VLSI DESIGN
1999, Vol. 10, No. 1, pp. 71–86
Reprints available directly from the publisher
Photocopying permitted by license only

Empirical Study of Block Placement by Cluster Refinement

JIN XU[a], PEI-NING GUO[b] and CHUNG-KUAN CHENG[b,*]

[a] *Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134;*
[b] *Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093-0114*

(Received 7 September 1998; In final form 20 November 1998)

In this paper we propose an efficient cluster refinement approach for macro-cell placement. The algorithm selects a cluster of blocks dynamically, and finds an optimal solution for all the blocks in the cluster simultaneously. This is different from previous zone refinement approach which optimizes the allocation of one single block in each operation. Experimental results on the MCNC benchmark circuits show that the approach achieves excellent area utilization while minimizing the wire length at the same time.

Keywords: VLSI, layout, physical design, placement

1. INTRODUCTION

Placement of blocks on a 2*D* surface is one critical process in VLSI layout design. The major objectives are chip area and wire length minimization. Since the number of possible placements increases explosively with the number of blocks, even subsets of the problem have been shown to be NP-complete or NP-hard [13].

Onodera *et al.* [13] presents a building block placement approach which employs a branch-and-bound strategy to search for an optimal solution within the whole solution space of size $2^{n(n+2)}$. The maximum number of blocks which can be placed in a reasonable amount of CPU time is around six. Large problems are decomposed to reduce the number of blocks below the manageable limit, and a placement is constructed hierarchically in a bottom-up manner.

H. Murata *et al.* [11] introduces a *P*-admissible solution space of size $(n!)^2\, 8^n$, where n is the total number of blocks, and applies a simulated annealing method to search for a good solution.

Shin *et al.* [15] applies zone refinement technique in IC layout compaction. Blocks are peeled off row by row from the precompacted layout, moved across an open zone, and reassembled at the other end of the zone in a denser configuration. They

*Corresponding author. Tel.: (619)534-6184, Fax: (619)534-7029, e-mail: kuan@cs.ucsd.edu sweep across the zone to optimize the allocation of one single block in each operation.

Note that the above approaches can be applied to general structures. For slicing structure, Yamanouchi *et al.* [18] proposes a partial clustering and module restructuring algorithm.

In this paper, we present a new cluster refinement approach, which consists of the following three parts:

(1) sequential cluster selection which reduces the complexity of the problem to a manageable degree;
(2) cluster optimization which employs an efficient branch-and-bound strategy to search for an optimal solution for all the blocks in the cluster;
(3) adaptive cluster overlapping which explores larger solution space to achieve better results.

To demonstrate the efficiency of the algorithm, we apply our algorithm to the MCNC benchmark circuits. Experimental results show that the algorithm obtains excellent area utilization while minimizing the wire length at the same time.

The following sections in this paper are organized as follows. We formulate the problem in Section 2. Section 3 describes the details of the algorithm. Then the complexity analysis and experimental results are given in Section 4 and Section 5 respectively. Finally, Section 6 presents concluding remarks.

2. PROBLEM DESCRIPTION

Inputs of the placement problem are

- a set of blocks with fixed geometries and fixed pin positions.
- a set of nets specifying the interconnections between pins of the blocks.
- a set of pads (external pins) with fixed positions.
- a set of user constraints, *e.g.*, block position/orientation, critical nets, if any Given the input specifications, the objective of the problem is to find the positions and orientations of each block, so that the chip area and wire length between blocks are minimized while satisfying all the given constraints.

Since it is impractical to calculate the exact wire length at placement stage where detailed routing has not yet been carried out, we estimate the length of each net as one-half of the perimeter of the minimum bounding box of the net.

The objective function, which measures the quality of the resulting placement, can be expressed as follows,

$$E = C_1 \times \text{ChipArea} + C_2 \times \text{WireLength}$$

where C_1 and C_2 are the constant weights for chip area and wire length respectively.

3. CLUSTER REFINEMENT ALGORITHM

Cluster refinement algorithm intends to improve over traditional zone refinement algorithm by giving more flexibility on cluster selection. With the capabilities of cluster optimization and overlapping, cluster refinement algorithm provides a better approach for block placement. The detail description of cluster refinement algorithm is given as follows.

3.1. Overall Algorithm

Zone refinement is a technique used in the purification process of crystal ingots. It provides a general framework for reducing the total number of blocks to the degree which can be handled at one time.

However, the traditional zone refinement algorithm [15] probably cannot find the optimal solution due to the predetermined order of the blocks. Figure 1 illustrates an example. If block A is placed in its best position, which is also the best position for block B. It is impossible for block B to be placed in its best position if we place block A first. Therefore, the better solution is prevented in this case.

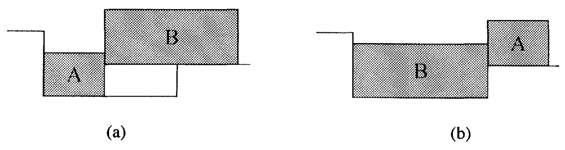


FIGURE 1 Illustration of placement by different orders. (a) *A* first, (b) *B* first.

The major contribution of our work is that we select a cluster of blocks dynamically, employ a branch-and-bound strategy to find an optimal solution for all blocks, then move the blocks in the cluster simultaneously.

Figure 2 shows the situation when cluster refinement is in progress. A placement consists of two regions of blocks, separated by a zone called a 'gap'. The lower bound of the top region forms the 'ceiling' profile, while the upper bound of the bottom region forms the 'floor' profile, as illustrated by the bold lines. The 'gap' profile is obtained by 'deducting' the ceiling from the floor profile.

