A New and Efficient Congestion Evaluation Model in Floorplanning: Wire Density Control with Twin Binary Trees

Steve T. W. Lai
Department of CSE
The Chinese Univ. of H.K.
Shatin, N.T., Hong Kong
twlai@cse.cuhk.edu.hk

Evangeline F. Y. Young
Department of CSE
The Chinese Univ. of H.K.
Shatin, N.T., Hong Kong
fyyoung@cse.cuhk.edu.hk

Chris C. N. Chu
Department of ECPE
Iowa State University
Ames, IA 20011-3060
cnchu@iastate.edu

Abstract

As technology moves into the deep-submicron era, the complexity of VLSI circuit design grows rapidly, especially in the interconnections between modules. Therefore, interconnect optimization has become an important concern in floorplanning today. Most routability-driven floorplanners [2][6][8] use grid-based approach that divides a floorplan into grids as in global routing. Congestion is estimated as the expected number of nets passing through each grid. Although this approach is direct and accurate, it is not efficient when dealing with complex circuit containing thousands of nets. In this paper, an efficient and innovative routability-driven floorplanner using twin binary trees (TBT)[9][10] representation is proposed. The congestion model we used is the wire density on the half-perimeter boundary of different regions in a floorplan. These regions are defined naturally by the TBT representation. In order to increase the efficiency of our floorplanner, a fast algorithm for the least common ancestor (LCA) problem in [1] is used to compute the wire density. From the experimental results, the number of unroutable wires can be reduced in a short time.

1. Introduction

1.1. Motivation

In the deep-submicron era, the complexities of VLSI circuits are growing rapidly. The interconnections between modules will become longer and denser in the future. Therefore, interconnect optimization in floorplan design has become ever more important than before. As floorplanning is at the beginning phase of the VLSI design cycle, an interconnect-optimized floorplan will favor the applicability and performance of the later designing stages like placement, global routing, detailed routing, etc, and, most importantly, allow timing closure to be achieved earlier.

Recently, some routability-driven floorplanners [2][6][8] are proposed. Most of them use the grid-based approach to measure the congestion of a floorplan. In this approach, a floorplan is divided into grids as in global routing. At each grid, the expected number of nets passing through is recorded as a weight to measure congestion. Although this approach is direct and simple, such kind of routing-oriented estimation is time consuming if it is performed in each iteration of the simulated annealing process in a floorplanner. It is impractical for complex circuit designs. Therefore, a new and fast congestion evaluation model using a suitable floorplan representation will be very useful.

1.2. Previous work

Recently, several floorplanners are proposed to consider routability in the floorplanning phase. In paper [2], a floorplan is divided into grids and congestion is estimated at each grid by assuming that each wire is routed in either L-shape or Z-shape. They use simple-geometry routing to plan the wires in reasonable time. In paper [8], a realistic global router is used to evaluate the congestion of each placement solution. In paper [6], a probabilistic method is proposed to estimate congestion and routability. A floorplan is divided into a 2-dimensional grid structure and congestion is estimated at each grid. Similar approaches are also proposed in [4] and [5]. Although the above congestion evaluation models have been shown to be effective in reducing interconnect cost, their computational costs are very high.

1.3. Our contribution

In order to provide a simple and efficient congestion evaluation method other than the complicated grid-based approach, an indirect congestion evaluation model, *wire density*, is proposed. Instead of estimating the congestion at each grid using global routing, we measure congestion as the wire density passing through the boundary of different regions in a floorplan. It is because a floorplan, that has a high wire density on average, has a greater chance of having congestion problem. An example is shown in figure 1. We use *twin binary trees (TBT)* as the floorplan representation because the regions to be evaluated can be naturally defined by the TBT representation. For a floorplan with n modules, $n-1$ regions

1530-1591/03 $17.00

are defined by each tree. In order to provide more regions for evaluation, we have constructed an additional pair of trees, which is the mirror of the original pair of trees. In order to increase the efficiency of our floorplanner, we have made use of a fast algorithm [1] for the *least common ancestor (LCA)* problem to compute the wire density. Experimental results have shown that an interconnect-optimized floorplan of a complex circuit can be obtained in less than five minutes.

This paper is divided into seven sections. In section 2, a brief review of the TBT floorplan representation will be given. Section 3 will give an overview of our floorplanner. In section 4 and 5, the ideas and implementation details of the wire density congestion evaluation model will be described and explained. Finally, experimental results will be shown in section 6.

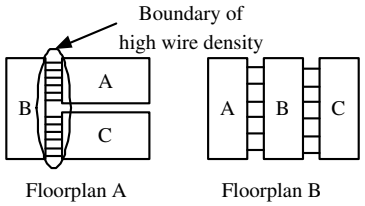

Figure 1. High wire density in floorplan A.

2. Twin binary trees

In our floorplanner, we use twin binary trees (TBT) as our floorplan representation. The TBT floorplan representation is first proposed in paper [9]. It shows an one-to-one mapping between TBT and mosaic floorplan. Recall that the definition of twin binary trees is as follows:

Definition 1 *The set of twin binary trees* $TBT_n \subseteq Tree_n \times Tree_n$ *is the set:*

$$TBT_n = \{(t_1,t_2) \mid t_1,t_2 \in Tree_n \text{ and } \theta(t_1) = \theta^c(t_2)\}$$

where $Tree_n$ *is the set of binary trees with n nodes, and* $\theta(t)$ *is the labelling of the binary tree t.*

The labelling of a binary tree t can be obtained by performing an in-order traversal on t. When the traversed node has no left child, a bit 0 is added to the sequence. Similarly, if it has no right child, a bit 1 is added to the sequence. The first 0 and the last 1 in the labelling are omitted. If a pair of trees (t_1,t_2) are twin binary to each other, their labellings will be the complement of each other, i.e., $\theta(t_1) = \theta^c(t_2)$.

Given a mosaic floorplan F, we can construct a pair of trees (t_1, t_2) by travelling along the slicelines of F. The root of t_1 is the upper right corner of the packing. By connecting the upper right corners of all the modules, the horizontal slicelines represent the tree edges connecting from a parent to its left child, while the vertical slicelines represent the tree edges connecting from a parent to its right child. The construction of t_2 can be done similarly by connecting the lower left corners of all the modules. It has been shown that the pair of trees constructed in this way must be twin binary to each other. Also, it is observed that the in-order traversal of the pair of trees are the same[10]. An example is shown in figure 2, $\theta(t_1) = 10010$ and $\theta(t_2) = 01101$, so $\theta(t_1) = \theta^c(t_2)$. Also, their in-order traversals are both $ABCFDE$.

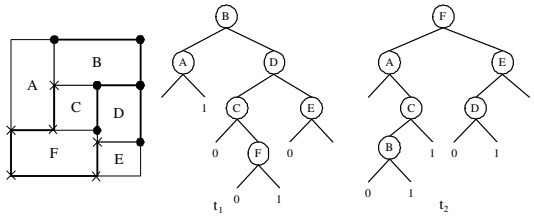

Figure 2. Construction of TBT.

3. Overview of our floorplanner

In this section, we will give a brief introduction to our routability-driven floorplanner with the new wire density congestion evaluation model. Our floorplanner is based on the TBT floorplan representation and simulated annealing is used. Given a candidate floorplan solution, the total wire length of the nets is estimated by the half-perimeter bounding box approach. The congestion cost is estimated by the wire density which is computed as the number of nets passing per unit length of the boundary of different regions. These regions are defined by the TBT representation naturally and hierarchically. The estimation of wire density will start from the leaf nodes and follow the post-order traversal of the tree. Each tree can provide $n-1$ samples, i.e., $n-1$ regions, for wire density estimation. In order to obtain more samples, two additional trees are constructed from the original pair of TBT to provide a total of $4(n-1)$ wire density values.

4. Wire density model

In order to improve the routability of a floorplan solution in an efficient way, an indirect but effective congestion evaluation model is used. This model aims at measuring congestion as the wire density (number of nets per unit length) on the boundary of different regions in a floorplan.

Definition 2 *Given a TBT (*t_1*,* t_2*), the region* $R(i)$ *covered by module i in* $t \in \{t_1,t_2\}$ *is the rooms occupied by module i and the modules in the subtree rooted at i in t.*

As shown in figure 3, the region $R(D)$ covered by module D in t_1 includes all the rooms occupied by module D, C, F and E. We can obtain $n-1$ wire density values for a tree with n nodes. It is because $R(root)$ is the whole packing and there will be no nets passing through the boundary of the packing.

The following gives the equation to calculate the *wire density* of $R(i)$:

$$C_i = \frac{N_i}{P_i} \tag{1}$$

where C_i is the wire density of $R(i)$, N_i is the total number of nets passing through the boundary of $R(i)$ and P_i is the normalized half-perimeter of $R(i)$. The details of the computation of N_i and P_i will be given in the coming sections.

We choose TBT as the floorplan representation in our floorplanner because it can define the regions for evaluation naturally. Also, a lot of fast and simple tree algorithms can be used in our congestion evaluation. We start the estimation of wire density from the leaf nodes and follow the post-order traversal of the tree to compute the terms N_i and P_i at each node i. By dynamic programming, the information computed at the children can be used to compute the wire density at the parent.

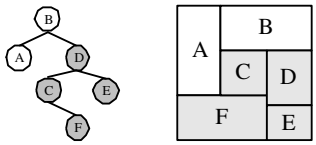

Figure 3. Formation of $R(D)$.

4.1. Computation of N_i

The term N_i, which is the total number of nets passing through the boundary of $R(i)$, can be computed as follows:

$$N_i = N'_i + N_{l(i)} + N_{r(i)} - M'_i \tag{2}$$

where $l(i)$ is the left child of i, $r(i)$ is the right child of i, N'_i is the number of nets connected to module i, M'_i is an offset for adjustments due to net merging and net completion, and it is computed as follows:

$$M'_i = \sum_{j=2}^{3} (j-1) m^i_j + \sum_{j=2}^{3} j \cdot c^i_j \tag{3}$$

where m^i_j and c^i_j are the number of nets merged and completed at i. The value j is the number of subnets of a single net that meet at i. It can be either two or three.

The adjustment for net merging m^i_j is needed because the repeated counting of an identical net in N'_i, $N_{l(i)}$ and $N_{r(i)}$ will over-estimate the term N_i. For $j = 2$, two subnets coming from $R(l(i))$, $R(r(i))$ or module i of a single net are merged. For $j = 3$, three subnets coming from $R(l(i))$, $R(r(i))$ and module i of a single net are merged. The term m^i_j is multiplied by $j-1$ because we need to keep one counting in N_i rather than j counting. An example is shown in figure 4. In figure 4, we consider the situation when we reach module D during the post-order traversal. We use thick solid lines to represent merged nets. There is one net merged between module D and $R(C)$, one between module D and $R(E)$, and one between $R(C)$ and $R(E)$, so $m^D_2 = 3$. There is also one net merged between module D, $R(C)$ and $R(E)$, so $m^D_3 = 1$.

Similarly, the adjustment for net completion c^i_j is needed because the repeated counting of an identical net in N'_i, $N_{l(i)}$ and $N_{r(i)}$ will over-estimate the term N_i. The value j in c^i_j has the same meaning as that in m^i_j. The term c^i_j is multiplied by j because the net has completed and all the counting should be eliminated. In figure 4, we use thick dotted lines to represent completed nets. There is one net completed between module D and $R(C)$, two nets completed between module D and $R(E)$, three nets completed between $R(C)$ and $R(E)$, so $c^D_2 = 1+2+3 = 6$. There is also one net completed between module D, $R(C)$ and $R(E)$, so $c^D_3 = 1$. Finally, $M'_D = m^D_2 + 2m^D_3 + 2c^D_2 + 3c^D_3 = 3 + 2(1) + 2(6) + 3(1) = 20$

In figure 4, N_D is computed as $N'_D + N_C + N_E - M'_D$ where $N'_D = 10$, $N_C = 13$, $N_E = 11$ and $M'_D = 20$. As a result, $N_D = 10 + 13 + 11 - 20 = 14$. There are 14 nets passing through the boundary of $R(D)$. The value of N'_i can be obtained easily as the net specification is given in the floorplanning phase. However, the term M'_i will vary for different packings, a naive method to compute M'_i will impose an $O(mn)$ time complexity where n is the total number of nets and m is the total number of modules. It is impractical for complex circuits. Therefore, we have made use of an efficient algorithm for the least common ancestor (LCA) problem to compute M'_i. Details of the implementation will be given in section 5.

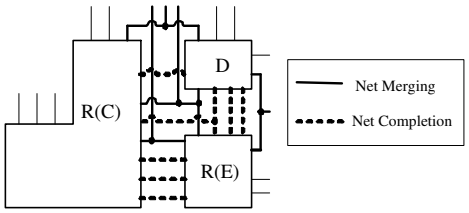

Figure 4. An example of computing N_D.

4.2. