고정블록을 포함한 CBL 기반 평면계획
(Floorplanning with Obstacles(Preplaced Block) based on CBL)

강 상 구 † 임 종 석 ††
(Sang Ku Kang) (Chong Suck Rim)

요 약 본 논문에서는 고정 블록이 포함된 회로를 대상으로 하는 새로운 CBL 기반 평면계획 방법을 제안한다. 기존의 CBL 기반의 고정블록 평면계획의 문제점을 파악하고 이를 개선하였다. 제안한 방법은 자유 블록 만으로 구성된 CBL에 고정블록을 삽입하여 원래 CBL과 그 위상이 유사하고 패킹이 가능한 새로운 CBL을 만드는 방법으로 이를 시뮬레이티드 어닐링에 적용하여 평면계획을 수행한다. 실험결과는 우리가 제안한 평면계획 방법이 고정블록을 효과적이고 효율적으로 배치할 수 있음을 보여준다.

키워드 : 평면계획, 코너 블록 리스트, 고정블록

Abstract In this paper we propose a new CBL-based floorplan method that accommodates pre-placed blocks. We identify the problem of the previous CBL-based pre-placed block floorplan method, and suggest the solution method of this problem. In our method, CBLs consisting of only free blocks are perturbed and maintained during the simulated annealing. Pre-placed blocks are inserted during packing in such a way that the topology of the CBL after insertion of a pre-placed block resembles the topology before insertion. Thus, even with the inclusion of pre-placed blocks, the searching effort via simulated annealing yields acceptable results. Experimental results show that our floorplan method places pre-placed blocks effectively and efficiently.

Key words : floorplan, corner block list, pre-placed block

1. 서 론

평면계획은 backend 공정의 첫 단계로서 각 기능 블록(functional block)의 방향과 위치를 결정한다[1]. 평면계획 방법은 기본적으로 면적을 최소화해야 함은 물론이고 설계자의 다양한 제약조건을 만족시켜 더 많은 디자인 유연성을 제공하여야 한다.

현대 SoC 시스템은 아날로그 블록, 메모리 블록 등의 다양한 기능 블록들을 포함하고 있다. 이러한 블록들의 배치가 시스템 성능에 큰 영향을 미치므로 위상 제약조건들은 점차 복잡해지고 늘어나는 추세에 있다. 평면계획에의 위상적 제약조건은 블록의 위치를 미리 지정하는 고정블록 제약조건(preplaced-block constraints)[2-5], 블록이 칩 경계의 특정 변에 인접하도록 하는 경계 제약조건(boundary constraints)[6,7], 특정 블록들이 서로 인접하게 놓이도록 하는 인접 제약조건(abutment constraints)[8-10], 특정 블록들을 특정 칩의 특정 영역 내에 배치하도록 하는 범위 제약조건(range constraints)[11], 특정 블록들을 미리 정해진 수평 또는 수직 선상에 놓이도록 하는 정렬 제약조건(alignment constraints)[12,13] 등이 있다. 이들 중에서 고정블록 제약조건은 특정 블록을 설계자가 원하는 곳에 배치하거나 특정 영역을 봉쇄영역(blockage area)으로 지정하여 블록들이 놓이지 않도록 함으로써 직접적으로 디자인 유연성을 보장한다. [14]에서는 위의 제약 조건들이 복합적으로 적용된 제약조건을 처리할 수 있는 방법이 제안되었다.

인코딩 기반 평면계획 방법은 평면계획을 인코딩하여 코드로 표현하고 시뮬레이티드 어닐링을 통해 그 코드의 상태를 반복적으로 조정하여 최적화된 평면계획을 얻는다[15]. 전형적인 인코딩 기반의 평면계획 방법은 퍼터베이션(perturbation), 패킹(packing), 비용 평가(cost-

† 학생회원 : 서강대학교 컴퓨터공학과
skkang@sogang.ac.kr
†† 종신회원 : 서강대학교 컴퓨터공학과 교수
csrim@sogang.ac.kr
논문접수 : 2006년 5월 16일
심사완료 : 2009년 3월 2일

정보과학회논문지: 시스템 및 이론 제36권 제3호(2009.6)

estimation)으로 이루어 진다[1]. 퍼터베이션은 현재의 인코딩된 해(solution)로부터 새로운 코드를 생성하는 단계이고, 패킹은 코드로 표현된 해로부터 평면계획을 얻는 과정이다. 그리고 비용산출은 평면계획의 면적, 배선거리 등과 같은 비용함수를 계산하는 과정이다.

현재까지 SP[16], BSG[17], CBL[18-20], O-tree[21], B*-tree[2], Q-sequence[22] 등과 같은 다양한 인코딩 방법들이 알려져 있다. 이들 중 CBL은 작은 해공간(solution space)으로 인해 수렴속도가 빠르고, 선형시간 패킹이 가능하며, 효율적이다. CBL을 기반의 다양한 연구들을 통해 여러 제약조건들을 해결할 수 있는 방법들이 제시되었고, 이미 [23]에서 고정블록 제약조건을 처리할 수 있는 방법이 발표되었다. 그러나, Ting-Chi Wang 등이 제안한 두 방법들 중 첫번째 방법은 최적의 배치를 얻을 수 없음이 이미 밝혀져 있고[23], 두번째 방법은 특정한 경우에 패킹이 불가능 하거나 모순되는 경우가 존재한다. 따라서 우리는 이러한 문제점을 해결 할 수 있는 CBL 기반 고정블록 평면계획 방법을 제안한다.

제안한 평면계획 방법에서는 [23]의 첫번째 방법과 같이 자유블록들 만으로 구성된 CBL로 퍼터베이션을 수행하는 시뮬레이티드 어닐링 방법을 사용한다. 고정블록들은 패킹 과정에서 위치에 맞도록 삽입된다. 기존 방법과의 차이점은 불필요한 계산을 피하기 위해 패킹 과정에는 항상 패킹이 가능한 CBL이 입력되도록 하고, 패킹이 가능하지 않으면 바로 다른 이웃해를 생성하기 위해 퍼터베이션 과정을 수행한다. 그리고 패킹과정에서 자유블록이 순차적으로 삽입될 때, 고정블록과 위치가 겹치는지 조사한다. 만약 겹친다면 해당 자유블록의 삽입을 미루고 고정블록을 삽입한다. 고정블록의 삽입은 그 이후 삽입되는 자유블록에 영향을 주므로 CBL의 성분을 조정하여 패킹가능하고 입력된 CBL과 위상(블록간의 상대적 위치)이 유사한 CBL이 되도록 한다. 그림 1은 제안된 평면계획 방법의 흐름도를 보인 것이다.

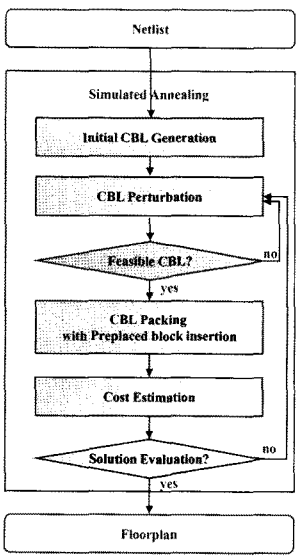


그림 1 Floorplan Flow

본 논문은 서론을 포함하여 5장으로 구성된다. 2장에서는 CBL과 기존 CBL 기반의 고정블록 처리 방법의 문제점에 대해 설명하고, 3장에서는 고정블록 삽입 방법에 대해 기술한다. 4장에서는 실험결과를 보이고 5장에서는 결론과 향후 연구 방향을 제시한다.

2. CBL(Corner Block List)

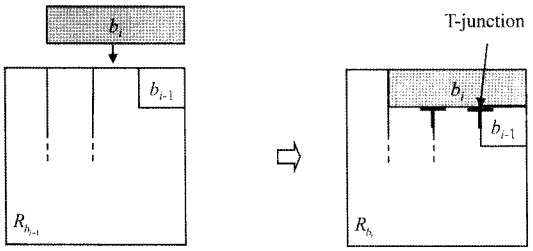
이 장에서는 CBL의 정의, CBL 패킹에 사용되는 두 개의 스택, 스택을 사용한 CBL 패킹 방법, feasible CBL 등에 대해 기술하고, [23]에서 제안한 방법의 문제점을 설명한다.

2.1 CBL의 정의 및 CBL Packing

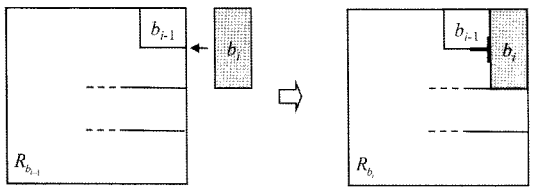
CBL은 S, L, T 리스트로 구성된 튜플이다. CBL은 모자이크 평면계획(mosaic floorplan)이라는 일반적인 평면계획을 모델화한 것으로 하나의 패킹 가능한 CBL로부터 하나의 모자이크 평면계획을 얻을 수 있다. 따라서 CBL를 해공간으로 하는 SA를 통하여 주어진 회로의 평면계획을 얻을 수 있다[18].

주어진 CBL(S,L,T)로부터 평면계획을 구하는 과정을 CBL 패킹이라 한다. $S=(b_1,b_2,...,b_n)$ 리스트는 주어진 회로의 블록 이름 리스트이며, CBL 패킹에서는 초기 빈 공간 R_{b_1}에 b_1부터 차례로 블록을 삽입한다. 이때 R_{b_1}은 b_1자신이 된다. 블록 b_1, b_2, ..., b_{i-1}까지 삽입하면 이들은 $R_{b_{i-1}}$을 구성한다. 그리고, 다음 블록 b_i를 $R_{b_{i-1}}$에 삽입하여 R_{b_i}를 생성한다(그림 2). 이 과정을 모든 블록이 추가될 때까지 반복하면 하나의 평면계획을 얻게 된다.

