OPTIMAL ASPECT RATIOS OF BUILDING BLOCKS IN VLSI

Shmuel Wimer [1,2] , Israel Koren [3] and Israel Cederbaum [1]

1) Dept. of Electrical Engineering Technion - Israel Institute of Technology, Haifa 32000, Israel
2) IBM Israel Scientific Center, Technion City, Haifa 32000, Israel
3) Dept. of Electrical and Computer Engineering, Univ. of Mass., Amherst MA 01003,
on leave from the Technion, Haifa, Israel

ABSTRACT

The building blocks in a given floorplan may have several possible physical implementations yielding different layouts. This paper discusses the problem of selecting an optimal implementation for each building block so that the area of the final layout is minimized. A polynomial algorithm that solves this problem for slicing floorplans was presented elsewhere, and it has been proved that for general (non-slicing) floorplans the problem is NP-complete. We suggest a branch and bound algorithm which proves to be very efficient and can handle successfully large general non-slicing floorplans. We show also how the non-slicing and the slicing algorithms can be combined to handle efficiently very large general floorplans.

1. INTRODUCTION AND BACKGROUND

Most of the existing algorithms for floorplanning require a completely defined geometry of the building blocks. Floorplanning however, is attempted at the very early stage of VLSI physical design, when there is only a rough estimate of the building blocks geometry. In many cases a good estimate of the building blocks areas is available at this stage, but their exact dimensions can still be varied in a wide range. An example is the register file block consisting of 64 registers in a CPU. This register file can be organized as a 1×64, 2×32 , 4×16 or 8×8 array, and if we consider also the two possible orientations for a single register and the whole file, there are 14 different implementations, as shown in Figure 1. Given the floorplan of a chip, we wish to take advantage of the many possible implementations of its building blocks and search for those implementations yielding a minimum area layout.

In [5] a similar problem has been solved, where it was assumed that the dimensions of each building block can vary continuously in some given interval while its area is assumed to be invariant. In [3] the problem of finding an optimal orientation of the building blocks in slicing floorplan was discussed and an efficient polynomial solution was presented. Its time and storage requirements are $O(b \log b)$, where b is the number of building blocks. Notice that the optimal orientation problem (in which each building block has two possible implementations) is a special case of our problem. Another efficient polynomial algorithm to determine the optimum geometry of the building blocks in slicing floorplans was presented in [2]. There, the geometry of the blocks is constrained by a piecewise linear function, which can approximate any smooth function. Its complexity is $O(k\, b \log b)$, where k is the maximal number of breakpoints in a geometry constraining function. It was also shown in [3] that the optimal orientation problem for general (non-slicing) floorplans is NP-complete, and an integer programming method to find the minimum area layout was presented in [7].

Consequently, the more general problem of determining the optimal dimensions (and not just the optimal orientation) of the building blocks in non-slicing floorplans is NP-complete. Since non-slicing floorplans often occur in practice, we propose in this paper a *branch and bound* algorithm. The proposed algorithm has been implemented and proved to be efficient and capable of handling successfully large floorplans.

2. SLICING FLOORPLANS

A common representation of a floorplan is through a pair of *dual polar graphs* , called the $x-$ graph and the $y-$ graph and denoted by $G(U,E)$ and $H(V,F)$, respectively (e.g. [1], [5]). Figure 2(a) shows a floorplan whose vertical line segments are denoted by u_1 through u_5 and its horizontal line segments by v_1 through v_6. Figure 2(b) shows the corresponding dual polar graphs, one drawn on the top of the other. A vertex in $G(U,E)$ represents a vertical line segment of the floorplan. An arc e directed from u_i to u_j exists if there is a sub-rectangle in the floorplan whose left and right edges lie on the corresponding vertical line segments. The source and the sink of $G(U,E)$ correspond to the leftmost and the rightmost vertical line segments of the floorplan, respectively. $H(V,F)$ is defined similarly for the horizontal line segments. A building block is denoted by B_i , $1 \le i \le b$, and it is assigned a finite set of n_i possible dimensions (x_j^i , y_j^i), corresponding to its various possible implementations. Its area a_i is invariant and thus given by $x_j^i y_j^i = a_i$, $1 \le j \le n_i$. Given the width and height of every building block, we assign them to the corresponding arcs in G and H. Then, the width w and the height h of the layout are determined by the length of the longest paths in G and H, respectively.