When cluster refinement starts, all blocks are in the ceiling region, while the floor region and the cluster are totally empty. First a cluster of blocks is selected by the cluster selection algorithm, and is 'peeled off' from the ceiling. Then the cluster optimization algorithm is applied to the cluster, and the cluster is placed onto the floor based on the results obtained. A new cluster is obtained by the cluster overlapping algorithm and the cluster

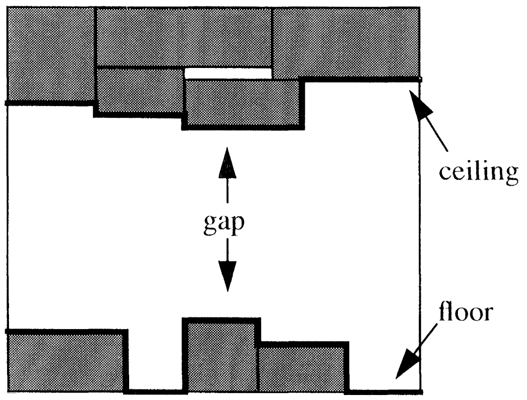


FIGURE 2 Cluster refinement in progress.

optimization algorithm is applied to the new cluster. The algorithm loops until the ceiling is empty.

Figure 3 shows the outline of the algorithm. The algorithm improves an initial placement along one direction, alternates the optimizing direction and begins the next iteration. It may iterate many times until no improvement is achieved, or a given number of iterations is reached.

3.2. Cluster Selection

Our cluster selection algorithm is based on the constraint graphs [15]. It selects a cluster adaptively and sequentially, according to the current partial placement. We first give the definition of neighbors, then present the cluster selection algorithm.

Definition Two blocks are *neighbors* if they are adjacent to each other in either the horizontal or the vertical constraint graph.

Therefore, two blocks are *horizontal neighbors* if they are adjacent in horizontal constraint graph, *vertical neighbors* if adjacent in vertical constraint graph.

For example, in Figure 4(a), the neighbors of block 1 are block 2(right), block 3(above) and block 4(above), based on the constraint graphs shown in Figure 4(b), where block 2 is its horizontal neighbor, while blocks 3 and 4 are its vertical neighbors.

The algorithm identifies the block which dominates the minimum gap distance to be the 'critical' block, then build up a cluster of size k with this block and its $k-1$ neighbors. Figure 5 shows the outline of the cluster selection algorithm.

In Figure 4(a), block 1 is identified first and a cluster of size 4 is built up with it and its three neighbors, blocks 2, 3 and 4.

Note that the cluster size k here is actually the upper bound. The algorithm can adopt flexible cluster size, depending on the number of neighbors of the critical block. For example, in Figure 6, if $k = 4$, but the critical block 1 only has 2

```
ClusterRefinement
{
    Select a compaction direction;
    for(n=0; n<IterationNumber; n++)
    {
        Construct the ceiling;
        Initialize the floor;
        while(ceiling not empty)
        {
            ClusterOverlapping;
            Update the ceiling;
            ClusterOptimization;
            Update the floor;
        }
        Alternate the compaction direction;
    }
}
```

FIGURE 3 Outline of the algorithm.

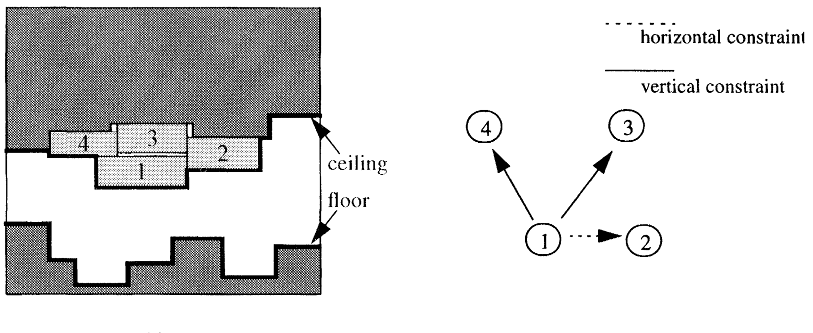


FIGURE 4 Cluster selection in progress. (a) Cluster selected (b) Constraint graphs for block 1.

neighbors, block 2 and 3, the cluster obtained will be of size 3.

3.3. Cluster Optimization

In order to find an optimal solution, we evaluate all of the possible configurations on the selected cluster. Because the number of possible combinations increases exponentially with the number of blocks in the cluster, we employ a branch-and-bound technique to explore the solution space effectively.

3.3.1. Branching Operations

We enumerate the state space as a tree whose nodes correspond to partial placements for some blocks in the cluster. The successors of a node correspond to the implementation of the blocks to be considered next. A path from the root to a leaf represents a complete placement for the cluster.

```
ClusterSelection(k)
{
    Identify a critical block u;
    Add u to the cluster;
    Sort u's horizontal neighbors in y dimension from low to high;
    Sort u's vertical neighbors in x dimension from right to left;
    Append vertical neighbor list to horizontal neighbor list;
    repeat until k blocks are selected or u has no neighbors left
    {
        Find the first neighbor v;
        if v has no vertical neighbors below it
            add v to the cluster;
        Delete v from u's neighbor list;
    }
}
```

FIGURE 5 Cluster selection algorithm.

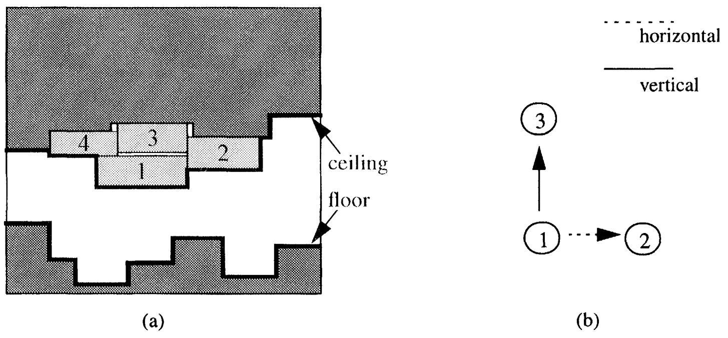


FIGURE 6 Cluster size k as a upper bound. (a) Cluster selected, (b) Constraint.

3.3.2. Order

Suppose k blocks are selected in the cluster and are numbered 1, $2, \ldots, k$. To find the best order of placing the blocks, we try all of their combinations, *i.e.*, $k!$ permutations, as shown in the recursive permutation tree in Figure 7.

3.3.3. Rotation

Each block may be specified in any one of eight orientations, *i.e.*, two rotations and four reflections (Fig. 8). We specify one rotation out of two for each block when placing each block, because rotation is closely related to the final chip area. Since reflection is only directly related to wire length, we try the four reflections for each block only when making estimates to reduce the total wire length.

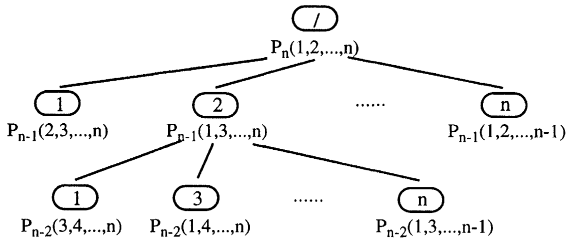


FIGURE 7 Recursive permutation tree.

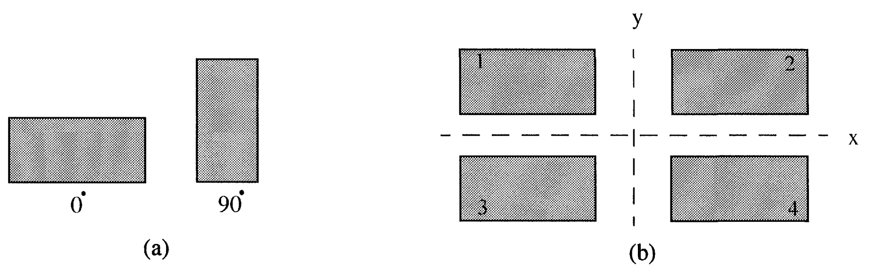


FIGURE 8 Orientation of a block. (a) Rotation, (b) Reflection.

3.3.4. Virtual Grid

When we search for the optimal positions for a block in the cluster, we have to try all of the possibilities, *i.e.*, all the virtual grids created by the corners of blocks, including those which have been placed onto the floor, and which are still in the ceiling, as illustrated in Figure 9.

3.4. Bounding Operations

The efficiency of the branch-and-bound strategy depends a lot on the bounding technique. By pruning unpromising branches in the decision tree, we can explore the solution space much more efficiently.

3.4.1. Corners Only

Assume we have k blocks numbered by $\{1, 2, \ldots, k\}$, the permutation of k blocks can be expressed as

$$P_1, P_2, \ldots, P_i, \ldots, P_k$$

where P_i is the number of the i-th block in the permutation. Given a permutation, we can reduce the number of virtual grids to be evaluated. For instance, when we are placing two neighboring blocks P_1, P_2 in a permutation, we have to evaluate all of the virtual grids for the block P_1 first, then for the P_2. When their order becomes P_2, P_1 in a new permutation, after the block P_2 has been placed, it creates some new grids in the floor. Then to place the block P_1, the only virtual grids we have to evaluate are those new grids, because they are the only grids that could not be evaluated when placing P_1 in the previous permutation. Therefore, the following important theorem holds.

THEOREM *When searching for the optimal position for the i-th block in a permutation, we only have to evaluate the new corners created by the* $(i-1)$*th block if*

$$P_i < P_{i-1}$$

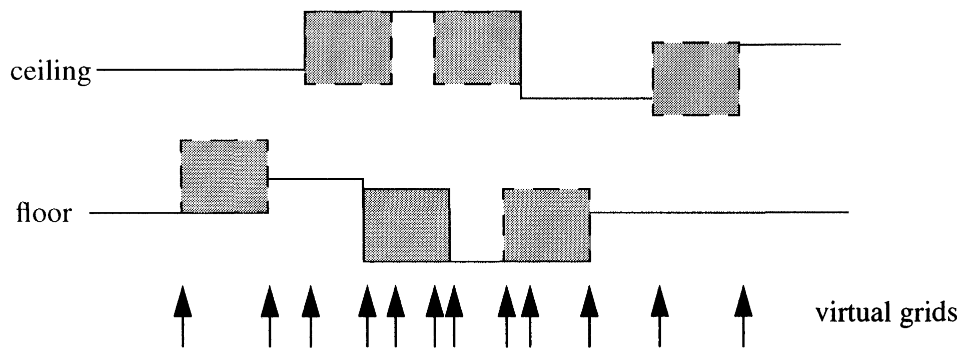


FIGURE 9 Virtual grid.

In general, we have to search for all of the virtual grids to place a block. The theorem above suggests that we only need to evaluate those virtual grids in the range from the leftmost to the rightmost position of the previous block (Fig. 10), because all other possibilities have been covered when placing the blocks by other permutations. This greatly reduces the search space and let the algorithm prune unpromising branches and reach the optimal solution much more efficiently.

For example, if $P_i < P_{i-1}$, after P_{i-1} has been placed as shown in the figure above, to place P_i, the number of virtual grids we have to evaluate is only two (Fig. 10).

3.4.2. Lower/Upper Bound

We first place each block in the cluster at its original x-position, and calculate the cost functions and record their values and the x-positions as the current best known solution. Those values, along with the constraints given, provide lower/upper bounds for the placements obtained in the search process later. The cost functions used in this paper are as follows: gap distance, chip width, chip length, dead space created, wire length.

Each node in the decision tree corresponds to a partial placement in which only rotations or positions of some blocks in the cluster are determined, and only permutation among those blocks is established, while those of the others have not yet been established. Associated with the nodes, are the partial cost values of the corresponding placement. If any one of them exceeds the corresponding lower/upper bound, we can say that

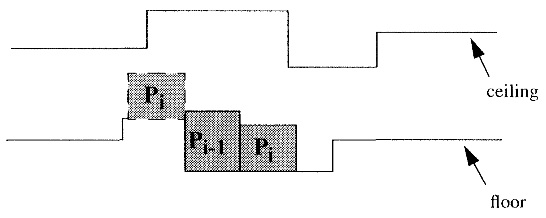


FIGURE 10 Corners only.

the branch is not promising, the search process along the branch will be terminated.

When the search process along a branch reaches a leaf node, a complete placement for the cluster is obtained. If the cost values of the placement are better than the current best known solution, we update the values and save the current x-positions as the best known solution.

Figure 11 illustrates the final decision tree we have to search for when $k = 4$.