블록 b_i를 $R_{b_{i-1}}$에 추가하는 방법은 L과 T 리스트에 의해 결정된다. 따라서, $L=(L_{b_1}, L_{b_2}, \cdots, L_{b_n})$, $T=(T_{b_1}, T_{b_2}, \cdots, T_{b_n})$라고 하자. b_i는 $R_{b_{i-1}}$의 위 또는 오른쪽에 삽입되는 형태로 추가되는데 b_i에 대응하는 L_{b_i}가 이 방향을 나타낸다. 즉, $L_{b_i}=0$이면 그림 2(a)와 같이 $R_{b_{i-1}}$의 상변의 오른쪽에 삽입되며, $L_{b_i}=1$이면 그림 2(b)와 같이 $R_{b_{i-1}}$의 우변의 위쪽에 삽입된다. b_i가 삽입될 때 삽입되는 방향으로 $R_{b_{i-1}}$에 속한 블록들과 b_i가 인접하게 되는데 이들로 인하여 'T' 형태의 접합이 생성된다(그림 2). 이러한 접합을 T-접합이라 하고 생성된 T-접합의 수를 T-접합수라고 한다. T 리스트의 T_{b_i}는 b_i가 삽입될 때 생성하여야 할 T-접합 수를 의미한다. 예를 들어 그림 2(a)



(a) Insertion of b_i, where $L_{b_i}=0$, $T_{b_i}=2$



(b) Insertion of b_i, where $L_{b_i}=1$, $T_{b_i}=1$

그림 2 Illustrations for a block insertion in CBL packing

는 $T_{b_i}=2$인 경우 b_i가 삽입되는 모양을 보인 것이고, 그림 2(b)는 $T_{b_i}=1$인 경우 b_i가 삽입되는 모양을 보인 것이다. b_i가 삽입되면 b_i는 이와 T-접합을 이루는 $R_{b_{i-1}}$의 블록들을 덮는 모양을 갖는다. 이때 우리는 "b_i가 이들 블록을 덮는다" 라고 기술한다. b_i가 삽입되면서 덮는 블록 수는 b_i에 의해 생성되는 T-접합 수와 관계가 있으며 그 수는 $T_{b_i}+1$ 이다.

CBL에서 b_1 의 경우, 맨 처음 삽입되는 블록이므로 삽입방향을 정의할 수 없고, 삽입되면서 생성되는 T-접합이 존재하지 않기 때문에 L_{b_1}과 T_{b_1}은 정의되지 않는다.

CBL 패킹에서 삽입되는 블록이 어느 블록을 덮을 것인가를 결정하기 위해서 평면계획 영역의 위쪽 경계와 오른쪽 경계에 인접한 블록들의 상태를 유지해야 한다. 이를 위해 두 개의 스택이 사용된다. 평면계획 영역의 위쪽 경계와 오른쪽 경계에 인접한 영역에 배치된 블록들을 S 리스트 순서로 유지하는 스택을 각각 T-stack, R-stack이라 정의하고, 각각 S^T, S^R로 표기한다. 블록 b_i가 삽입된 후, S^T, S^R의 상태를 각각 $S^T_{b_i}$, $S^R_{b_i}$로 표기하고, 각 스택에 유지되고 있는 블록의 수를 각각 $|S^T_{b_i}|$, $|S^R_{b_i}|$로 표기한다. 그리고, S^T, S^R은 다음과 같은 방법으로 상태를 유지한다.

첫째, 블록 b_1가 삽입될 때, $S^T_{b_1}=[b_1]$, $S^R_{b_1}=[b_1]$이다. 즉, b_1 삽입 이전에 S^T와 S^R은 모두 비어있는 상태이고 b_1이 삽입되면 b_1은 R_{b_1}과 같으므로 R_{b_1}의 상변과 우변에 모두 인접한다. 따라서, S^T와 S^R에 b_1이 각각 추가된다.

Procedure : Stack adjustment
-Input : $b_i, L_{b_i}, T_{b_i}, S^T_{b_{i-1}}$, and $S^R_{b_{i-1}}$
-Output : $S^T_{b_i}$ and $S^R_{b_i}$

1: if ($L_{b_i}=0$) // insertion of b_i onto the top of $R_{b_{i-1}}$
2: delete $T_{b_i}+1$ elements from $S^T_{b_{i-1}}$;
3: else // insertion of b_i onto the right of $R_{b_{i-1}}$
4: delete $T_{b_i}+1$ elements from $S^R_{b_{i-1}}$;
5: insert b_i to both $S^T_{b_{i-1}}$ and $S^R_{b_{i-1}}$ to obtain $S^T_{b_i}$ and $S^R_{b_i}$;

그림 3 Stack adjustment procedure for insertion of b_i onto $R_{b_{i-1}}$

둘째, 블록 b_i가 삽입될 때, $L_{b_i}=0 (L_{b_i}=1)$일 경우, $S^T_{b_i-1}(S^R_{b_{i-1}})$에 대해 삭제를 $T_{b_i}+1$번 수행한 후 b_i를 삽입하고, $S^R_{b_i-1}(S^T_{b_{i-1}})$에 대해 b_i를 삽입함으로써 상태를 유지한다. 즉, $L_{b_i}=0$인 경우, $R_{b_{i-1}}$의 상변에 인접한 블록들 중에서 b_i에 의해 덮이는 블록의 수가 $T_{b_i}+1$이고 이 블록들은 더 이상 R_{b_i}의 상변과 인접하지 않으므로 S^T에서 제거되어야 하고, 대신 b_i가 R_{b_i}의 상변에 인접하게 되므로 b_i를 S^T에 추가해야 한다. 그리고, $R_{b_{i-1}}$의 우변에 인접한 블록들 중에서 b_i에 의해 덮이는 블록이 없기 때문에 S^R에서 제거되는 블록은 없고, 새로 b_i가 R_{b_i}의 우변에 인접하게 되므로 b_i를 S^R에 추가해야 한다. $L_{b_i}=1$의 경우도 유사한 방법으로 설명할 수 있다. 그림 3은 스택 조정을 위한 프로시저를 보인 것이다.

그림 4는 CBL 패킹 과정의 예를 보인 것이다. 그림 4에서, CBL가 주어지면 우선 S 리스트에서 첫 블록 b_1으로 배치영역을 초기화하고, 각 스택에 b_1을 삽입한다 ($S^T_{b_1}=[b_1]$, $S^R_{b_1}=[b_1]$). b_2는 T-접합 없이($T_{b_2}=0$) 위쪽 ($L_{b_2}=0$)에서 블록 b_1을 위에서 덮으며 삽입된다. 따라서, $S^T_{b_1}$에서 블록 b_1가 제거된 다음 b_2가 삽입되고 $S^T_{b_1}$에 b_2가 삽입된다($S^T_{b_2}=[b_2]$, $S^R_{b_2}=[b_1,b_2]$). 나머지 블록 b_3, b_4, b_5를 차례로 삽입하면 입력으로 주어진 CBL의 평면계획이 완성되며 그 결과는 유일하다[18].

위에 기술한 스택의 특성에 의해 $|S^T_{b_i}|$, $|S^R_{b_i}|$은 다음 lemma 1을 만족한다.

Lemma 1.

$|S^T_{b_i}|$, $|S^R_{b_i}|$에 대하여,

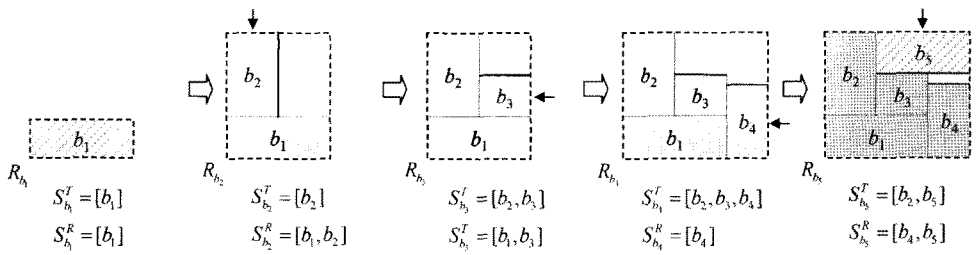


그림 4 CBL packing process with S = (b_1,b_2,b_3,b_4,b_5), L = (0,1,1,0), T = (0,0,1,1)

$i=1$, $\left|S^T_{b_1}\right|=1$, $\left|S^R_{b_1}\right|=1$

$$i=2,3,\cdots,n,\quad \left|S^T_{b_i}\right|=\begin{cases}\left|S^T_{b_{i-1}}\right|-T_{b_i} & if\ L_{b_i}=0\\ \left|S^T_{b_{i-1}}\right|+1 & if\ L_{b_i}=1\end{cases},$$

$$\left|S^R_{b_i}\right|=\begin{cases}\left|S^R_{b_{i-1}}\right|+1 & if\ L_{b_i}=0\\ \left|S^R_{b_{i-1}}\right|-T_{b_i} & if\ L_{b_i}=1\end{cases}$$

Lemma 1의 증명은 [6]의 위쪽 경계와 오른쪽 경계에 존재하는 T-접합 수의 특성에 의해 쉽게 증명된다.

2.2 Feasible CBL

CBL 퍼터베이션은 6가지 이동방법(move operation)을 사용하여 이웃해를 생성한다[18]. 그러나 퍼터베이션을 통해 생성된 CBL가 항상 패킹 가능한 것은 아니다. L과 T 리스트에 대한 임의적인 수정은 패킹 과정에서 블록 삽입에 필요한 위쪽 또는 오른쪽 경계의 블록 수를 부족하게 만들 수 있기 때문이다. CBL가 패킹 가능할 경우, 이것을 feasible CBL라 한다. 다음 lemma는 feasible CBL의 조건을 기술한다.

Lemma 2. (Feasible CBL)

어떤 CBL에 포함된 모든 블록 $b_i \in S\ (1 \le i \le n)$에 대하여, $\left|S^T_{b_i}\right| \ge 1$이고 $\left|S^R_{b_i}\right| \ge 1$이면, 이 CBL는 feasible CBL이다.

증명. $i=1$일 때, lemma 1에 의해 $\left|S^T_{b_1}\right|=1$, $\left|S^R_{b_1}\right|=1$이다. $i=k$일 때, $\left|S^T_{b_k}\right| \ge 1 \wedge \left|S^R_{b_k}\right| \ge 1$이 참임을 가정한다. $i=k+1$일 때, $L_{b_{k+1}}=0$이면 $\left|S^T_{b_{k+1}}\right|=\left|S^T_{b_k}\right|-T_{b_k}$, $\left|S^R_{b_{k+1}}\right|=\left|S^R_{b_k}\right|+1$이다. 그리고, $L_{b_{k+1}}=1$이면 $\left|S^T_{b_{k+1}}\right|=\left|S^T_{b_k}\right|+1$, $\left|S^R_{b_{k+1}}\right|=\left|S^R_{b_k}\right|-T_{b_k}$이다. $L_{b_{k+1}}=0$인 경우, $\left|S^T_{b_k}\right|-T_{b_k} \le 0$을 가정하면 $R_{b_{k+1}}$의 위쪽 경계와 인접한 블록이 존재하지 않음을 의미하고 이는 모순이다. 그러므로, 항상 $\left|S^T_{b_{k+1}}\right| \ge 1$을 만족해야 한다. 그리고, 가정에 의해 $\left|S^R_{b_k}\right| \ge 1$ 이므로 $\left|S^R_{b_{k+1}}\right|=\left|S^R_{b_k}\right|+1 \ge 1$을 만족해야 한다. $L_{b_{k+1}}=1$인 경우도 마찬가지로 쉽게 성립함을 보일 수 있다. 따라서, 수학적 귀납법에 의해 lemma 2는 항상 성립된다.

논문에서 제안된 패킹 단계는 항상 feasible CBL를 입력으로 한다. 따라서 CBL의 feasibility를 검증할 수 있는 방법이 필요하고, 이것은 lemma 1과 lemma 2를 사용해 쉽게 검증 가능하다. 즉, 임의의 CBL이 주어졌을 때, 모든 블록에 대해 $\left|S^T_{b_i}\right|$, $\left|S^R_{b_i}\right|$이 lemma 1에 의해 계산가능하고, $\left|S^T_{b_i}\right|$, $\left|S^R_{b_i}\right|$이 lemma 2를 만족하면 주어진 CBL는 feasible CBL가 된다. 이 방법은 스택 연산, 좌표계산 등의 연산을 수행하지 않고 CBL의 feasibility를 검증할 수 있으므로 계산 시간을 단축할 수 있다.

2.3 기존 연구의 문제점

[23]에서 고정블록 제약조건을 처리할 수 있는 두 가지 방법을 제안하였다. 첫번째 방법은 자유블록 만으로 구성된 CBL과 고정블록 리스트에 대하여, 패킹과정 중 고정블록과 자유블록 사이에 겹침이 발생하면 자유블록의 좌표를 삽입방향으로 고정블록과의 겹침이 제거될 때까지 이동시키는 방법이다. 두번째 방법은 고정블록과 자유블록을 모두 포함하는 CBL에 대하여, 패킹과정에서 발생하는 고정블록과 자유블록의 겹침을 블록 교환(swap)을 통하여 문제를 해결하는 방법이다. 그림 5는 두번째 방법을 간략하게 기술한 것이다.

첫번째 방법은 [23]에서 이미 지적했듯이 그림 10(c)와 같은 최적해를 생성할 수 없다. 두번째 방법의 문제점은 모든 경우의 CBL에 대하여 안정적인 평면계획을 생성하지 못하는 문제점이 있다. 그림 6은 [23]의 두번째 방법으로 패킹했을 때 정상적인 패킹이 불가능한 경우를 예로 보인 것이다. b_3, b_4가 고정블록이고, 그림 6(a)는 b_1까지 삽입이 완료된 상태이다. 그림 6(b)는 b_2와 고정블록 b_3 사이에 겹침이 발생했음을 보인다. 그림 6(c)는 b_2와 b_3를 교환하고 b_3를 미리 정해진 위치에 삽입한 뒤, 다시 b_2를 삽입하고자 했을 때 b_2와 고정블록 b_4 사이에 겹침이 발생했음을 보인다. 이때 b_3는 R_{b_1}의

```
Procedure : Packing
  -Input : CBL with all blocks, pre-placed block list
  -Output : Floorplan with satisfying pre-pleced block constraints
-----------------------------------
1: for ( i=1 ; i<n ; i++ ) {
2:   if (b_i is a free block) {
3:     overlap_check( b_i, preplaced blocks list );
4:     if ( b_i overlaps with a preplaced block b_j ) {
5:       swap(b_i,b_j);
6:       place(b_j); //b_j is placed in b_j's pre-placed position.
7:     }
8:     else place(b_i); //b_i is placed in packing position.
9:   }
10:  else { //b_i is a pre-placed block.
11:    if ( b_i overlaps with b_j's pre-placed position )
12:      place(b_i); //b_i is placed in b_j's pre-placed position.
13:    else
14:      swap(b_i, b_j); //b_j is the next free block after b_i in CBL.
15:  }
16: }
```

그림 5 CBL packing method in [23]

위쪽에서 삽입된다. 그러나 $L_{b_3}=1$로 모순된 값을 갖는다. 그림 6(d)는 b_2와 b_4를 교환하고 b_4를 미리 정해진 위치에 삽입한 뒤, 다시 b_2를 삽입한 결과를 보인 것이다. 이때 b_4는 R_{b_3}의 우측에서 삽입된다. 그러나 $L_{b_4}=0$로 모순된 값을 갖는다. 이처럼 블록 교환에 의해서 고정블록의 실제 삽입방향과 관계없는 L값으로 인해 그림 6(d)에서 b_2가 삽입될 때 문제가 발생하고 이로 인해 최적해를 찾을 수 없게 된다.

3. Preplaced Block Insertion

CBL 패킹과정은 자유블록 만으로 구성된 feasible CBL와 고정블록 리스트 P를 입력 받아 자유블록과 고정블록이 모두 포함된 평면계획을 구하는 과정이다. 이 과정에서 고정블록이 하나 삽입될 때마다 그 블록에 대응하는 구성요소가 C에 새로이 추가되고 C의 기존 요소 값이 조정되어, C는 feasible하고 본래 주어진 위상과 크게 다르지 않은 새로운 CBL로 갱신된다.

CBL C의 S, L, T를 각각 $S=(b_1,b_2,...,b_n)$, $L=\left(L_{b_2}, L_{b_3}, \cdots, L_{b_n}\right)$, $T=\left(T_{b_2}, T_{b_3}, \cdots, T_{b_n}\right)$라고 하고 고정블록 리스트 $P=(p_1,p_2,...,p_m)$라고 하자. 패킹 초기에 S 리스트의 블록들은 모두 자유블록이며 b_1부터 차례로 CBL 패킹 방법에 따라 패킹된다. 그런데 만일 블록 b_i를 패킹할 차례가 되었을 때 b_i와 어떤 고정블록 p_k간에 겹침(overlap)이 발생한다면 다음과 같은 단계를 거쳐 b_i 대신 p_k을 패킹한다:

1. p_k 의 L, T 값을 결정한다(LT-determination).
2. 블록 $b_i,b_{i+1},\cdots,b_n$ 의 T 값을 본래 CBL 위상과 크게 다르지 않도록 조정한다(T-legalization).
3. $S=(b_1,\cdots,b_{i-1},p_k,b_i,\cdots,b_n)$, $L=(L_{b_2},\cdots,L_{b_{i-1}},L_{p_k},L_{b_i},\cdots,L_{b_n})$, $T=(T_{b_2},\cdots,T_{b_{i-1}},T_{p_k},T'_{b_i},\cdots,T'_{b_n})$로 CBL C를 갱신한다. 즉, p_k가 S 리스트에 새로 추가되었으며 L_{p_k}, T_{p_k}는 단계 1에서 결정한 값이다. T 리스트의 $T'_{b_i},\cdots,T'_{b_n}$ 값들은 단계 2에서 새로 조정된 값이다.
4. 고정블록 p_k를 삽입한다.

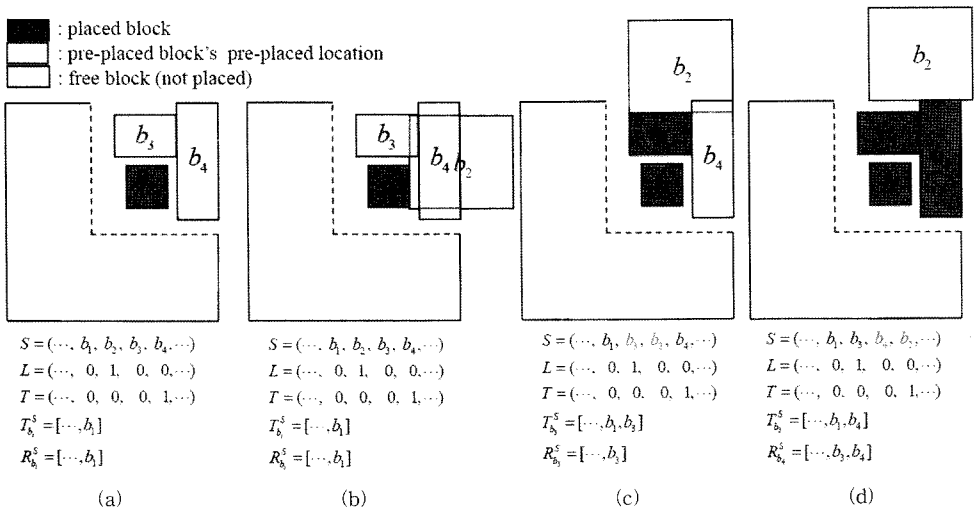


그림 6 Packing example by 2nd method in [23]

새로 구성된 CBL을 다시 $S=(b_1,b_2,\cdots,b_n)$, $L=(L_{b_2},L_{b_3},\cdots,L_{b_n})$, $T=(T_{b_2},T_{b_3},\cdots,T_{b_n})$로 표기하면 n은 이전 값보다 하나 증가된 값이고 b_{i+1}은 이전 CBL의 b_i에 해당한다. 따라서, 블록 b_{i+1}부터 동일한 패킹과정을 모든 자유블록과 고정블록이 삽입될 때까지 반복한다. 그리고 과정 3에서 T 값에 영향을 받는 블록들을 p_k의 희생블록(victim block)이라 한다.

블록 b의 너비와 높이를 각각 w_b, h_b로 표기하고, 좌하부 모서리 좌표를 (x_b, y_b)로 표기한다. 임의의 자유블록 b와 고정블록 p가 겹침(overlap)을 가질 조건은 다음과 같다:

$$x_b + w_b \geq x_p \text{ and } y_b + h_b \geq y_p \quad (1)$$

CBL에서 고정블록들 사이에는 삽입 순서가 존재한다. 임의의 고정블록 p_a, p_b가 다음과 같은 조건이 만족하면 p_a는 p_b보다 먼저 삽입되어야 한다:

$$x_{p_a} + w_{p_a} < x_{p_b} + w_{p_b} \text{ and } y_{p_a} + h_{p_a} < y_{p_b} + h_{p_b} \quad (2)$$

만일 (2)를 만족함에도 불구하고 p_b가 p_a보다 먼저 삽입된다면 이를 통하여 만들어지는 패킹 영역과 p_a가 겹치게 되므로 p_a를 삽입할 수 없다. 그러나 고정블록들이 (2)를 만족하지 않으면 삽입 순서에 제약을 받지 않는다. 따라서, 고정블록들 사이의 삽입순서는 두 단계의 정렬을 통하여 결정할 수 있다. 우선 고정블록들을 x+w로 정렬하고, 그 결과를 y+h로 안정(stable) 정렬한다. 그러면 고정블록들 사이에 순서가 존재할 경우, 그 순서를 만족하는 정렬된 리스트 $P=(p_1,p_2,\cdots,p_m)$를 얻을 수 있다.

블록 겹침 검사 과정에서 b_i와 p_k 사이에 겹침이 존재하지 않고 다음과 같은 조건을 만족하면, 정렬된 고정블록 리스트 P에서 p_k 이후 b_i와 겹침이 있는 고정블록은 존재하지 않는다.

$$x_{b_i} + w_{b_i} < x_{p_k} + w_{p_k} \text{ and } y_{b_i} + h_{b_i} < y_{p_k} + h_{p_k} \quad (3)$$

따라서 b_i의 블록 겹침 검사는 (1)을 만족하거나 (3)을 만족하면 중단한다. 이것은 불필요한 검사를 하지 않음으로써 계산시간을 단축할 수 있다.

현재 CBL $S=(b_1,b_2,\cdots,b_n)$, $L=(L_{b_2},L_{b_3},\cdots,L_{b_n})$, $T=(T_{b_2},T_{b_3},\cdots,T_{b_n})$에 대해서 블록 b_{i-1}까지 패킹된 상태이고 블록 b_i가 고정블록 p_k와 겹쳐 b_i 대신 p_k를 삽입한다고 가정하자. 이 경우 $b_1,b_2,\cdots,b_{i-1}$는 이미 삽입된 자유블록과 고정블록들이며 $b_i,b_{i+1},\cdots,b_n$는 아직 삽입되지 않은 자유블록들이다. 본 장의 나머지 부분에서는 이러한 상태에서 p_k의 L, T 값을 결정하는 과정인 LT-determination 과정과 블록 $b_i,b_{i+1},\cdots,b_n$들의 T 값을 조정하는 과정인 T-legalization 과정을 기술한다.

3.1 LT-Determination

앞에서 언급한 것처럼 현재 CBL C에서 블록 b_{i-1}까지 삽입된 상태이고(구성된 영역을 $R_{b_{i-1}}$라고 하자) 블록 b_i와 고정블록 p_k가 겹쳐 b_i 대신 p_k를 삽입하고자 한다. 고정블록 p_k는 그 위치와 크기가 이미 고정되어 있으므로 이를 CBL 패킹 방법으로 $R_{b_{i-1}}$에 삽입하기 위해서는 L_{p_k}, T_{p_k}를 적절히 결정하여야 한다. 이 결정은 CBL 패킹방법으로 다른 블록들을 연속해서 원활히 삽입하기 위해서도 필요하다.

고정블록 p_k를 삽입하기 위해서는 패킹영역 $R_{b_{i-1}}$의 상변 또는 우변에 p_k가 커버할 수 있는 블록이 존재하여야 한다. 고정블록 p_k의 L_{p_k}, T_{p_k} 값은 다음에 보이는 세가지 경우 중 하나에 의하여 결정된다:

Case 1. $R_{b_{i-1}}$의 상변에 위치한 블록들을 좌로부터 우로 $u_d,u_{d-1},\cdots,u_1$이라고 하자. 만일 $x_{u_t} \leq x_{p_k} < x_{u_{t-1}}$을 만족하는 블록 $u_t,u_{t-1},\cdots,u_1$이 존재하고 $y_{p_k} \geq \max_{1\leq j\leq t}\{y_{u_j}+h_{u_j}\}$라면 p_k는 $R_{b_{i-1}}$의 위에서 수직으로 삽입할 수 있다. 따라서 이 경우 $L_{p_k}=0$이며 t개의 블록을 커버하므로 $T_{p_k}=t-1$로 정하면 된다.

Case 2. $R_{b_{i-1}}$의 우변에 위치한 블록들을 위에서 아래로 $r_1,r_2,\cdots,r_h$라고 할 때 $y_{r_v} \leq y_{p_k} < y_{r_{v-1}}$을 만족하는 블록 $r_v,r_{v-1},\cdots,r_1$이 존재하고 $x_{p_k} \geq \max_{1\leq j\leq v}\{x_{r_j}+w_{r_j}\}$이면 p_k를 $R_{b_{i-1}}$의 우측에서 수평으로 삽입할 수 있다. 따라서 이 경우 $L_{p_k}=1$, $T_{p_k}=v-1$이다.

Case 3. 고정블록 p_k가 위에서 보인 두 경우를 모두 만족할 수 없다는 것은 자명하다. 그러나 만일 p_k가 두 조건 모두 만족하지 않는다면 $R_{b_{i-1}}$의 상변과 우변 모두 p_k가 커버할 수 있는 블록이 존재하지 않는다. 이 경우 편의상 $L_{p_k}=0$, $T_{p_k}=0$으로 설정하여 p_k가 블록 b_{i-1}을 수직으로 커버한다고 가정한다.

CBL 패킹에서 블록 하나가 삽입될 때 이의 좌표는 기존 삽입된 블록들에 의하여 결정된다. 만일 어떤 블록 b의 삽입방향이 수직(수평)이고 b가 커버하는 블록 중 가장 왼쪽(아래쪽) 블록을 b_c라고 하면 $x_b = x_{b_c}$ ($y_b = y_{b_c}$)로 결정된다. 마찬가지로 고정블록 p_k의 경우 비록 그

위치가 고정되어 있으나 원활한 패킹을 위하여 그 좌표 및 크기를 임시로 변경한다. 임시로 변경된 p_k의 좌표를 (x'_{p_k}, y'_{p_k}), 너비를 w'_{p_k}, 높이를 h'_{p_k}라 하자. 그러면, case 1의 경우 $x'_{p_k} = x_{u_t}$, case 2의 경우 $y'_{p_k} = y_{r_c}$, 그리고 case 3의 경우 $x'_{p_k} = x_{b_{i-1}}$이다. 그리고 모든 경우에 대해 $w'_{p_k} = w_{p_k} + x_{p_k} - x'_{p_k}$, $h'_{p_k} = h_{p_k} + y_{p_k} - y'_{p_k}$이다.

고정블록 p_k가 위 세가지 경우 중 어디에 해당되는지는 S^T와 S^R을 검색하여 판단할 수 있다. 블록 b_{i-1}을 삽입한 후의 스택 상태는 $S^T_{b_{i-1}} = [u_d, u_{d-1}, \cdots, u_1]$, $S^R_{b_{i-1}} = [r_b, r_{b-1}, \cdots, r_1]$이므로 이 두 스택을 검색하여 조건이 맞는지 조사할 수 있다.

3.2 T-Legalization

LT-determination 과정에서 결정된 L_{p_k}, T_{p_k}로 고정블록 p_k를 삽입하면 패킹 영역의 상변과 우변에 접해있는 블록들의 상태가 달라져 블록 b_i부터 나머지 블록들을 삽입하는데 문제가 있을 수 있다. T-legalization 과정에서는 블록 b_i부터 차례로 블록을 검사하여 이들 각각이 차지하는 영역이 p_k를 삽입하지 않고 패킹했을 때 차지하는 영역과 유사하도록 블록의 T 값을 조정한다. 고정블록 p_k가 수직으로 삽입될 경우와 수평으로 삽입될 경우의 T 값 조정방법은 유사하므로 본 절에서는 p_k가 수직으로 삽입될 경우($L_{p_k} = 0$)의 T 값 조정에 대해서만 기술한다.

T-legalization 과정은 p_k를 삽입하기 전에 수행되므로 패킹영역은 $R_{b_{i-1}}$이며 LT-determination 과정에서와 마찬가지로 $R_{b_{i-1}}$의 상변에 위치한 블록들은 좌로부터 우로 $u_d, u_{d-1}, \cdots, u_1$이라고 하고($u_1 = b_{i-1}$), p_k는 블록 $u_t, u_{t-1}, \cdots, u_1$를 커버한다고 하자. 블록 p_k부터 차례로 $R_{b_{i-1}}$에 삽입한다고 할 때, $L_{b_i} = 0$이면 b_i를 수평으로 커버하는 블록을 b_h라고 하고, $L_{b_i} = 1$이면 b_{i-1}을 수평으로 커버하는 블록을 b_h라고 하자. 그리고, 블록 $u_d, u_{d-1}, \cdots, u_1$ 중 일부를 커버하면서 수직으로 처음 삽입되는 블록을 b_f라고 하자.

만일 고정블록 p_k를 삽입한 후 CBL C의 나머지 블록들을 삽입하면 블록 b_h는 본래 커버하는 블록 외에 p_k를 추가로 커버하게 되어 $T'_{b_h} = T_{b_h} + 1$이 된다. 그리고 b_h 이후 수평으로 삽입되는 블록들은 b_h가 p_k를 수평으로 커버하므로 p_k 삽입 이전과 동일한 위상을 갖는다. 따라서, 이 블록들의 T 값을 조정할 필요가 없다. 또한 b_h 이전에 수평으로 삽입되는 블록들은 p_k의 영향을 받지 않으므로 이들 역시 T 값을 조정할 필요가 없다.

블록 b_f 이전에 수직으로 삽입되는 블록들은 p_k의 영향을 받지 않으므로 이들의 T 값을 조정할 필요가 없다. 블록 b_f는 블록 $u_d, u_{d-1}, \cdots, u_1$ 중 일부를 처음으로 커버하는 블록이므로 b_f가 삽입될 때 블록 $u_d, u_{d-1}, \cdots, u_1$들은 패킹영역 $R_{b_{f-1}}$의 상변에 그대로 인접해 있고 추가로 b_f 이전에 수직으로 삽입된 블록들이 인접해 있을 수 있다. 모두 블록 u_1의 우측에 위치하는 이들을 좌로부터 $u^e_g, u^e_{g-1}, \ldots, u^e_1$라고 하자.

블록 $u_d, u_{d-1}, \cdots, u_1$ 중 블록 b_f가 커버하는 블록을 $u_c, u_{c-1}, \cdots, u_1$이라고 하면 b_f의 T 값은 다음의 두 경우에 따라 조정된다:

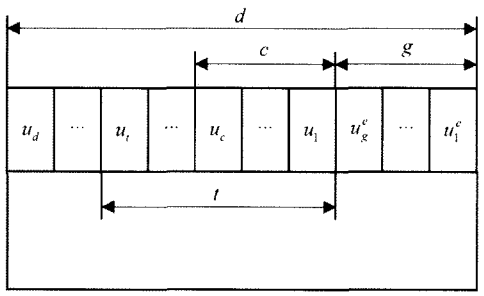
Case 1. $c < t$이면 b_f는 본래 $c+g$개의 블록을 커버하였으나 고정블록 p_k가 삽입된 경우 이를 새로 커버하고 대신 c개의 블록을 커버하지 않게 되므로 $T'_{b_f} = T_{b_f} - c + 1$이 된다.

Case 2. $c \geq t$이면 b_f는 본래 $c+g$개의 블록을 커버하였으나 고정블록 p_k가 삽입된 경우 이를 새로 커버하고 대신 t개의 블록을 커버하지 않게 되므로 $T'_{b_f} = T_{b_f} - t + 1$이 된다.

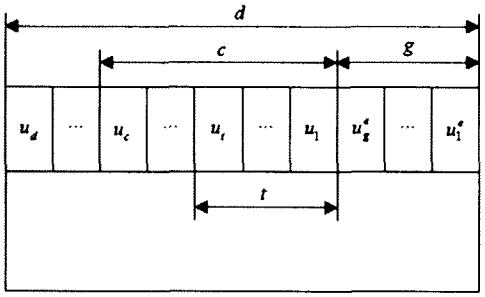
블록 b_f의 T 값이 위의 case 2에 의하여 조정되었다면, 이것은 u_c가 b_f에 의해 커버됨을 의미한다. 따라서 b_f 이후에 수직으로 삽입되는 블록들은 p_k 삽입 이전과 동일한 위상관계를 갖고 이들의 T 값을 조정할 필요가 없다. 그러나 만일 case 1에 의하여 T 값이 조정되었다면 수직으로 이어서 삽입되는 블록들에 대해서 T 값을 조정할 필요가 있는지 계속 조사하여야 하며 이는 어떤 블록이 p_k가 삽입되며 커버할 블록들을 모두 커버할 때까지 진행하여야 한다. 그림 7은 b_f가 삽입되기 직전의 평면계획 상태를 나타낸 것으로 (a)는 $c<t$, (b)는 $c \geq t$의 경우를 보인 것이다.

현재 블록 b_w까지 조사를 마쳤고 b_w의 T 값이 조정되었다고 가정하자. 만일 고정블록 p_k를 삽입하지 않은 상태에서 블록들을 b_w까지 계속 삽입하였을 때 블록 $u_t, u_{t-1}, \cdots, u_1$ 전부가 b_w를 포함한 이전 삽입된 블록들에 의하여 커버되었다면 이후 삽입되는 블록들은 더 이상 p_k의 영향을 받지 않으므로 T-legalization 과정을 종료하게 된다.

블록 $u_t, u_{t-1}, \ldots, u_1$ 중 일부가 b_w를 포함한 이전 삽입된 블록들에 의하여 아직 커버되지 않았다고 가정하자. 이 경우 $e(<t)$ 개의 블록만이 커버되었다고 하면 패킹

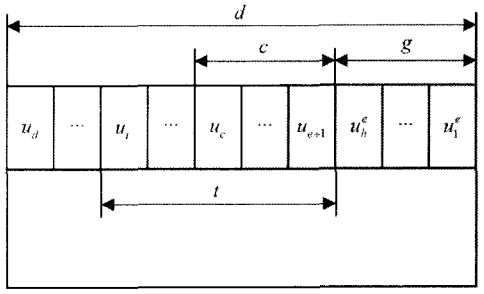


(a) $c < t$

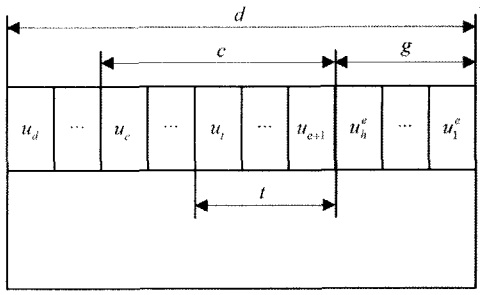


(b) $c \geq t$

그림 7 T-legalization : b_f victim block case



(a) $c < t$



(b) $c \geq t$

그림 8 T-legalization : b_s victim block case

영역 R_{b_w} 의 상변에는 $u_d, u_{d-1}, \cdots, u_t, \cdots, u_{c+1}, b_w$ 등의 블록이 인접해 있다($e = t - 1$일 경우, $u_{e+1} = u_t$ 이다). 블록 b_{w+1} 부터 계속해서 차례로 R_{b_w} 에 블록을 삽입한다고 할 때 블록 $u_d, u_{d-1}, \cdots, u_{e+1}$ 중 일부를 커버하면서 수직으로 처음 삽입되는 블록을 b_s 라고 하자. 만일 $w < j < s$ 인 블록 b_j 가 존재하면, b_j 는 p_k 의 영향을 받지 않음으로 이의 T 값은 조절할 필요가 없다.

블록 b_w 를 포함하여 b_s 전까지 삽입된 블록들의 일부가 패킹영역 $R_{b_{s-1}}$ 의 상변 우측에 인접해 있을 수 있는데 이들을 $u_h^e, u_{h-1}^e, \cdots, u_1^e$ 라고 하자. 패킹영역 $R_{b_{s-1}}$ 의 상변에는 블록 $u_d, u_{d-1}, \cdots, u_t, \cdots, u_{e+1}, u_h^e, u_{h-1}^e, \cdots, u_1^e$ 가 좌에서 우로 인접해 있다. 이러한 상태에서 블록 $u_d, u_{d-1}, \cdots, u_t, \cdots, u_{e+1}$ 중 블록 b_s 가 커버하는 블록을 $u_c, u_{c-1}, \cdots, u_{e+1}$ 이라고 하면 아래 보이는 두 가지 경우에 따라 b_s 의 T 값을 갱신한다.

Case 1. $c < t$ 이면 b_s 는 본래 $c - e + h$개의 블록을 커버하였으나 고정블록 p_k 가 삽입된 경우 $b_w (= u_h^e)$ 를 커버하고 대신 $c - e$개의 블록을 커버하지 않게 되므로 $T_{b_s}^{'} = T_{b_s} - (c - e)$ 이 된다.

Case 2. $c \geq t$ 이면 b_s 는 본래 $c - e + h$개의 블록을 커버하였으나 고정블록 p_k 가 삽입된 경우 $b_w (= u_h^e)$ 를 커버하고 대신 $t - e$개의 블록을 커버하지 않게 되므로 $T_{b_s}^{'} = T_{b_s} - (t - e)$ 이 된다. 그리고 이 경우 더 이상 T-legalization 과정을 진행할 필요가 없다.

그림 8은 b_s 가 삽입되기 직전의 평면계획 상태를 나타낸 것이다.

3.3 스택을 이용한 빠른 T-Legalization

3.2절에서 기술한 T-Legalization 과정은 CBL 패킹에 사용하는 스택 S^T, S^R 을 이용하여 빠르게 수행할 수 있다. 본 절에서는 이러한 방법을 기술하며 편의상 3.2절에서 사용한 각 경우의 블록 표기를 그대로 사용한다.

CBL C의 블록 b_i 부터 b_n 까지 이들의 T 값을 조정하는 과정은 패킹영역 $R_{b_{i-1}}$ 에 고정블록 p_k 를 삽입하지 않고 블록 $b_i, b_{i+1}, \ldots, b_n$ 을 차례로 삽입한다고 가정하였을 때 변화하는 스택의 크기를 조사하여 수행할 수 있다. 먼저 첫번째 희생블록 b_f 는 아래 lemma 3에 의하여 찾을 수 있다.

Lemma 3. 블록 b_i 부터 차례로 조사하여 현재 $L_{b_q} = 0$ 인 블록 b_q 를 조사한다고 할 때 만일 $|S_{b_{i-1}}^T| < |S_{b_c}^T|$ ($i \leq c < q$)이고 $|S_{b_{i-1}}^T| \geq |S_{b_q}^T|$ 라면 b_q는 첫번째 희생블록 b_f

이다.

증명. 블록 b_{i-1} 은 고정블록 p_k 가 커버하는 첫번째 블록 u_1 이다. 만일 어떤 블록 b_c 를 삽입하였을 때 $|S^T_{b_{i-1}}|<|S^T_{b_c}|$ 라면 S^T에 블록이 추가된 것이므로 b_c 는 b_{i-1} 을 커버하지 않고 삽입된 것이다. 따라서, b_c 의 T 값을 조정할 필요가 없다. 그러나 $|S^T_{b_f}|\geq|S^T_{b_q}|$ 이면 블록 b_q 는 b_{i-1} 를 커버하고 삽입되었음을 의미하므로 b_q 는 첫번째 희생블록 b_f 가 된다. □

첫번째 희생블록 b_f 를 구하였으면 이의 T 값은 아래 Lemma 4에 의하여 조정한다.

Lemma 4. 블록 b_f 의 조정된 T 값 T'_{b_f} 는 다음의 두 경우에 따라 결정된다:

Case 1. 만일 $|S^T_{u_1}|<|S^f_{b_f}|$ 이면 $T'_{b_f}=T_{b_f}+|S^T_{b_f}|-|S^T_{b_{i-1}}|$ 이다.

Case 2. 반대로 $|S^T_{u_1}|\geq|S^T_{b_f}|$ 이면 $T'_{b_f}=T_{b_f}-T_{p_k}$ 이다. 이 경우 b_f 는 마지막 희생블록이다.

증명. Case 1. $|S^T_{b_{i-1}}|=d$, $|S^T_{b_{f-1}}|=d+g$ 이다. b_f 를 삽입하면 $g+c$개가 S^T에서 제거되고 b_f 가 추가되므로 $|S^T_{b_f}|=d-c+1$ 이다. 그러므로 $|S^T_{b_f}|-|S^T_{b_{f-1}}|=-c+1$ 이다. 따라서 $T'_{b_f}=T_{b_f}+|S^T_{b_f}|-|S^T_{b_{i-1}}|=T_{b_f}-c+1$ 이다.

Case 2. $T_{p_k}=t-1$ 이다. 따라서 $T'_{b_f}=T_{b_f}-T_{p_k}=T_{b_f}-t+1$ 이다. □

블록 b_w 가 가장 최근 발견된 희생블록이라 할 때 다음 희생블록 b_s 는 다음의 lemma 5에 의하여 찾을 수 있다.

Lemma 5. 블록 b_{w+1} 부터 차례로 조사하여 $L_{b_q}=0$ 인 블록 b_q 를 현재 조사한다고 할 때 만일 $|S^T_{b_w}|\leq|S^T_{b_c}|$ $(w+1\leq c<q)$이고 $|S^T_{b_w}|>|S^T_{b_q}|$ 라면 b_q 는 b_w 다음 희생블록 b_s 이다.

증명. 블록 b_w 는 가장 최근 발견된 희생블록이고 b_w 가 b_e 를 커버하므로 $|S^T_{b_{e+1}}|=|S^T_{b_w}|-1$ 이다. 따라서, $|S^T_{b_{e+1}}|<|S^T_{b_c}|$ 와 $|S^T_{b_w}|\leq|S^T_{b_c}|$ 는 동치이고, $|S^T_{b_{e+1}}|\geq|S^T_{b_q}|$ 와 $|S^T_{b_w}|>|S^T_{b_q}|$ 는 동치이다. 만일 어떤 블록 b_c 를 삽입하였을 때 $|S^T_{b_{e+1}}|<|S^T_{b_c}|$ 라면 S^T에 블록이 추가된 것이므로 b_c 는 b_{e+1} 를 커버하지 않고 삽입된 것이다. 따라서 b_c 의 T 값은 조정할 필요가 없다. 그러나 $|S^T_{b_{e+1}}|\geq|S^T_{b_q}|$ 이면 b_q 가 b_{e+1} 를 커버하고 삽입되었음을 의미하므로 b_q 는 b_w 다음 희생블록 b_s가 된다. □

발견한 희생블록 b_s 의 T 값은 다음 lemma 6에 의하여 조정된다.

Lemma 6. 블록 b_s 의 조정된 T 값 T'_{b_s} 는 다음의 두 경우에 따라 결정된다:

Case 1. 만일 $|S^T_{b_{i-1}}|-T_{p_k}<|S^T_{b_s}|$ 이면 $T'_{b_s}=T_{b_s}-|S^T_{b_w}|+|S^f_{b_s}|$ 이다.

Case 2. 반대로 $|S^T_{b_{i-1}}|-T_{p_k}\geq|S^T_{b_s}|$ 이면 $T'_{b_s}=T_{b_s}-|S^T_{b_w}|+|S^T_{b_{i-1}}|-T_{p_k}$ 이다. 이 경우 b_s 는 마지막 희생블록이다.

증명. Case 1. $|S^T_{b_w}|=d-e+1$, $|S^T_{b_s}|=d-c+1$ 이다. 그러므로 $|S^T_{b_w}|-|S^T_{b_s}|=c-e$ 이다. 따라서 $T'_{b_s}=T_{b_s}-|S^T_{b_w}|+|S^T_{b_s}|=T_{b_s}-(c-e)$ 이다.

Case 2. $|S^T_{b_w}|=d-e+1$, $|S^T_{b_{i-1}}|=d$, $T_{p_k}=t-1$ 이다. 그러므로 $|S^T_{b_w}|-|S^T_{b_{i-1}}|+T_{p_k}=t-e$ 이다. 따라서 $T'_{b_s}=T_{b_s}-|S^T_{b_w}|+|S^T_{b_{i-1}}|-T_{p_k}=T_{b_s}-(t-e)$ 이다. □

그림 9에 본 절에서 보인 lemma들을 이용하여 블록 b_1 부터 차례로 검사하여 각 블록의 T 값을 조정하는 과

```
Procedure : T-legalization
  -Input : Feasible CBL with only free blocks, Pre-placed block list
  -Output : Feasible CBL with all blocks
---------------------------------------------------------------------
1: q = i,  first_R_flag = 0, first_T_flag = 0;
2: |S^T| = |S^T_{b_i}|,  |S^R| = |S^R_{b_i}|,  |S^f_{b_1}| = |S^T_{b_{i-1}}| - T_{p_k};
3: while ( q < n ) {
4:   if (L_{b_q} = 1) {
5:     |S^T|++;   |S^R| = |S^R| - T_{b_q};
6:     if (first_R_flag = 0 ) {
7:       if ( |S^R_{b_c}| ≥ |S^R| ) { T_{b_q} = T_{b_q} + 1;   first_R_flag=1; }
8:     }
9:   } else {
10:    |S^T| = |S^T| - T_{b_q};   |S^R|++;
11:    if ( first_T_flag = 0 ) {
12:      if ( |S^T_{b_{w+1}}| ≥ |S^T| ) {
13:        first_T_flag = 1;
14:        if (|S^T_{b_c}| < |S^T| ) { T_{b_q} = T_{b_q} + |S^T| - |S^T_{b_{w+1}}|;   |S^T_{b_w}| = |S^T|; }
15:        else { T_{b_q} = T_{b_q} - T_{p_k};   break; }
16:      }
17:    } else if ( |S^T_{b_w}| > |S^T| ) {
18:      if (|S^T_{b_{w+1}}| - T_{p_k} < |S^T| ) { T_{b_q} = T_{b_q} + |S^T_{b_w}| - |S^T|;   |S^T_{b_w}| = |S^T|; }
19:      else { T_{b_q} = T_{b_q} - |S^T_{b_c}| + |S^T_{b_{c-1}}| - T_{p_k};   break; }
20:    }
21:  }
22:  q++;
23: }
```

그림 9 T-legalization procedure

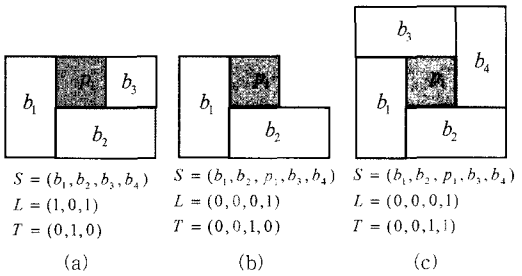


그림 10 T-legalization procedure

정을 보인다. 실제 패킹은 블록 b_{i-1}까지만 되어있고 스택 또한 $S^T_{b_{i-1}}, S^R_{b_{i-1}}$까지만 구성되어 있다. 그림 9의 과정은 블록을 삽입한다는 가정하에 스택의 크기만 조정하여 각 블록의 T값을 조정하는 과정이다.

그림 10은 S = (b_1,b_2,b_3,b_4), L = (1,0,1), T = (0,1,0)와 고정블록 p_1이 주어졌을 때, 제안된 방법의 패킹 예를 보인 것이다. 그림 10(a)는 b_1, b_2가 이미 삽입되어 있고, b_3를 삽입하려 할 때, p_1과 겹침이 발생한 상태를 보인 것이다. 그림 10(b)는 b_3 대신 p_1을 삽입하기 위해 $L_{p_1} = 0$, $T_{p_1} = 0$을 결정하고 그에 따라 p_1을 삽입한 결과를 보인 것이다. 그림 10(c)는 p_1에 의해 영향 받는 b_4의 T_{p_1}을 수정하고 그 평면계획 결과를 보인 것이다.

