.                Pack_SP_new := Compact(Pack_SP_new);
6.                SP_new := Reverse(Pack_SP_new);
7.           }
8.           Δh := Cost(SP_new) - Cost(SP);
9.           if ((Δh < 0) or (random < e^(-Δh/T))) then
10.               SP := SP_new;
11.               N := N - 1;
12.     }
13. End
```

그림 2. 2단계 SA 탐색 메커니즘을 가지는 수정된Metropolis 알고리즘

제안 기법은 그림2에서 보이는 바와 같이 기존 알고리즘(그림1의 Metropolis)에 조건부 루틴(4번째 줄~7번째 줄)을 추가하였다. 조건부 루틴을 수행하기 위한 조건은 $CPR \leq Area_{total}/Cost(SP_{new})$ 이며, 이때 CPR(Completion Ratio)은 현재 해의 평가값에 대한 블록의 총 면적 비율이다. 예를 들어, 블록의 총 면적이 120일 때, Neighbor 함수로 생성된 SP_{new} 의 평가값이 125 라면 CPR=0.96 이다. 이것은 현재 해의 배치 완성도가 96%임을 의미한다. 따라서 수정된 Metropolis_proposed 알고리즘은 SA과정 중 해의 배치 완성도가 CPR 값 이하일 때 추가된 조건부 루틴을 수행하고 그 이상일 때 기존의Metropolis 알고리즘과 동일한 방법으로 동작한다.

조건부 루틴은 Compact와 Reverse 알고리즘으로 구성된다. Compact 알고리즘은 Neighbor 함수로 생성된 SP_{new} 에 대응하는 배치의 모든 블록이 각각 왼쪽과 아래쪽 방향으로 최소한 하나 이상의 다른 블록 또는 왼쪽 아래 영역 경계선과 인접하도록 배치의 위상을 변형하는 과정이다. Compact 과정에서 생성된 새로운 배치를 $Pack\_SP_{new}$ 라 할때, Reverse 알고리즘은 $Pack_SP_{new}$로부터 이에 대응하는 새로운 SP_{new}를 생성하는 과정이다.

Compact와 Reverse 알고리즘은 다음과 같은 특징을 가진다.

SP_{new} : Neighbor(SP)의 결과
$CompSP_{new}$: Compact(SP_{new})의 결과
$RevSP_{new}$: Reverse(CompSP_{new})의 결과

라 할 때,

$$Cost(RevSP_{new}) = Cost(CompSP_{new}) \leq Cost(SP_{new}).$$

먼저, $Cost(CompSP_{new}) \leq Cost(SP_{new})$ 식은 명백하다. SP_{new} 의 배치에서 모든 블록에 대해 왼쪽과 아래쪽으로 최소한 하나 이상의 인접한 블록 또는 왼쪽 아래 영역 경계선을 갖도록 위상을 변경하는 것은 인접하지 않은 블록이 갖는 white-space를 제거하는 과정이다. 이것을 그림3의 예로 보인다.

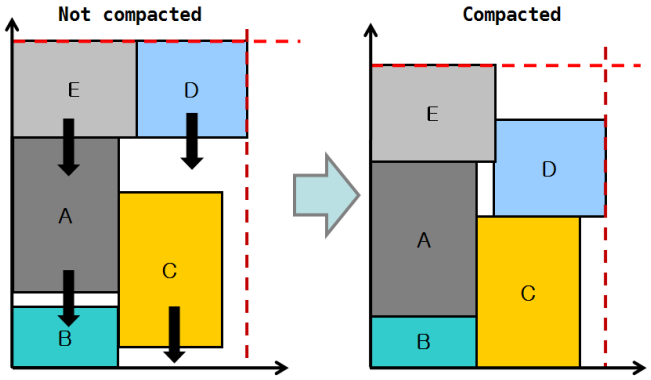

그림 3. 아래 쪽 Compact 과정의 예

그림3에서 왼쪽 배치의 블록 *A, C, D, E*(*A*블록의 이동결과 아래쪽으로 이동가능)는 아래쪽 방향으로 인접한 블록을 가지지 않는다. 따라서 아래쪽 방향으로 압축이 가능하며 왼쪽 그림이 아래쪽 방향 압축 결과이다.

두 번째, $Cost(RevSP_{new}) = Cost(CompSP_{new})$ 식은 Reverse 알고리즘이 비용의 손실 없이 Metropolis에서 이웃해를 구하는 또 다른 방법임을 말한다. Reverse 알고리즘은 3.3절에서 설명한다.