3.5. Cluster Overlapping

After cluster optimization, the best positions and order for placing the blocks in the cluster have been obtained. We can place only some of the blocks in the cluster onto the floor, while replacing the other blocks again to find better solutions. Therefore, the first l ($\leq k$) blocks in the cluster, in the best order obtained, are deleted from the cluster, and placed onto the floor in their corresponding best positions, while the other $k - l$ blocks are still kept in the cluster. The algorithm then selects another l blocks from the ceiling. These l new blocks, together with the $k - l$ original blocks, form a new cluster of size k, which has $k - l$ blocks overlapping with the original cluster. The cluster optimization algorithm is applied to the new cluster as described in the previous sections.

For example, if $k = 4$, $l = 2$, block 1, 2, 3, 4 are the four blocks in the cluster, the best order obtained for placing the four blocks are 1, 3, 4, 2, and the corresponding best positions are as shown in Figure 12(a). The first two blocks 1 and 3 are placed onto the floor in their best positions, and deleted from the cluster, while blocks 2 and 4 are still kept in the cluster. The algorithm then gets another two blocks A and B from the ceiling, and builds up a new cluster by the four blocks 2, 4, A, B (Fig. 12(b)). A better result will be achieved after cluster optimization, as illustrated in Fig. 12(c).

Figure 13 shows the outline of the cluster overlapping algorithm.

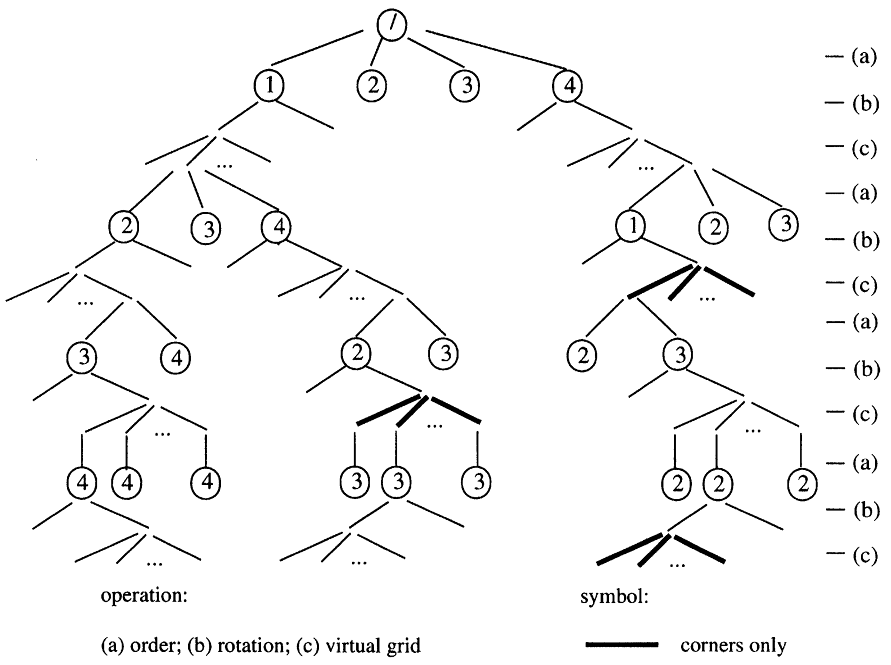


FIGURE 11 Decision tree when $k = 4$.

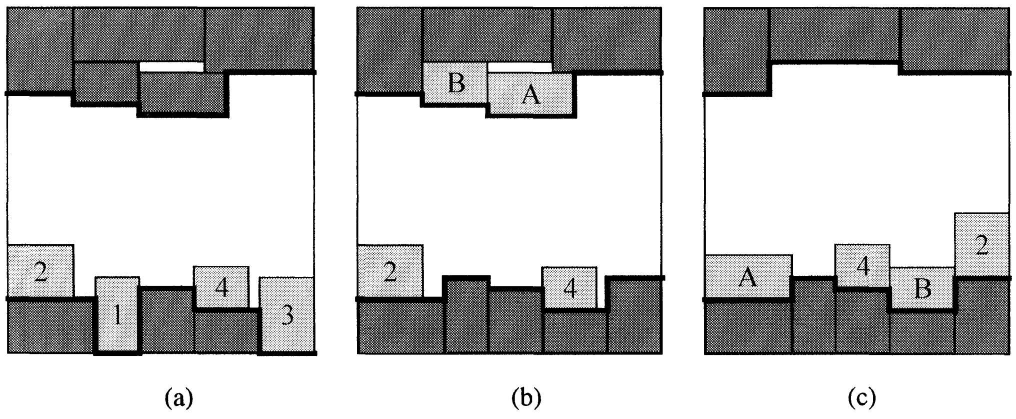


FIGURE 12 Cluster overlapping.

```
ClusterOverlapping(k,l)
{
    if(cluster not empty) {
        Delete the first l blocks by the best order found;
        ClusterSelection(l);
    }
    else
        ClusterSelection(k);
}
```

FIGURE 13 Cluster overlapping algorithm.

4. COMPLEXITY ANALYSIS

We will start investigating the complexity of our algorithm by adopting the well-known analysis of the traditional zone-refinement (*Z-R*) algorithm [15]. First we discuss the complexity of constraint graphs and the traditional *Z-R* algorithm, and then analyze our extension parts of clustering and branch-and-bound algorithms.

4.1. Complexity of the Constraint Graphs and *Z-R* Algorithm

Traditional *Z-R* algorithm deals with n blocks and utilizes two necessary data structures: constraint graphs and ceiling-floor-gap relations. These structures are maintained throughout the program and updating them are the key operations of the entire algorithm.

Shin *et al.* [15] gives a detail proof that the initial construction of the structures takes $O(n^2)$ and that the updating process needs $O(n)$ for each block moving. A complete pass of *Z-R* needs $O(n)$ update operations resulting in an overall complexity of $O(n^2)$.

4.2. Complexity of Cluster Refinement Algorithm

We assume k blocks per cluster and m virtual grids for each cluster moving. The bounding operations give a enormous amount of pruning from the original solution space of size $O(k!m^k)$. The following is the complexity analysis of our algorithm.

LEMMA 1 *Given a permutation* $(p_1, p_2, \ldots, p_k)$, *the branches at level i is*

$$b_i = \begin{cases} m, & \text{if } i = 1 \vee p_{i-1} < p_i \\ 2, & \text{otherwise} \end{cases}$$

The number of branches in each node is either m for new pattern of permutations, or 2 when it is corner-only because some previous permutations have covered most of the virtual grids already.