In the case of a slicing floorplan the above graphs are series-parallel. A series-parallel graph can be represented by a decomposition tree [4] and Stockmeyer's algorithm [3] is based on it. Each leaf of this tree represents an area in which a certain block must be placed in one of its two possible orientations. Having b blocks, there are 2^b possible configurations of the layout. Among them we are looking for the one that minimizes the area $A = wh$.

Stockmeyer's algorithm starts at the leaves of the decomposition tree. At each step it attempts to combine two smaller blocks into a bigger one, called *super block* . The main idea behind it is that if two super blocks at a lower level can be implemented in n_1 and n_2 different ways, respectively, it is unnecessary to consider all the $n_1 \times n_2$ possible combinations for the resulting super block at the higher level. Instead, it is proved that only $O(n_1 + n_2)$ possible combinations are relevant to the optimal final layout. If at some level k , $1 \leq k \leq \log b$, of the decomposition tree we list all the relevant possible implementations of each super block, the total number of implementations in this list is $O(b)$ and the time to generate them is also $O(b)$, resulting in a total running time and storage requirement of $O(b \log b)$. The super block at the root of the decomposition tree is the entire layout, having a list of $O(b)$ possible implementations (each one is a pair of possible width w and height h). Among the elements in this list we choose the one which minimizes the area $A = wh$.

The above algorithm is also valid when several possible implementations for each B_i are considered rather than only the two orientations. Instead of starting at a leaf with only two possible implementations, we consider B_i to be a super block having as many as desired possible implementations, and proceed the same way as before.

3. ALGORITHM FOR GENERAL FLOORPLANS

Our goal is to assign dimensions to the building blocks B_i 's so that $A = wh$ is minimized. If B_i has n_i possible implementations, the space of all the assignments of dimensions contains $n_1 \times \cdots \times n_b$ states, each one yields some area $A = wh$. These states can be enumerated by an enumeration tree, in which blocks are first assigned to levels, and then at each level we examine all the possible dimensions of the corresponding block. Each node of the enumeration tree corresponds to a partial layout. A path starting at the root and ending at a leaf represents a complete layout. Figure 3(a) depicts a 3-block floorplan, where B_1 , B_2 and B_3 may have 4, 3 and 2 possible implementations, respectively. The full enumeration tree is shown in Figure 3(b), where at each node the dimensions of the corresponding block are indicated. The areas of the partial layouts are written next to the corresponding nodes.

The branch and bound algorithm proceeds as follows. First we determine which block should be considered at a given level of the enumeration tree. Without loss of generality we assume that B_i is considered at the i th level, where the root of the tree is at level 0 and the leaves are reached at level b. At the root we assign to each arc of G and H the smallest length it may have among all the possible implementations of its corresponding block. Then, going downwards from level i to level $i + 1$, appropriate lengths are assigned to the arcs of G and H corresponding to B_{i+1}. When going backwards, the lengths of the arcs are reset to their initial values. At each node we calculate the width w and the height h of the partial layout. When going downwards along a path from the root to a leaf, $A = wh$ is non-decreasing since w and h are non-decreasing functions of the set of already laid out blocks. Let $A_{\min}$ denote the minimum value of $A = wh$ achieved thus far at some leaf of the tree. Then, the enumeration proceeds as follows:

```
begin (branch and bound)
A_min = ∞ ;
Assign initial lengths to the arcs of G and H ;
while not the root is backwards traversed do
  begin if A ≥ A_min
    then backtrack
    else if all possible implementations of the
            current block are exhausted
      then backtrack
      else if a leaf was reached
        then if A < A_min
          then A_min = A and backtrack
          else backtrack
        else forward step ;
  end;
end (branch and bound) ;
```

Notice that when the above procedure is applied to the tree in Figure 3(b), the starred nodes are not traversed. The reason for this is that the area of the partial layout upon traversing their parent node is greater than the area of some complete layout that has been previously examined.

The efficiency of the above branch and bound procedure is affected by the following factors:

1. The value of $A_{\min}$. It is desirable to reduce it as soon as possible, resulting in earlier backtracks.
2. The area $A = wh$ of the partial layout at a node is a lower bound on the area of a complete layout obtained at any leaf reachable from this node. Therefore, if we succeed to raise this lower bound, a backtrack will occur sooner.

3. The order in which the possible dimensions of a block are examined at the corresponding level of the tree.

In the following we discuss each of the above items in greater detail.