조건부 루틴의 목표는 가능한 빠른 시간에 CPR값에 대응하는 SP_{new} 를 찾기 위함이다. 이것은 SA 알고리즘의 특성상 초기 반복에 소요되는 시간 보다 최종해를 찾는데 걸리는 시간이 월등히 많으므로 초기 빠른 반복 시점에 CPR값에 해당하는 해를 찾고 최종해를 찾는데 시간을 더 사용하도록 유도하기 위함이다.

3.2 Compact 알고리즘

압축 알고리즘의 개요를 그림3에 기술하였다.

```
Algorithm Compact(Packing)
1. Begin
2.      Packing_comp1 := LeftFirstDownSecond(Packing);
3.      Packing_comp2 := DownFirstLeftSecond(Packing);
4.      if(Cost(Packing_comp1) < Cost(Packing_comp2))
5.           return Packing_comp1;
6.      else
7.           return Packing_comp2;
```

그림 3. Compact 알고리즘의 개요

그림3에서 함수 LeftFirstDownSecond는 주어진 배치의 각 블록에 대해 왼쪽 방향으로 최소한 하나의 인접한 블록을 가지도록 압축하고 아래쪽으로 다시 압축하며 함수

DownFirstLeftSecond는 아래쪽 방향으로 먼저 압축한 후 왼쪽 압축을 수행한다. 이 두 배치 중 평가값이 더 좋은 배치를 반환한다. 압축 알고리즘은 [5]에 소개된 윤곽선(contour-map) 기법을 이용하여 구현하였으며, *O(nlogn)* 시간에 수행된다.

3.3 Reverse 알고리즘

함수 Reverse는 주어진 Packing으로부터 이에 대응하는 sequence-pair를 찾는 알고리즘이다. Reverse 알고리즘의 목표는 크게 두 가지로 요약된다.

(1) Cost($RevSP_{new}$) = Cost($CompSP_{new}$).
(2) (1)조건을 만족하면서 $SP_{prev} \approx RevSP_{new}$, 이때 SP_{prev}는 Compact 이전의 Neighbor 함수의 결과.

목표(1)은 $CompSP_{new}$의 평가값과 $RevSP_{new}$의 평가값이 같아야 함을 말하는데 그렇지 않을 경우 잘못된 Reverse 알고리즘 일 뿐만 아니라 Cost($RevSP_{new}$) > Cost($CompSP_{new}$) 일 경우, 이 후의 SA 탐색 과정에서 수락(accept)되는 해가 없을 수 있기 때문이다. Sequence-pair 모델의 단점으로 주어진 하나의 배치에 대해 이에 대응하는 sequence-pair가 2개 이상 존재한다는 것인데, 목표(2)는 SA의 해 탐색 과정에서 이전해와 다음 해 사이의 연관성을 보장하기 위함이다. 실험을 통한 경험에서 압축이 어느 정도 진행된 경우 [1]에서 제안한 Gridding 알고리즘을 사용하면 목표(2)는 보장됨을 알 수 있었다. 목표(2)와 관련하여 또 다른 특징은 $RevSP_{new}$가 SP_{prev}보다 더 나은 평가값을 갖는 변형된 형태의 sequence-pair 란 사실이다. 이것은 Neighbor 함수와 더불어 Reverse 함수 또한 하나의 변형(perturb) 함수로 볼 수 있음을 말한다.
Reverse 알고리즘을 그림5에 기술하였고, 그림4에서 sequence-pair의 Γ+를 구하는 과정을 예로 보인다.

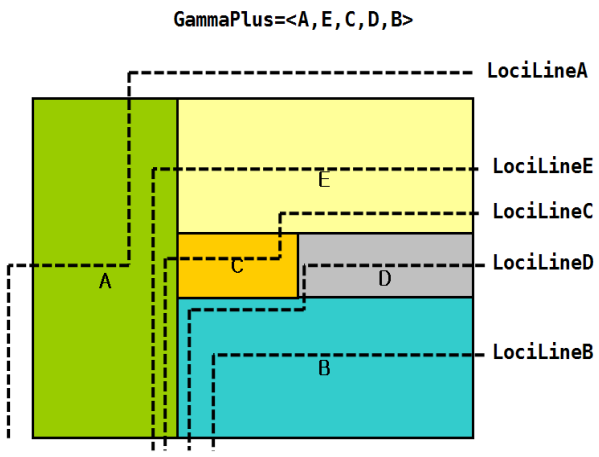

그림 4. LociLine을 이용한 Γ+를 구하는 예

Γ+와 Γ-는 각각 [1]에서 제안한 Gridding 알고리즘을 사용한다. 그림4에서 점선은 LociLine을 의미하고 Gridding 기법을 통해 각 블록의 LociLine을 구한 후 이 선이 포함하는 면적의 크기 순서대로 블록을 정렬시킨다. 그림4에서 블록 A의 LociLine 면적이 가장 크므로 Γ+에서 가장 처음에 위치하고 블록 B의 면적이 가장 작으므로 맨 마지막에 위치한다. Γ-를 구하는 방법도 이와 유사하다.