LEMMA 2 *Given a permutation* $(p_1, p_2, \ldots, p_k)$, *the total number of branches is*

$$B(p_1, p_2, \ldots, p_k) = \prod_{i=1}^{k} b_i.$$

LEMMA 3 *Given a cluster* $(1, 2, \ldots, k)$, *the total number of branches in its decision tree is*

$$\sum_{(p_1, p_2, \ldots, p_k) \in \mathit{permutation}} B(p_1, p_2, \ldots, p_k).$$

For most cases, the cluster size k is equal to or less than 5 and m is about $O(\sqrt{n})$. Table I gives the complexity when k is less than or equal to 5.

LEMMA 4 *Given k and l, the overall complexity of the algorithm will increase by a constant factor k/l from the cluster refinement without overlapping algorithm.*

Thus, the overall complexity of the cluster refinement algorithm will be $O(2^k m^k n^2)$. And for small k, it becomes $O(n^{2+k/2})$.

4.3. Comparison to Other Approaches

Cluster refinement takes the advantages of the traditional *Z-R* algorithm and utilizes exhaustive search in local area to improve it. Extra branch-and-bound searching increases the complexity by the factor of $O(2^k m^k)$ to the traditional *Z-R* algorithm's $O(n^2)$.

Onodera's topological relationship approach [13] takes $O(2^{n(n+2)})$ to find the optimal solution, and can only handle up to six blocks as mentioned by the author. Another approach, Murata's

TABLE I Cluster size and complexity

k	number of branches
1	m
2	$m^2 + 2 \cdot m$
3	$m^3 + 4 \cdot 2 \cdot m^2 + 2^2 \cdot m$
4	$m^4 + 11 \cdot 2 \cdot m^3 + 11 \cdot 2^2 \cdot m^2 + 2^3 \cdot m$
5	$m^5 + 26 \cdot 2 \cdot m^4 + 66 \cdot 2^2 \cdot m^3 + 26 \cdot 2^3 \cdot m^2 + 2^4 \cdot m$

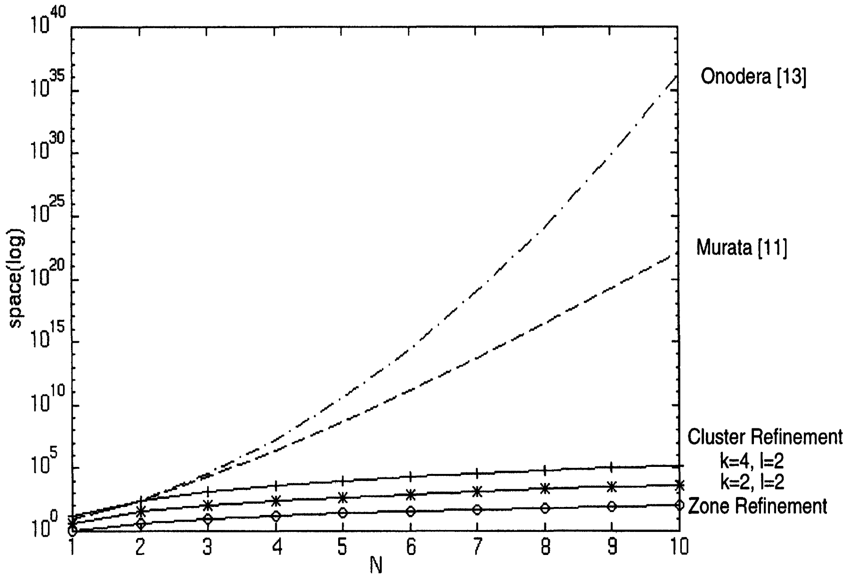


FIGURE 14 Comparisons.

sequence-pair [11] has solution space of size $O(8^n(n!)^2)$, and only a small fraction of the whole space is searched by their simulated annealing method.

Figure 14 shows the comparison of various approaches.

5. EXPERIMENTAL RESULTS

To examine the efficiency of the proposed algorithm, we apply our algorithm to the MCNC benchmark circuits. The algorithm is implemented in *C* and executed on a Sun Sparc20 workstation.

5.1. Initial Placement

The algorithm may start from any initial configuration. If no initial configuration is given, then it constructs an initial placement itself. The initial placements used in this paper are constructed as follows. We select one block each time, by the sequence of the input benchmark files, try two rotations and all the virtual grids, search for the smallest *y*-position in the floor profile as the best position to place the block. Power or ground is treated as a single net. All initial placements, as shown in Table II, are constructed in less than one second of CPU time.

5.2. The Effect of Clustering

The algorithm can employ different cluster sizes. Without clustering, *i.e.*, when the cluster size k is one, the cluster refinement is just the traditional zone refinement. In general cases, the more blocks are selected in the cluster, the more CPU time the algorithm will take. It is also expected that the better results the algorithm will achieve, because the branch-and-bound algorithm will search for larger solution space.

Table III shows the results obtained without cluster overlapping by using different cluster sizes for the biggest MCNC benchmark circuit *ami49*, where $k = l = 4$, $C_1 = 1$, and $C_2 = 0$.

As we can see, when the cluster size k increases from one to five, the chip area decreases. When k is one, *i.e.*, without clustering, the dead space and wire length are 10.29% and 925 respectively. When k is four, the dead space drops to 5.95%,

TABLE II Initial placements constructed

Circuit	Apte	Hp	Xerox	Ami33	Ami49
#blocks	9	11	10	33	49
#nets	97	83	203	123	408
#pins	214	264	696	480	931
#I/Os	73	45	2	42	22
area (mm^2)	84.24	14.54	29.65	1.382	41.00
dead space (%)	44.73	39.25	34.73	16.32	13.57
wire length (mm)	706	266	740	121	1231

TABLE III Placements for *ami49* using different cluster sizes

Cluster size	1	2	3	4	5
area (mm^2)	39.51	38.99	38.38	37.69	37.26
dead space (%)	10.29	9.08	7.64	5.95	4.87
wire length (mm)	925	959	885	764	888
CPU (sec)	5.0	11.3	46.5	1861.7	23202.3

and wire length drops to 764. Though the CPU time also increases with the cluster size, when k is four, the CPU time required is still a very reasonable half an hour.