3.1 Fast Reduction Of $A_{\min}$

Reaching a leaf with a lower $A_{\min}$, we may consider it as a good starting point for a heuristic search in which we attempt to further reduce the area as much as possible, applying a non-enumerative search. The following search procedure attempts to find a local minimum of the area of a complete layout in the sense that no further reduction can be achieved by changing the dimensions of a single block.

begin (further reduction)
while reduction *do*
  begin for $i = 1$ *to* b *do*
    find (x_j^i, y_j^i), $1 \le j \le n_i$, which minimizes A ;
    if $A < A_{\min}$
      then $A_{\min} = A$;
  end ;
end (further reduction) ;

The above heuristic search must terminate after a finite number of steps since G and H are finite graphs whose arc lengths are selected from a finite set of possible values. When this procedure ends, the search in the branch and bound algorithm continues from the leaf where the further reduction procedure has been invoked. This guarantees that the global minimum can still be achieved.

3.2 Lower Bound On A

In the following we calculate a lower bound on the area that must be added to the partial layout as a result of placing the yet unplaced blocks in their appropriate regions in the floorplan, regardless of the specific dimensions they will be assigned. Then, if in the branch and bound procedure the statement *if* $A \ge A_{\min}$ is replaced by *if* $(A + \text{bound}) \ge A_{\min}$, the backtrack will occur earlier.

Suppose that the algorithm has reached some node at the level i (the dimensions of B_1 through B_{i-1} are determined while B_i is currently considered). Let G_{i-1} and H_{i-1} denote the graphs corresponding to the partial layout of the first $i-1$ blocks, whose width and height are denoted by w_{i-1} and h_{i-1}, respectively. For $i \le j \le b$, let l_j be the length of the longest path in G_{i-1} from the source to the vertex representing the vertical line segment supporting the left edge of B_j, and let r_j be the length of the longest path in G_{i-1} from the vertex representing the vertical line segment supporting the right edge of B_j to the sink. Similarly we denote by t_j and b_j the length of longest paths in H_{i-1} corresponding to the vertices representing the top and the bottom horizontal line segments supporting B_j. The area available in the partial layout for the placement of any yet unplaced block B_j, regardless of its final implementation, is given by $(w_{i-1} - r_j - l_j) \times (h_{i-1} - b_j - t_j)$. If $a_j > (w_{i-1} - r_j - l_j) \times (h_{i-1} - b_j - t_j)$, the embedding of B_j (in any of its possible implementations) requires at least $a_j - (w_{i-1} - r_j - l_j) \times (h_{i-1} - b_j - t_j)$ additional area in the complete layout. Therefore, a lower bound on the total additional area required to complete the layout is given by:

$$\sum_{j=i}^{b} \max\{0,\ a_j - (w_{i-1} - r_j - l_j) \times (h_{i-1} - b_j - t_j)\} .$$

This additional area provides a very effective bound that can be calculated at any downwards traversed node of the search tree.

3.3 Ordering The Possible Implementations Of The Blocks

We first derive a backtracking condition which can be invoked in the general implicit enumeration scheme. Let v be some downwards traversed node of the tree, let B_i be the building block considered at v (with some implementation (x_k^i, y_k^i)) and let $T(v)$ denote the subtree rooted at v. The branch and bound algorithm enumerates (implicitly) all the paths of $T(v)$. For a node $\mu \in T(v)$ we denote by $G(\mu)$ and $H(\mu)$ the graphs corresponding to the partial layout at this node, and by $\Gamma(\mu)$ and $\Delta(\mu)$ their longest paths, respectively. (For the sake of clarity we assume that the longest paths are unique, but the following discussion holds also for multiple critical paths.) Then, the following lemma provides a backtracking condition.

Lemma 1: Let M be the set of nodes which have been reached from v such that no further forward step was taken at them (the "front" of the traversed portion of $T(v)$), and let $\mu \in$ M be a terminating node of a path starting at v. Assume also that for every $\mu \in$ M the pair (e_i, f_i) of dual arcs corresponding to B_i (whose length at v is given by (x_k^i, y_k^i)), satisfies $e_i \notin \Gamma(\mu)$ and $f_i \notin \Delta(\mu)$. Then, the remaining implementations of B_i at v need not be further considered, and backtracking at v can be performed. ■

The proof is found in [6]. Lemma 1 states that if the result obtained by examining $T(v)$ is independent of the specific implementation (x_k^i, y_k^i) of B_i which is considered at v, then the examination of all the subtrees whose roots are the remaining implementations of B_i, cannot yield further reduction of $A_{\min}$. Notice that all the subtrees rooted at the nodes corresponding to different implementations of B_i are isomorphic, except the roots themselves.