```
Algorithm Reverse(Packing)
1. INPUT: Packing
2. OUTPUT: SP (consists of SP_GPlus and SP_GMinus)
3. Begin
4.      //find GammaPlus from the given packing
5.      for each b_i ∈ Packing: 0 ≤ i < n {
6.        //b_i ∈ Packing.
7.       GPlus_LociArea_i.area := Eval_GPLociLine_Area(b_i);
8.        GPlus_LociArea_i.id := index of b_i;
9.      }
8.      sort array GPlus_LociArea by decreasing order
9.      for each SP_GPlus_j: 0 ≤ j < n {
10.       SP_GPlus_j := GPlus_LociArea_j.id;
11.     }
12.    /* GammaMinus can be found from similar fashion of
          the GammaPlus routine */
13.     return SP;
14. End
```

그림 5. 주어진 packing으로부터 sequence-pair를 구하는 Reverse 알고리즘

4. 실험 및 결과

제안 기법의 실험은 WindowsXP/AMD 1.8G 시스템 상에서 수행 하였으며, C 언어로 구현하고 minw-gcc-3.4.2로 컴파일 하였다. 실험에 사용한 회로는 표준 벤치마크 회로 apte, xerox, hp, ami33, ami49 5개를 사용하였으며 각각은 회로의 수가 9, 10, 11, 33, 49개인 hard-block이다.
각 회로에 대한 실험에서 같은 회로의 경우 동일한SA 스케줄링 환경(동일한 반복수)에서 수행하였으며, 회로의 특성마다 스케줄링 환경을 달리하였다. 예를 들어, hp 회로의 경우 높은 온도에서 균등한 cooling 비율에 대해 동일한 반복회수를 주었고, ami49의 경우 낮은 온도에서 서서히 cooling을 진행하며 높은 비율로 반복회수를 늘렸다.

4.1 각 회로에 대한 제안 기법의 효과

이 절에서는 각 회로에 대해 제안기법의 효과를 그래프로 요약하고 그 특징을 살펴본다.

hp 회로의 경우 제안 기법으로 수행한 결과 제안기법의 평가값이 9.34인 반복 시점에서 기존기법은 10.92값을 가지고 있었다. 그림의 경우 기존기법에서는 이후의 반복에서 제안 기법보다 더 좋은 결과를 얻을 수 없었다.

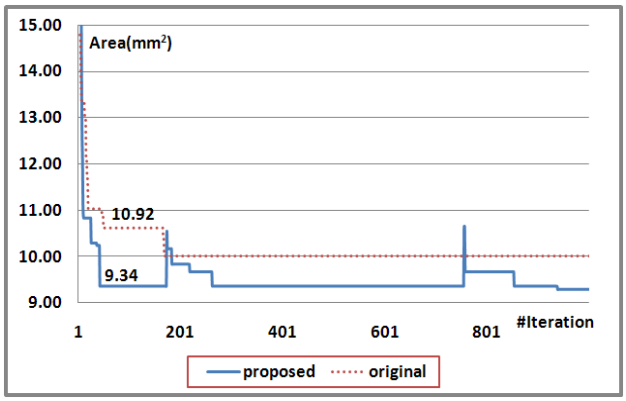

그림 6. hp 회로에 대한 실행 그래프

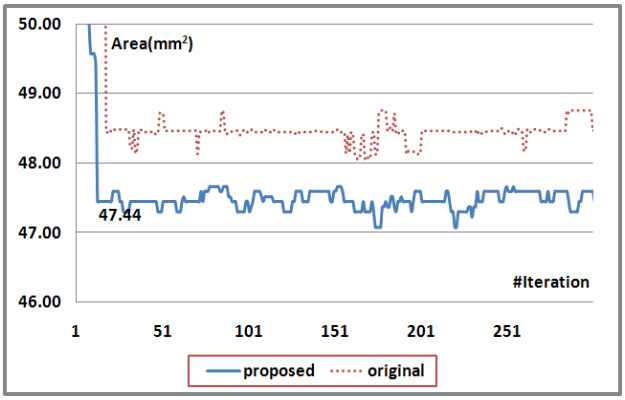

그림 7. apte 회로에 대한 실행 그래프

apte 회로의 경우 그림7에서 보이는 결과값 47.44를 제안기법으로 찾은 반면 기존기법은 동일 반복 시점에 비교할 만한 해를 찾지 못함을 보여준다.