5.3. The Effect of Cluster Overlapping

Cluster overlapping enables the algorithm to search for larger solution space to find better results. Table IV shows the results for all MCNC benchmark circuits obtained without cluster overlapping when the cluster size is four, where $k = l = 4$, $C_1 = 1$, and $C_2 = 0$.

Table V shows the effect of cluster overlapping on placements for the benchmarks, where $k = 4$, $l = 2$, $C_1 = 1$, and $C_2 = 0$.

Table VI shows the effect of different overlapping sizes $(k - l)$ on placements for the biggest benchmark circuit *ami49*, where $k = 4$, $C_1 = 1$, and $C_2 = 0$.

5.4. The Effect of C's

Parameter C's are the corresponding constant weights of chip size and wire length in the cost function. Apparently, the bigger C_1 is given with respect to C_2, the better the chip size would be achieved, and *vice versa.*

Experimental results for the MCNC benchmarks are shown in Tables VII–XI, where $k = 4$ and $l = 2$.

The corresponding initial placements and final placements are shown in Figure 15.

It is difficult to fairly compare our algorithm to the other approaches, because they include routing space in their placements, which is not necessary any more because of recent technology progress. However, it is estimated that the chip area and wire length would increase around 10% and 5% respectively, if routing space is introduced based on the technology factor T [11]. So our approach still outperforms others a lot.

TABLE IV Placements without cluster overlapping

Circuit	Apte	Hp	Xerox	Ami33	Ami49
area (mm^2)	48.42	9.575	20.30	1.207	37.69
dead space (%)	3.83	7.77	4.69	4.15	5.95
wire length (mm)	321	185	477	64	764
CPU (sec)	23.8	18.0	18.8	603.4	1861.7

TABLE V The effect of cluster overlapping

Circuit	Apte	Hp	Xerox	Ami33	Ami49
area (mm^2)	48.12	9.205	19.80	1.177	36.63
dead space (%)	3.25	4.07	2.25	1.81	3.25
wire length (mm)	558	221	696	82	1330
CPU (sec)	33.0	20.2	38.2	1514.9	3822.8

TABLE VI Placements for *ami49* using different overlapping sizes

Overlapping size	3	2	1
area (mm^2)	36.32	36.63	36.73
dead space (%)	2.42	3.25	3.50
wire length (mm)	1357	1330	1533
CPU (sec)	7566.1	3822.8	1669.9

TABLE VII Experimental results for *apte*, $k = 4$, $l = 2$

	C_1	C_2	C_1	C_2	C_1	C_2	C_1	C_2
	1	0	1	0.001	1	0.01	0	1
area (mm^2)	48.12		48.44		48.44		49.74	
dead space (%)	3.25		3.88		3.88		6.38	
wire length (mm)	558		374		334		330	

TABLE VIII Experimental results for *hp*, $k = 4$, $l = 2$

	C_1	C_2	C_1	C_2	C_1	C_2	C_1	C_2
	1	0	1	0.001	1	0.01	0	1
area (mm^2)	9.205		9.436		9.575		9.899	
dead space (%)	4.07		6.42		7.77		10.79	
wire length (mm)	221		215		192		185	

TABLE IX Experimental results for *xerox*, $k = 4$, $l = 2$

	C_1	C_2	C_1	C_2	C_1	C_2	C_1	C_2
	1	0	1	0.001	1	0.01	0	1
area (mm^2)	19.80		20.01		20.54		20.70	
dead space (%)	2.25		3.32		5.79		6.53	
wire length (mm)	696		627		526		482	

TABLE X Experimental results for *ami33*, $k = 4$, $l = 2$

	C_1	C_2	C_1	C_2	C_1	C_2	C_1	C_2
	1	0	1	0.001	1	0.01	0	1
area (mm^2)	1.177		1.195		1.203		1.246	
dead space (%)	1.81		3.26		3.84		7.18	
wire length (mm)	82		74		67		54	

TABLE XI Experimental results for *ami49*, $k = 4$, $l = 2$

	C_1	C_2	C_1	C_2	C_1	C_2	C_1	C_2
	1	0	1	0.001	1	0.01	0	1
area (mm^2)	36.63		36.65		37.34		38.38	
dead space (%)	3.25		3.30		5.08		7.64	
wire length (mm)	1330		879		791		727	