The backtracking condition established in Lemma 1 is independent of the order in which the possible implementations of a building block are examined. However, a particular order of examination may yield an earlier backtracking. In what follows we suggest such an ordering.

A series of pairs $(x_1, y_1),...,(x_n, y_n)$ is said to be in an *increasingly interlaced order* if $x_i \leq x_j$ for odd i and for all $j > i$, and $y_i \leq y_j$ for even i and for all $j > i$. For example, the series (1,64), (64,1), (2,32), (32,2), (4,16), (16,4), (8,8) is in an increasingly interlaced order. Following the same arguments as in Lemma 1, we can derive the following backtracking condition.

Lemma 2: Let the series of B_i's implementations be in an increasingly interlaced order, and let v be a node corresponding to an odd (even) numbered implementation of B_i. Let M be the set of nodes which have been reached from v such that no further forward step was taken at them. Assume that for every $\mu \in M$ there exists $f_i \notin \Delta(\mu)$ $(e_i \notin \Gamma(\mu))$. Then, the remaining implementations of B_i at v need not be further considered, and a backtrack at v can take place. ■

The backtracking conditions established in Lemma 2 can be easily checked upon traversing a node backwards. Notice that when the series of blocks' implementations is in an increasingly interlaced order, the conditions of Lemma 2 provide a stronger backtrack criterion than Lemma 1 does, since the existence of the conditions in Lemma 1 imply the existence of those in Lemma 2.

4. COMBINING THE GENERAL AND THE SLICING ALGORITHMS

Stockmeyer's algorithm [3] is very efficient, but fails to handle general floorplans. On the other hand, although the branch and bound algorithm as previously presented is general, it is time consuming for floorplans with many blocks. Therefore, we propose to combine them and devise an algorithm which can handle very large general floorplans. To this end, we first decompose G into its maximal series-parallel components. This can be done in linear time as described in [4]. A maximal series-parallel component satisfies:

Lemma 3: Let G_i, $1 \leq i \leq q$, be a maximal series-parallel component of G, and let H_i be its corresponding portion of H. Then, the pairs (G_i, H_i), $1 \leq i \leq q$, correspond to the maximal slicing portions of the floorplan given by (G,H). ■

Let b_i be the number of arcs in G_i and H_i, $1 \leq i \leq q$. To each of the components G_i we apply Stockmeyer's algorithm, thus obtaining a list of $O(b_i)$ relevant possible implementations. We may look now at each slicing component as a super block having several possible implementations. Next, we replace G_i and H_i by a single arc in G and H, respectively. We then apply the branch and bound algorithm to the new G and H, in which the number of arcs has been reduced from $b = b_1 + \cdots + b_q$ to q.

The following example demonstrates the effectiveness of the combined algorithms. We shall see how the size of the search tree is significantly reduced.

Example 1: Figure 4(a) depicts a 20 block general floorplan, where the thick lines enclose the maximal slicing portions (corresponding to the maximal series-parallel components in G and H). Let each building block have 8 possible implementations. The entire problem cannot be solved by Stockmeyer's algorithm. Employing the branch and bound algorithm, a tree having $8^{20} \simeq 10^{18}$ leaves must be considered. If we first apply Stockmeyer's algorithm to the slicing components, we obtain about 32 possible implementations for each super block that are relevant for further consideration. Then, applying the branch and bound algorithm to the super blocks, a tree with only $32^5 \simeq 10^{7.5}$ will have to be considered. ■

In the above discussion we combined the two algorithms hierarchically. At the lower level we employed Stockmeyer's algorithm to the maximal slicing components and then at the higher level we employed the branch and bound algorithm. Sometimes it may happen that such a hierarchy does not exist. An example is the floorplan given in Figure 4(b). Here, it is impossible to find any slicing components at the lower level. However, the thick lines define a slicing structure at the higher level and we wish to take advantage of this structure. As in the former discussion, we shall employ the two algorithms hierarchically, but in this case the branch and bound at the lower level and Stockmeyer's algorithm at the higher level. To establish this hierarchy more formally, some definitions are in order. Let G_1 and G_2 be floorplan graphs with sources s_1 and s_2, respectively, and sinks t_1 and t_2, respectively. We say that G is a *series composition* of G_1 and G_2 if $t_1(t_2)$ and $s_2(s_1)$ are identified. G is called a *parallel composition* of G_1 and G_2 if s_1 is identified with s_2 and t_1 is identified with t_2. Obviously, the result of series or parallel composition of floorplan graphs is a new floorplan graph. A floorplan graph is called *elementary non series-parallel* if it cannot be obtained as a series or parallel composition of floorplan graphs. Following Lemma 3, we may prove that