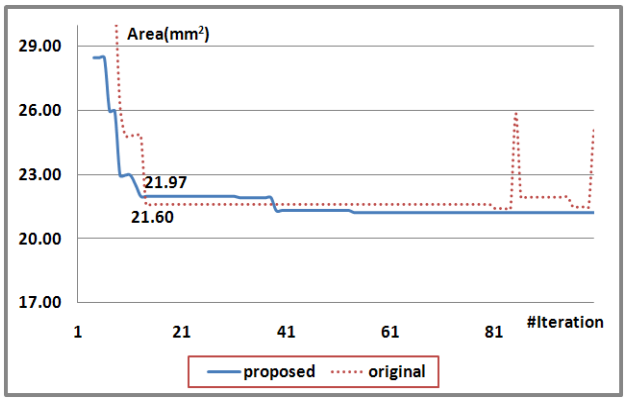

그림 8. xerox 회로에 대한 실행 그래프

xerox 회로의 경우 예외적으로 기존기법에서21.60 결과값을 찾는 동안 제안기법에서는 21.97결과값을 보여주고 있다. 이것은 제안기법에서 효과적으로 *CPR*값에 도달하도록 유도하는 알고리즘이 아닌 단순히sequence-pair의 순서쌍 교환을 통해 이웃해를 구하기 때문으로 분석된다.

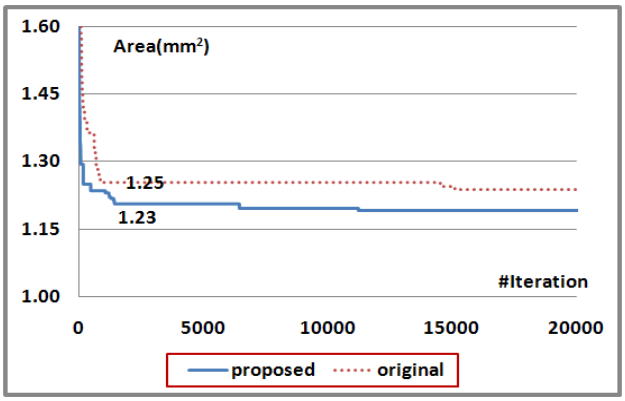

그림 9. ami33 회로에 대한 실행 그래프

그림9에서 보이는 바와 같이 제안기법의 경우 특정 반복시점에 결과값 1.23을 찾는 동안 기존기법은 결과값 1.25 상태에 있다.

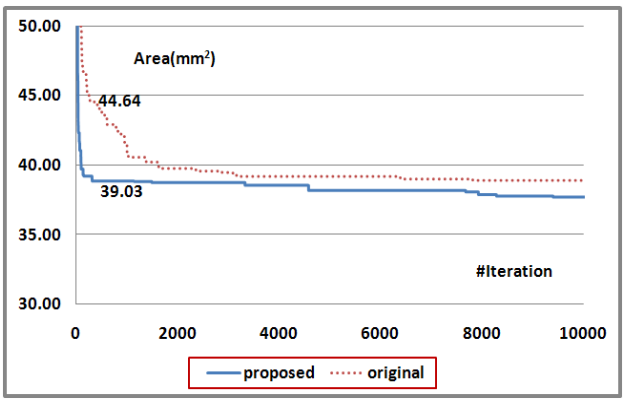

그림 10. ami49 회로에 대한 실행 그래프

회로 ami49는 벤치마크 회로 중 가장 크기가 큰 회로이다. ami49의 경우 일반적으로 다른 회로에서 보이는 그래프 패턴과 비슷한 면을 보인다 SA 초기 반복 시점에서 제안기법의 경우 39.03의 해를 찾는데 반해 기존기법의 경우 동일 반복 시점에서 44.64의 해를 찾고 있음을 알 수 있다. 이 점을 통해 남은 SA의 탐색 시간 동안 제안기법이 SA의 수렴 해를 찾는데 있어 초기 반복에서 시간을 낭비하지 않음을 알 수 있다

3.3 결과 비교

표1, 2, 3에 각각 최종 결과에 대한 worst/best/average 결과 값과 수행 시간을 요약하였다 실험에 사용된 CPR 값은 모두 0.90으로써 초기 반복 중 90%의 배치 완성도를 가지기 전 까지 모두 Compact와 Reverse를 사용하였다. 그리고 표에 기술된 시간(sec)은 해당 결과값을 찾는데 까지 걸린 총 시간을 말한다

표 1. Worst-case 결과 비교 (*CPR=0.90*)

Circuit	Original		Proposed	
	area (mm^2)	time (sec)	area (mm^2)	time (sec)
apte	48.05	1	48.05	1
xerox	20.42	8	20.47	3
hp	9.52	5	9.27	1
ami33	1.24	53	1.21	8
ami49	38.55	127	37.57	209

Worst-case 결과 비교에서 xerox와 hp 그리고 ami33 회로의 경우 기존기법의 해 보다 좋은 결과를 더 빠른 시간에 찾았음을 알 수 있다.