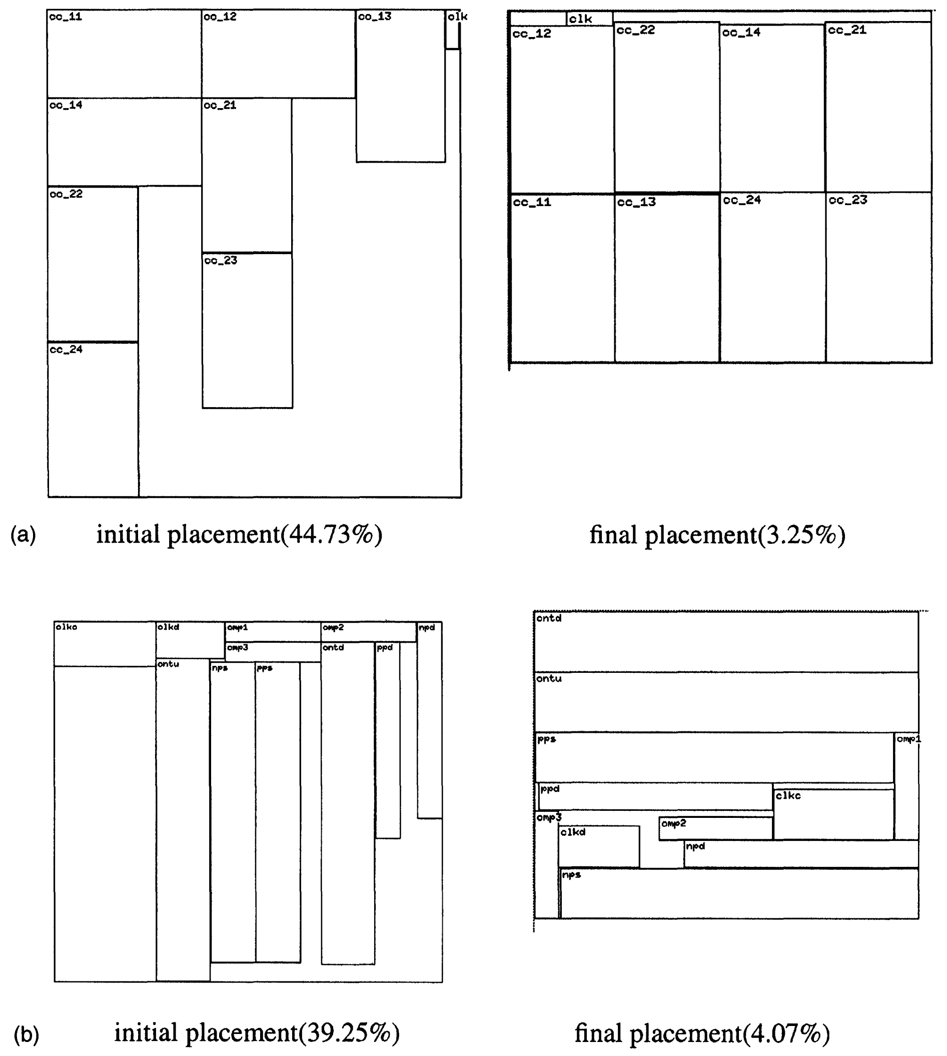
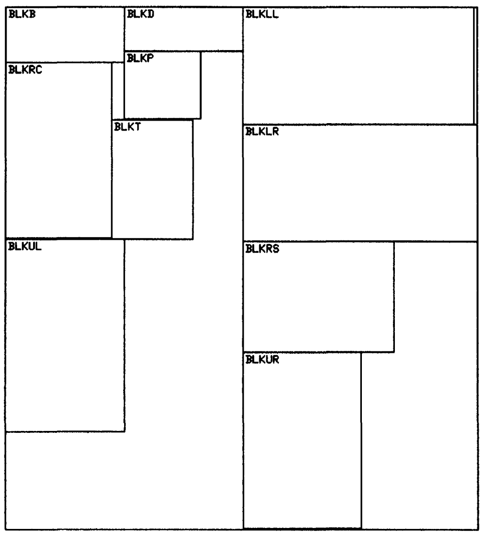
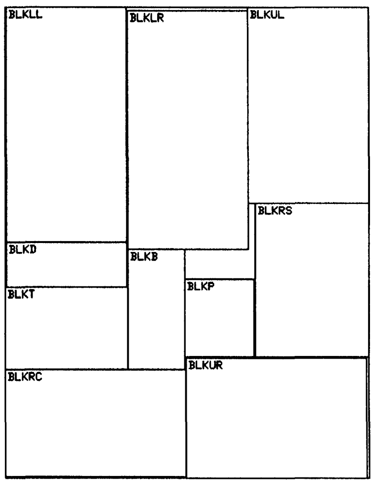
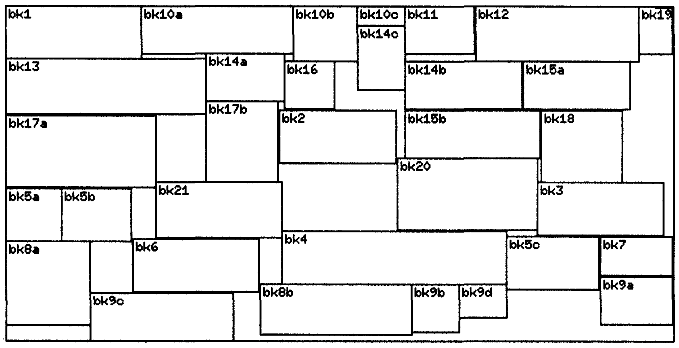


FIGURE 15 Final placements for the MCNC benchmark circuits. (a) apte (b) hp (c) xerox (d) ami33 (e) ami49.

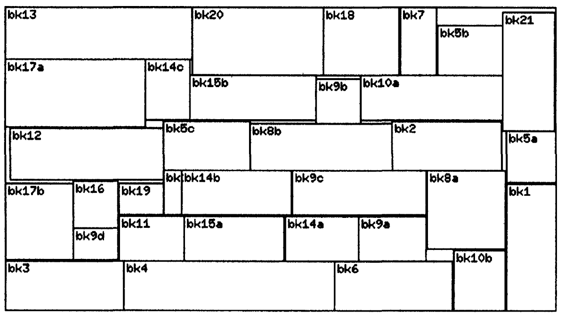




(c) initial placement(34.73) final placement(2.25)

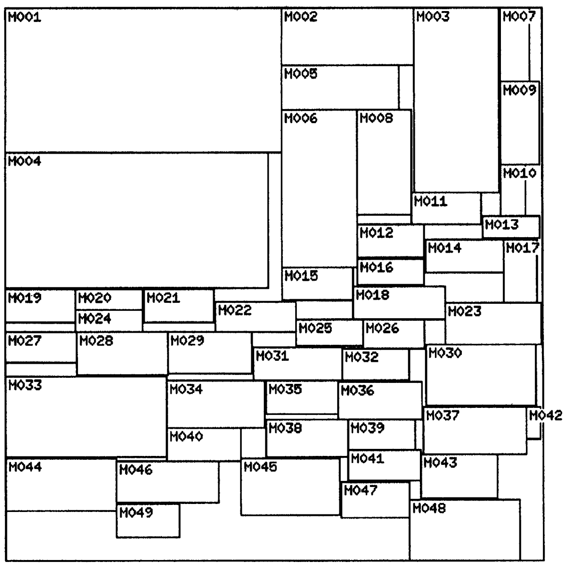


initial placement(16.32%)

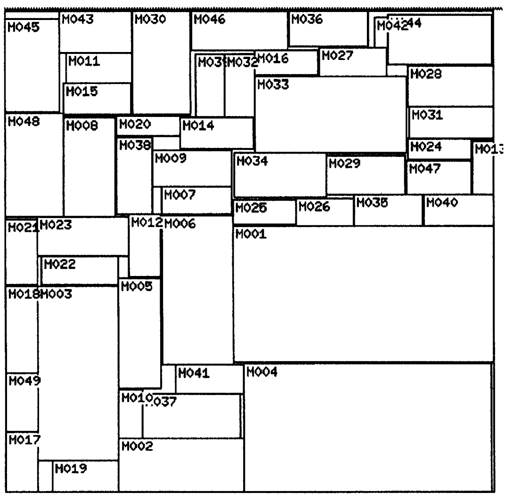


(d) final placement(1.81%)

FIGURE 15 (Continued).



initial placement(13.57%)



(e) final placement(2.42)

FIGURE 15 (Continued).