Lemma 4: Let G_i, $1 \leq i \leq q$, be a maximal elementary non series-parallel component of G, and let H_i be its corresponding portion in H. Then, the pairs (G_i, H_i), correspond to a maximal elementary non slicing portions of the floorplan given by (G,H). ■

Lemma 4 suggests decomposing of G and H into their maximal elementary non series-parallel components, employ the branch and bound procedure to each component individually and then proceed with Stockmeyer's algorithm for the series-parallel structure at the higher level. The following example shows the advantage of this approach.

Example 2: Let each building block in Figure 4(b) have 8 possible implementations. The entire problem cannot be solved by Stockmeyer's algorithm. Employing the branch and bound algorithm, a tree having $8^{20} \simeq 10^{18}$ must be examined. If we first employ the branch and bound algorithm to the maximal elementary non series-parallel components (enclosed by the thick lines), we shall consider a tree of $8^5 \simeq 10^{4.5}$ leaves for each component. We next employ Stockmeyer's algorithm to the combined structure in which there are four blocks, each one has at most 8^5 possible implementations (practically, very few complete layouts are relevant to the higher level). Then, according to the complexity of Stockmeyer's algorithm, the total time and memory required at this step is bounded by $O[4 \times 8^5 \log(4 \times 8^5)] \simeq O(10^{6.3})$. ■

It is not necessary for the smallest area implementation of a maximal elementary non series-parallel component to participate in the entire layout which occupies minimum area. However, if two different implementations of a non series-parallel component have width w_1 and w_2 , respectively, and height h_1 and h_2 , respectively, satisfying $w_2 \geq w_1$ and $h_2 \geq h_1$, the latter implementation need not be considered at the higher level. The branch and bound algorithm presented in section 3 is slightly modified to produce all the relevant implementations of the components at the lower level. Its output is a list of (w,h) pairs which are the relevant implementations. Initially this list is empty. At every node of the enumeration tree we check whether the width and the height of the corresponding partial layout dominate the width and the height of some pair in the list. If this happens to be the case, a backtrack takes place. Otherwise, the algorithm proceeds in a forward step. Whenever a leaf is reached, the corresponding pair is added to the list, while every existing pair dominating the new pair is deleted from the list.

5. COMPUTATIONAL RESULTS

Figure 5 depicts a 24 block floorplan, where the possible implementations of each block are listed as (x,y) pairs within the corresponding region. There is a total of 2.03×10^{16} possible configurations (equivalent to a 50 block optimal orientation problem), and the algorithm proposed in this paper searches for the one yielding the smallest area. Summing up the areas of the individual blocks yields 1024 area units. Notice that if the blocks take the highlighted implementations in Figure 5, the area of the resulting layout is of size 1024, which is obviously the desired minimum.

The branch and bound algorithm was run four times. Initially, the basic algorithm without any heuristic was employed. In the second run, the further area reduction discussed in Section 3.1 was introduced. Next, the improved area bound discussed in Section 3.2 was supplemented. Finally, the backtracking conditions based on the increasingly interlaced order of block implementations, discussed in Section 3.3, were incorporated. Table 1 summarized the results and demonstrates the effectiveness of the above heuristics. To reduce the dependence of the comparison on the specific software implementation and the computer system used, the results are given in terms of the ratio between the number of visited nodes and the total number of leaves in the search tree.

6. CONCLUSIONS

Given a general (non-slicing) floorplan and several possible physical implementations for each building block, this paper presented a practical algorithm that determines the implementation of each building block such that the area of the entire layout is minimized. Although this problem is NP-complete, the suggested branch and bound algorithm handles successfully large floorplans. It was shown that one can take advantage of the slicing structures which are usually found in general floorplans, by combining the branch and bound algorithm with the known polynomial algorithms for the slicing case. This combination further increased the size of the problems that can be solved.

Acknowledgement: Discussions with R. Y. Pinter are gratefully acknowledged.