표 2. Best-case 결과 비교 (*CPR=0.90*)

Circuit	Original		Proposed	
	area (mm^2)	time (sec)	area (mm^2)	time (sec)
apte	47.07	1	47.07	1
xerox	19.86	17	19.83	14
hp	9.14	5	9.11	10
ami33	1.21	13	1.19	16
ami49	36.81	76	36.75	148

Best-case 결과 비교에서 최종 결과값의 경우 제안 기법에서 우수함을 보였고 수행시간 측면에서 제안기법이 더 많은 시간을 소요했음을 알 수 있다. 이것은 일반적인 형태로써 제안기법이 기존기법에 비해SA 알고리즘 내부에 두 가지 알고리즘을 더 사용하기 때문이다.그러나 동일한 SA 환경(같은 수의 반복)에서 제안기법이 더 좋은 최종 결과를 보여준다.

표 3. Average-case 결과 비교 (*CPR=0.90*)

Circuit	Original		Proposed	
	area (mm^2)	time (sec)	area (mm^2)	time (sec)
apte	47.68	1	47.32	1
xerox	20.23	11	20.12	10
hp	9.25	9	9.20	7
ami33	1.29	26	1.20	49
ami49	37.46	74	36.99	114

표 3에서 보이는 바와 같이 평균적으로 제안기법이 기존기법과 비교해서 수행시간을 많이 필요로 하는 반면 최종 결과에서 우수함을 보여준다. 따라서 제안기법에서 사용하는 Compact와 Reverse 알고리즘을 개선한다면 제안기법의 SA 알고리즘이 더 좋은 수행시간 개선을 보일 것으로 예상한다.

5. 결론 및 향후 과제

본 논문에서는 sequence-pair 모델을 기반으로 하는 simulated-annealing 프레임웍에서 동일한 SA 환경이 주어졌을 때, 빠른 반복 시점에서 더 좋은 배치를 찾도록 simulated-annealing기법을 2단계 탐색 과정으로 변형하였다. 기존의 SA 알고리즘보다 더 많은 처리 루틴을 사용하며, 비효율적인 Reverse 알고리즘을 사용한다 하더라도 결과를 비교해 볼 때, 비교할 만한 시간에 더 좋은 해를 구할 수 있음을 알 수 있다.

향후과제로 제안기법에서 사용하는 Reverse 알고리즘의 계산 속도를 개선하고 Tang[3]이 제안한 기존의 *O(nlogn)* 평가 알고리즘을 도입하는 것이다.

참고문헌

[1] H.Murata, K.Fujiyoshi, S.Nakatake, and Y.Kajitani, "VLSI module placement based on rectangular-packing by the sequence-pair", IEEE Trans. on CAD of Computer-Aided Design of Integrated Circuits and Systems, vol.15:12, pp. 1518-1524, 1996.

[2] X.Tang, R.Tian, and D.F.Wong, "Fast evaluation of sequence pair in block placement by longest common subsequence computation", DATE-2000, pp. 106-111, 2000.

[3] X.Tang and D.F.Wong, "FAST-SP: a fast algorithm for block placement based on sequence pair", Proc. of the 2001 conference on ASP-DAC, pp. 521-526, 2001.

[4] S.Nakatake, K.Fujiyoshi, H.Murata and Y.Kajitani, "Module Placement on BSG-Structure and IC Layout Applications," Proc. of ICCAD, 1996.

[5] P.Guo, C.Cheng and T.Yoshimura, "An O-Tree Representation of Non-Slicing Floorplan and Its Applications," Proc. of DAC, pp.268-273, 1999.

[6] X.Hong, G.Huang, Y.Cai, J.Gu, S.Dong, C.Cheng and Jun Gu, "Corner Block List: An Effective and Efficient Topological Representation of Non-Slicing Floorplan," Proc. of ICCAD, pp.8-12, 2000.

[7] Y.Pang, C.K.Cheng, and T.Yoshimura, "An Enhanced Perturbing Algorithm for Floorplan Design Using the O-tree Representation", ISPD 2000, pp. 168-173, 2000.