3.4 Preplaced Block Insertion의 시간 복잡도

본 논문에서 제안한 새로운 CBL 패킹방법은 기존 CBL에 비해 블록 겹침 검사, LT-determination, T-legalization 등에 추가비용이 요구된다. 블록 겹침 검사는 각 자유블록에 대해 모든 고정블록과 겹침이 존재하는지 확인하는 과정이므로 최악의 경우 $O(lm)$의 시간이 필요하다. LT-determination 과정은 p_k가 커버하는 블록의 수를 찾는데 소요되는 시간이며 이것은 하나의 자유블록을 패킹하기 위한 시간과 같다. 따라서 LT-determination 과정은 $O(1)$의 시간 복잡도를 갖는다. T-legalization의 시간 복잡도는 패킹 후의 S 리스트가 $S = (p_1, \cdots, p_m, b_1, \cdots, b_l)$와 같을 경우 최악의 시간복잡도 $O(lm)$을 갖는다. 그러므로 논문에서 제안된 새로운 패킹 방법의 계산 복잡도는 $O(lm)$이 된다. 따라서, $m \ll l$일 경우, $O(n)$에 근접한 성능을 보이고, $m \approx l$일 경우, $O(l^2)$의 성능을 보인다. 이것은 해공간의 크기가 SP의 $O((n!)^2)$과 CBL의 $O(n!2^{3n-3}/n^{1.5})$에 비해 $O(m!2^{3m-3}/m^{1.5})$로 축소된 것과 함께 전체 계산시간을 단축하는데 기여한다.

4. 실험 결과

본 논문에서 제안한 평면계획 방법은 Microsoft Visual C++ 6.0으로 구현되었다. 실험에 사용된 시스템은 1G byte의 주메모리와 하나의 Intel Pentium IV 2.0Ghz 프로세서를 가진 시스템이다. 결과 비교를 위해 실험에 사용된 회로는 MCNC 벤치마크 회로[24]이고, 그 특징은 표 1과 같다. 실험에 사용된 회로에 포함된 모든 블록들은 하드블록으로 구성되어 있다. 참고문헌에서 가져온 회로를 제외하고 모든 회로의 목표 종횡비를 1.0으로 하여 실험하였다. 그리고 결과를 하기 위해 [20]에서 제안한 CBL 패킹 방법을 적용하였다. 이 방법은 false 블록 추가 없이 ECBL[19]에 근접한 결과를 보인다.

표 2는 고정블록을 포함하지 않은 회로와 고정블록을 포함한 회로에 대한 평면계획 결과를 보인 것이다. 이 결과를 얻기 위해서, 우선 회로에 포함된 모든 블록을 자유블록으로 설정하여 평면계획을 수행하였다. 그리고 각 회로에 포함된 블록들 중 크기가 큰 블록들을 우선

표 1 MCNC Benchmark Circuits

circuit	# of blocks	# of pads	# of nets
apte	9	73	97
xerox	10	2	203
hp	11	45	83
ami33	33	42	123
ami49	49	22	408

표 2 Floorplan results without and with pre-placed blocks

circuit	without constraints			with pre-placed blocks					
	area	white space	run time	# preplaced blocks	area		white space	run time	
					area	incr.		time	incr.
apte	47.53	2.0	1.4	2	47.96	0.9	2.9	1.6	13.9
xerox	19.83	2.4	1.4	2	20.05	1.1	3.5	1.7	15.3
hp	9.36	5.7	1.6	2	9.44	0.8	6.4	1.8	13.8
ami33	1.21	4.1	4.0	3	1.20	-0.7	3.4	5.5	38.0
ami49	36.66	3.3	7.5	4	37.21	1.5	4.7	11.1	48.0
					avrg	0.7		avrg	25.8

(area: mm^2, white space: %, run time: sec, area and time increments: %)

선택하여 고정블록으로 설정하여 평면계획을 수행하였다. 선택된 고정블록의 위치는 자유블록만으로 구성된 평면계획의 결과의 것을 사용하였다. 이 실험은 우리가 제안한 방법이 고정블록을 포함한 회로를 얼마나 효율적이고 효과적으로 배치하는 지를 보여주기 위한 것이다. 표에서 보여진 것과 같이 면적은 평균 0.7% 증가, 수행시간은 평균 25.8% 증가하였다.

표 3은 ami33과 ami49 회로에 대한 평면계획 결과를 보인 것이다. ami33과 ami49 회로에 대해서 각각 3개와 4개의 블록을 고정블록으로 선택하였다. 이 실험에서는 표 2의 실험과는 다르게 고정블록의 위치를 임의로 설정한 각 10가지 경우를 실험하였다. 표 3에서 보여진 것과 같이 면적은 각각 평균 1.0%, 1.7% 증가하였다. 이 실험은 고정블록의 위치가 사용자에 의해 임의로 주어질 경우에도 본 논문에서 제안한 평면계획 방법이 효과적임을 의미한다.

표 4는 많은 수의 고정블록을 포함한 회로들의 실험결과를 보인 것이다. ami33과 ami49 회로에 대해서 각각 8개와 10개의 고정블록을 선택하여, 고정블록들을 임의로 배치하여 각 10개의 경우를 실험을 위해 생성하였다. 실험 결과 고정블록을 포함하지 않은 경우와 비교하여, 고정블록이 회로 전체에서 차지하는 면적과 수가 크므로, 면적에서 각각 평균 5.1%, 2.2%의 증가를 보이고, 계산 시간에서는 각각 87%, 79.3%의 증가를 보인다. 그림 11은 표 4(ami49)의 최선의 경우(circuit 1)와 최악의 경우(circuit 10)를 평면계획 결과를 보인 것이다. 이

표 3 Floorplan results for ami33 and 49

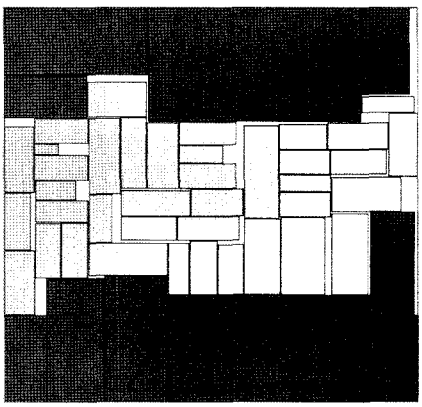
circuit	ami33 (3 pre-placed blocks)					ami49 (4 pre-placed blocks)				
	area		white space	run time		area		white space	run time	
	area	incr		time	incr.	area	incr		time	incr.
0(*)	1.21	-	4.1	4.0		36.66		3.3	7.5	
1	1.21	0.7	4.7	5.5	38.0	37.08	1.2	4.4	11.6	54.6
2	1.22	1.3	5.3	5.4	37.7	36.97	0.8	4.1	11.3	50.6
3	1.23	1.9	5.8	5.5	38.5	37.29	1.7	5.0	11.4	53.2
4	1.21	0.5	4.6	6.0	49.1	37.73	2.9	6.1	11.5	53.6
5	1.22	1.3	5.3	5.5	38.0	37.14	1.3	4.6	11.7	56.6
6	1.22	0.8	4.9	5.6	41.3	37.29	1.7	5.0	11.6	55.2
7	1.21	0.8	4.8	5.4	37.2	37.26	1.6	4.9	11.5	54.0
8	1.21	0.7	4.7	5.6	42.0	37.33	1.8	5.1	10.9	46.5
9	1.21	0.3	4.3	5.5	38.0	37.32	1.8	5.0	11.1	49.0
10	1.22	1.1	5.1	5.5	39.2	37.26	1.6	4.9	11.3	51.4
avrg		1.0			40.0		1.7			52.5

(*) : floorplan results without constraints.
(area: mm^2, white space:%, run time: sec, area and time increments: %)

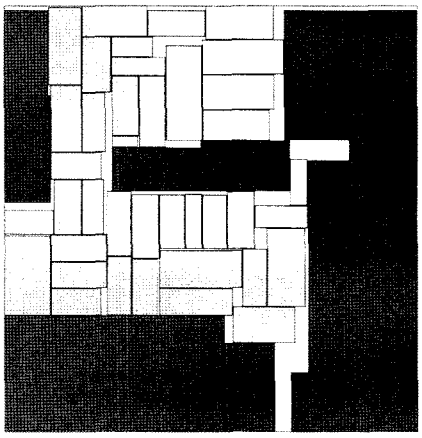
표 4 Another floorplan results for ami33 and 49

circuit	ami33 (8 pre-placed blocks)					ami49 (10 pre-placed blocks)				
	area		white space	run time		area		white space	run time	
	area	incr.		time	incr	area	incr		time	incr
0(*)	1.21	-	4.1	4.0	-	36.66	-	3.3	7.5	-
1	1.24	2.7	6.6	6.4	93.9	37.05	1.1	4.3	13.1	75.8
2	1.29	6.7	10.1	6.1	85.7	37.32	1.8	5.0	13.2	76.6
3	1.25	3.9	7.7	6.1	85.4	37.29	1.7	5.0	12.9	73.2
4	1.29	7.3	10.6	6.0	82.6	37.75	3.0	6.1	13.7	83.8
5	1.26	4.5	8.4	6.1	86.9	37.18	1.4	4.7	13.2	76.7
6	1.28	5.9	9.4	6.2	88.7	37.29	1.7	5.0	12.7	69.9
7	1.25	3.4	7.2	6.1	87.2	37.28	1.7	4.9	13.8	84.7
8	1.25	3.4	7.2	6.0	83.8	37.85	3.2	6.4	13.8	84.1
9	1.29	7.0	10.3	6.1	85.7	37.51	2.3	5.5	13.7	83.3
10	1.28	6.0	9.5	6.2	89.9	38.03	3.7	6.8	13.8	84.5
avrg		5.1			87.0		2.2			79.3