REFERENCES

[1] Ciesielski M. J. and Kinnen E., **"Digraph relaxation for 2-dimensional placement of IC blocks,"** *IEEE Trans. on CAD of Integrated Circuits and Systems,* Vol. 6, 1987, pp. 55-66.

[2] Otten R.H.J.M, **"Efficient floorplan optimization,"** *ICCD83 - IEEE Intl. Conf. on Computer Design,* 1983, pp. 499-502.

[3] Stockmeyer. L., **"Optimal orientations of cells in slicing floorplan designs,"** *Information and Control,* Vol. 57, 1983, pp. 91-101.

[4] Valdes J., Tarjan R. E. and Lawler E., L., **"The recognition of series parallel digraphs,"** *SIAM J. Comput.,* 1982, pp. 298-313.

[5] Wimer S., Koren I. and Cederbaum I., **"Floorplans, planar graphs and layouts,"** *to appear in IEEE Trans. on Circuits and Systems,* March, 1988.

[6] Wimer S., **D.Sc. thesis,** *Department of Electrical Engineering, Technion - Israel Institute of Technology,* 1988.

[7] Zibert K. and Saal R., **"On computer aided hybrid circuit layout,"** *IISCS74 - IEEE Intl. Symp. on Circuits and Systems,* 1974, pp. 314-318.

No heuristics	Further area reduction	Improved area bound	Increasingly interlaced order
2.38×10^{-11}	5.32×10^{-12}	1.84×10^{-12}	4.95×10^{-13}

Table 1: Results of 24 block floorplan.

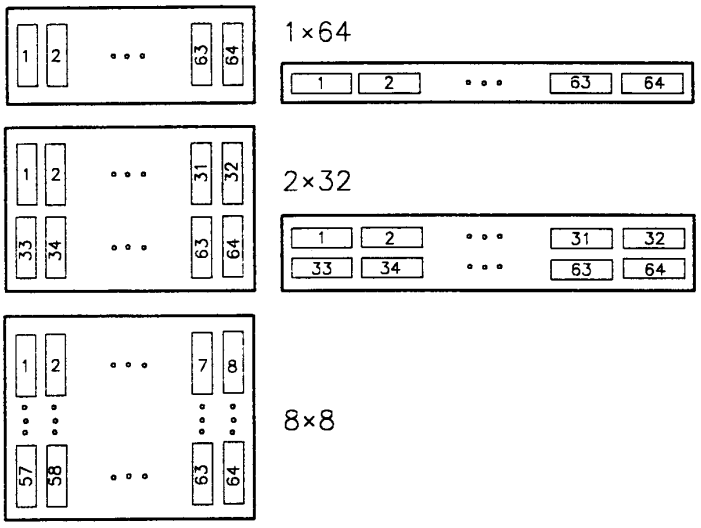

Figure 1. Some implementations of a register file.

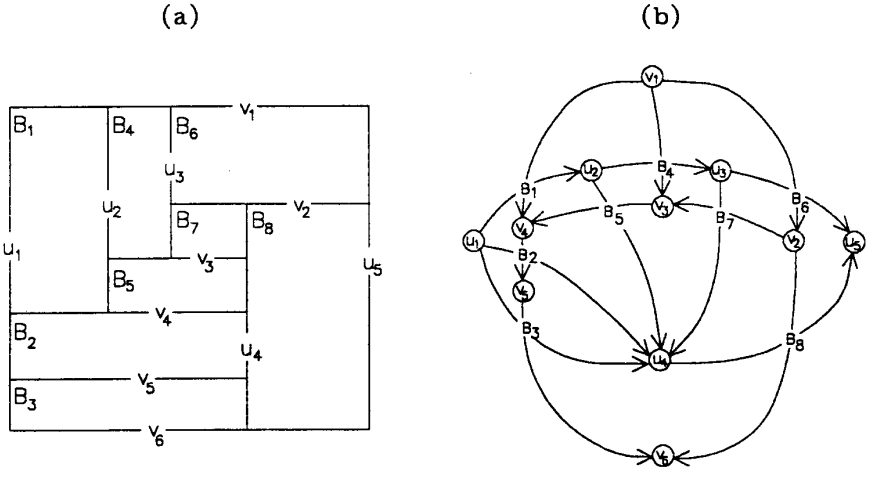

Figure 2. A floorplan and its graph representation.

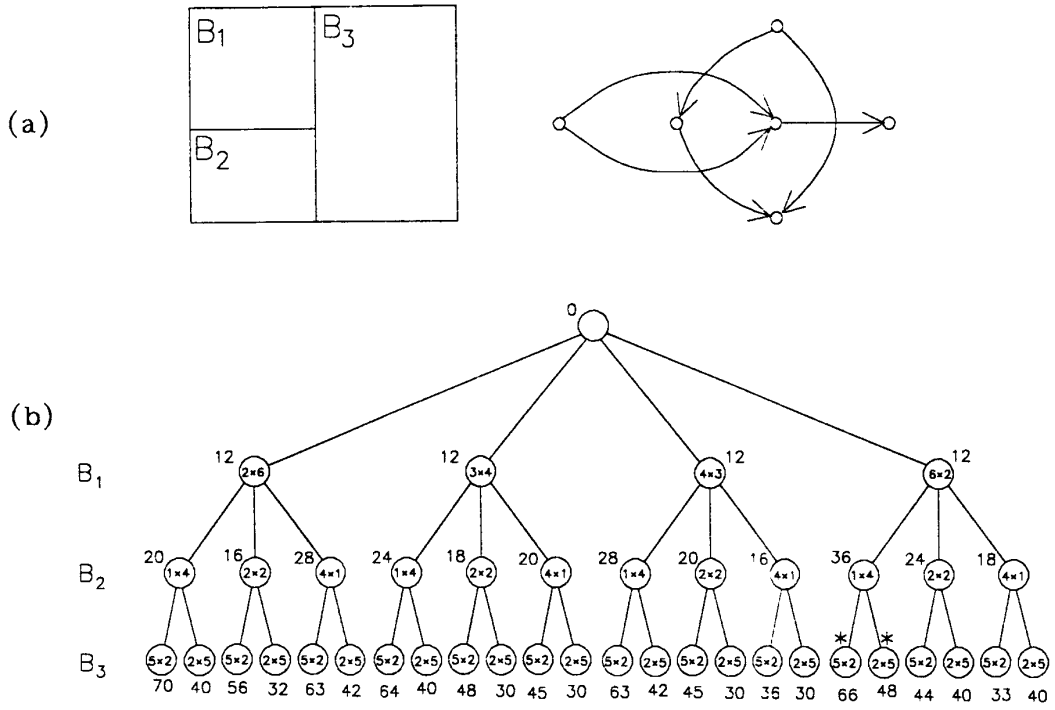

Figure 3. A floorplan and its search tree.

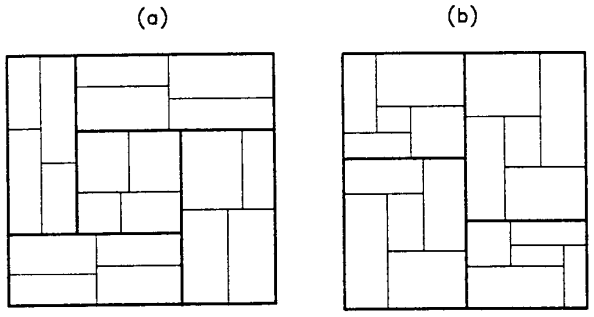

Figure 4. Combination of slicing and non-slicing structures.

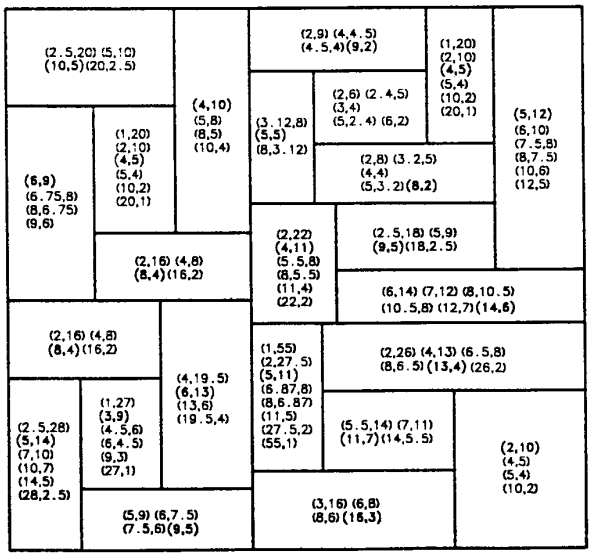

Figure 5. 24 block floorplan.