6. CONCLUSIONS

We present an efficient cluster refinement algorithm which minimizes the chip area and wire length at the same time. We try all possible configurations on the selected cluster to minimize the gap distance between the ceiling and the floor. A virtual grid and permutation order are generated dynamically to eliminate redundant branches, which was the cause of much higher complexity in other approaches. The experimental results demonstrate clearly that the bounding operations proposed are very effective, and result in a very efficient search within the large solution space.

Experimental results on MCNC benchmark circuits show excellent potential for the algorithm. The algorithm reaches good solutions in a practical amount of CPU time on the benchmark circuits.

Acknowledgment

The authors would like to thank the reviewers for their comments and the support by NSF grant MIP-9529077 and by California MICRO program.

References

[1] Banerjee, P. (1994). "Parallel Algorithms for VLSI Computer-Aided Design", PTR Prentice Hall.

[2] Burkard, R. E. and Bonniger, T. (1983). "A heuristic for Quadratic Boolean Programs with Applications to Quadratic Assignment Problems", *European Journal Operational Research*, **13**, 374–386.

[3] Cheng, C. K. and Kuh, E. S., "Module Placement based on Resistive Network Optimization", *IEEE Trans. Computer-Aided Design*, **CAD-3**, 218–225, July 1984.

[4] Gao, T., Caidya, P. M. and Liu, C. L. (1992). "A Performance Driven Macro-Cell Placement Algorithm", *Proc. 29th Design Automation Conf.*, pp. 147–152.

[5] Hamada, T., Cheng, C. K. and Chau, P., "An Efficient Multi-level Placement Technique Using Hierarchical Partitioning", *IEEE Trans. Circuits and Systems*, **39**, 432–439, June 1992.

[6] Hu, T. C. (1982). "Combinatorial Algorithms", Addison Wesley.

[7] Dai, W. M. and Kuh, E. S., "Simultaneous Floor Planning and Global Routing for Hierarchical Building-Block Layout", *IEEE Trans. Computer-Aided Design*, **CAD-6**, 828–837, Sept. 1987.

[8] Esbensen, H. and Kuh, E. S., "EXPLORER: An Interactive Floorplanner for Design Space Exploration", *Proc. EuroDAC'96*, pp. 356–361, Sept. 1996.

[9] Koakutsu, S., Kang, M. and Dai, W. W.-M. (1996). "Genetic Simulated Annealing and Application to Non-slicing Floorplan Design", *Proc. 1996 Physical Design Workshop*, pp. 134–141.

[10] Lengauer, T. (1990). "Combinatorial Algorithms for Integrated Circuit Layout", John Wiley & Sons Ltd.

[11] Murata, H., Fujiyoshi, K., Nakatake, S. and Kajitani, Y., "Rectangular-Packing-Based Module Placement", *Proc.*

IEEE International Conf. on Computer-Aided Design'95, pp. 472–479, Nov. 1995.

[12] Neapolitan, R. and Naimipour, K. (1996). "Foundations of Algorithms", D. C. Heath and Company.

[13] Onodera, H., Taniguchi, Y. and Tamaru, K. (1991). "Branch-and-Bound Placement for Building Block Layout", *Proc. 28th Design Automation Conf.*, pp. 433–439.

[14] Otten, R. H. J. M. (1982). "Automatic Floorplan Design", *Proc. 19th Design Automation Conf.*, pp. 261–267.

[15] Shin, H., Sangiovanni-Vincentelli, A. L. and Sequin, C. H., "'Zone-Refining' Techniques for IC Layout Compaction", *IEEE Trans. Computer-Aided Design*, **9**, 167–178, Feb. 1990.

[16] Wong, D. F. and Liu, C. L. (1986). "A New Algorithm for Floorplan Design", *Proc. 23rd Design Automation Conf.*, pp. 101–107.

[17] Xu, J., Guo, P. N. and Cheng, C. K. (1997). "Cluster Refinement for Block Placement", *Proc. 34th Design Automation Conf.*, pp. 762–765.

[18] Yamanouchi, T., Tamakashi, K. and Kambe, T. (1996). "Hybrid Floorplanning Based on Partial Clustering and Module Restructuring", *Proc. IEEE International Conf. on Computer-Aided Design*, pp. 478–483.

Authors' Biographies

Jin Xu received the B.S. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 1985, and the M.S. degree from Tsinghua University, Beijing, China, in 1988, both in electrical engineering. She is a Senior Member of Technical Staff ar Cadence Design Systems, Inc., San Jose, CA. Her research interest are in VLSI physical design algorithms, especially in floorplanning and placement.

Pei-Ning Guo received the B.S. degree in computer science from the National Taiwan University, Taiwan, the M.S. degree in computer science from New York University, and the Ph.D. degree in computer science and engineering from the University of California, San Diego in 1998. He joined Mentor Graphics Corp., San Jose, as a senior engineer in 1999. His research interests include floorplan and placement approaches for VLSI physical design.

Chung-Kuan Cheng received the B.S. and M.S. degrees in electrical engineering from the National Taiwan University, Taiwan, and the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, in 1984. From 1984–1986 he was a Senior CAD Engineer at Advanced Micro Device Inc. In 1986, he joined the University of California, San Diego (UCSD), where he was a Professor in the Computer Science and Engineering Department. Currently, he serves as a Chief Scientist at Mentor Graphics while taking leave from the university. His research interests include network optimization and design automation on microelectronic circuits. Dr. Cheng has been an Associate Editor of IEEE Transaction on Computer-Aided-Design of Integrated Circuits and Systems since 1994. He is a recipient of the best paper award IEEE Transaction on Computer-Aided-Design of Integrated Circuits and Systems, 1997, the NCR Excellence in Teaching award from the School of Engineering, UCSD in 1991.