(*) : floorplan results without constraints.
(area : mm^2, white space : %, run time: sec, area and time increments : %)



(a) circuit 1 in Table 4

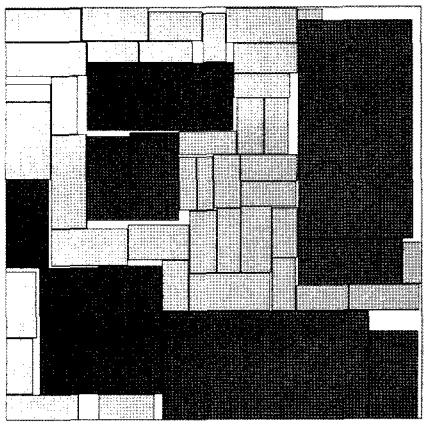


(b) circuit 10 in Table 4

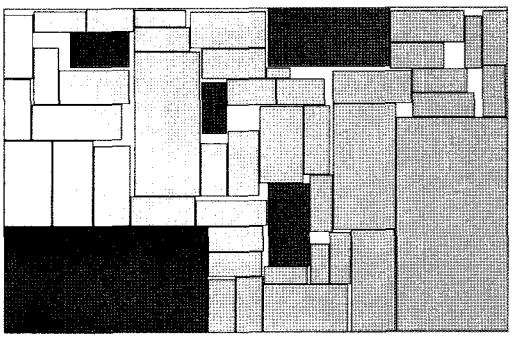
그림 11 Floorplan results of ami49. Pre-placed blocks are drawn as dark-shaded rectangles

실험은 회로에 다수의 고정블록이 포함된 경우에도 본 논문에서 제안한 방법이 효과적임을 보였다.

표 5는 참고문헌의 실험결과와 본 논문에서 제안한 방법의 실험결과를 비교하여 보인 것이다. 첫째 열은 10개의 고정블록을 가진 ami49에 대한 결과이고 실험에 사용된 회로는 [4]로부터 가져온 것이다. 둘째 열과 셋째 열의 실험에 사용된 회로는 각각 4개의 고정블록을 포함한 ami33과 5개의 고정블록을 포함한 ami49 회로이며 [3]에서 가져온 것이다. [4]과 [3]에 보고된 결과와 본 논문에서 제안한 방법의 결과를 비교하면, 면적이 감소되었음을 알 수 있다. 이것은 우리가 제안한 방법이 다른 방법들과 비교할 만한 결과를 효과적으로 생성할 수 있음을 보인 것이다. 그림 12는 [16]에서 가져온 ami49와 [3]에서 가져온 ami49를 논문에서 제안한 방법으로 평면계획한 결과를 보인 것이다. 표 5에서 우리 방법이 다른 방법들 보다 빠르게 평면계획 결과를 생성하지만, 서로 다른 컴퓨터 상에서 수행된 결과이므로 직접적인



(a) ami49(10 pre-placed blocks) from [16]



(b) ami49(5 pre-placed blocks) from [3]

그림 12 Floorplan results of ami49. Pre-placed blocks are drawn as dark-shaded rectangles

표 5 Comparative results to the results in the reference

circuit	# preplaced blocks	results in the literature				ours			
		area	white space	time	ref.	area		white space	time
						area	incr		
ami49	10	38.35	7.6	1004	[4]	37.89	-1.2	6.5	13.7
ami33	4	1.25	7.5	114	[3]	1.22	-2.6	6.7	5.8
ami49	5	37.41	5.2	261	[3]	37.37	-0.1	5.1	11.6

(area : mm^2, white space : %, run time: sec, area and time increments : %)

수행시간 비교는 의미가 없다.

5. 결 론

본 논문에서는 고정블록을 포함한 회로를 배치할 수 있는 CBL 기반의 평면계획 방법을 소개하였다. 제안된 방법은 [23]에서 제안한 고정블록 처리 방법의 문제점을 해결하여 최적해를 생성할 수 있음을 보였다. 그리고 실험 결과를 통하여 우리가 제안한 방법이 효율적이고 효과적임을 보였다.

앞으로의 계획은 본 논문에서 사용된 기법을 응용하여 최근 평면계획에서 중요한 부분으로 부각되고 있는 인접제약조건, 변 제약조건 등과 같은 위상 제약조건들에 적용시킬 수 있는 방법을 연구하는 것이다.

참 고 문 헌

[1] M. Sarrafzadeh, C. K. Wong, An introduction to VLSI physical design, McGraw-Hill Companies, Inc., 1996.

[2] Y. C. Chang, Y. W. Chang, G. M. Wu, and S. W. Wu, "B*-Trees: a new representation for non-slicing floorplans," in Proc. IEEE/ACM DAC, pp. 458-463, 2000.

[3] S. Dong, X. Hong, S. Chen, X. Qi, R. Wang and J. Gu, "VLSI module placement with pre-placed modules and with consideration of congestion using solution space smoothing," IEICE Trans. Fundamentals, Vol.E86-A, No.12, pp. 3136-3147, 2003.

[4] H. Murata, K. Fujiyoshi, and M. Kaneko, "VLSI/PCB placement with obstacles based on sequence pair," IEEE Trans. CAD, Vol.17, No.1, pp. 60-68, 1998.

[5] F. Y. Young, and D. F. Wong, "Slicing floorplans with pre-placed modules," in Proc. IEEE/ACM ICCAD, pp. 252-258, 1998.

[6] Y. Ma, S. Dong, X. Hong, Y. Cai, C. K. Cheng, and J. Gu, "VLSI floorplanning with boundary constraints based on corner block list," in Proc. IEEE/ACM ASP-DAC, pp. 509-514, 2001.

[7] F. Y. Young, D. F. Wong, and H. H. Yang, "Slicing floorplans with boundary constraints," IEEE Trans. CAD, Vol.18, No.9, pp. 1385-1389, 1999.

[8] K. Fujiyoshi, and H. Murata, "Arbitrary convex and concave rectilinear block packing using sequence-pair," in Proc. IEEE/ACM ISPD, pp. 103-110, 1999.

[9] Y. Ma, X. Hong, S. Dong, Y. Cai, C. K. Cheng, and J. Gu, "Floorplanning with abutment constraints and L-shaped/T-shaped blocks based on corner block list," in Proc. IEEE/ACM DAC, pp. 770-775, 2001.

[10] F. Y. Young, H. H. Yang, and D. F. Wong, "On extending slicing floorplans to handle L/T-shaped modules and abutment constraints," IEEE Trans. CAD, Vol.20, No.6, pp. 800-807, 2001.

[11] F. Y. Young, D. F. Wong, and H. H. Yang, "Slicing floorplans with range constraint," IEEE Trans. CAD, Vol.19, No.2, pp. 272-278, 2000.

[12] S. Chen, S. Dong, X. Hong, Y. Ma, and C. K. Cheng, "VLSI block placement with alignment constraints," IEEE Trans. CAS-II, Vol.53, No.8, pp. 622-626, 2006.

[13] H. Xiang, X. Tang, and D. Wong, "Bus-driven floorplanning," IEEE Trans. CAD, Vol.23, No.11, pp. 1522-1530, 2004.

[14] E. F. Y. Young, C. C. N. Chu, and M. L. Ho, "Placement constraints in floorplan design," IEEE Trans. VLSI Systems, Vol.12, No.7, pp. 735-745, 2004.

[15] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," Science, Vol.220, No.4598, pp. 671-680, 1983.

[16] H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence pair," IEEE Trans. CAD, Vol.15, No.12, pp. 1518-1524, 1996.

[17] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, "Module placement on BSG-structure and IC layout application," in Proc. IEEE/ACM ICCAD, pp. 484-491, 1996.

[18] X. Hong, G. Huang, Y. Cai, J. Gu, S. Dong, C. K. Cheng, and J. Gu, "Corner Block List: an effective and efficient topological representation of non-slicing floorplan," in Proc. IEEE/ACM ICCAD, pp. 8-12, 2000.

[19] S. Zhou, X. Hong, T. Cai, C. K. Cheng, and J. Gu, "ECBL: an extended corner block list with solution space including optimum placement," in Proc. IEEE/ACM ISPD, pp. 150-155, 2001.

[20] 강상구, 신정호, 임종석, "Corner Block List를 이용한 개선된 블록 배치 방법", IEEK SOC Design Conference, pp. 263-268, 2003.

[21] P. N. Guo, and C. K. Cheng, "An O-tree representation of non-slicing floorplan and its applications," in Proc. IEEE/ACM DAC, pp. 268-273, 1999.

[22] K. Sakanushi, Y. Kajitani and D. P. Mehta, "The quarter-state-sequence floorplan representation," IEEE Trans. CAS-I, Vol.50, No.3, pp. 376-386, 2003.

[23] S. Dhamdhere, N. Zhou and T.-C. Wang, "Module Placement with Pre-Placed Modules Using the Corner Block List Representation," IEEE ISCAS, pp. I-349-I-352, 2002.

[24] K. Koziminski, "Benchmarks for layout synthesis-evolution and current status," in Proc. IEEE/ACM DAC, pp. 265-270, 1991.

강 상 구

1998년 서강대학교 수학과 학사. 2001년 서강대학교 컴퓨터학과 석사. 2001년 3월~현재 서강대학교 컴퓨터학과 박사과정. 관심분야는 설계자동화 알고리즘, ASIC 설계

임 종 석

1981년 서강대학교 전자공학과 학사. 1983년 한국과학기술원 전기 및 전자공학과 석사. 1989년 Univ. of Maryland, College Park, 전기공학과 박사. 1983년 3월~1990년 8월 한국전자통신연구소 연구원. 1990년 9월~현재 서강대학교 컴퓨터학과 교수. 관심분야는 설계자동화 알고리즘, ASIC 설계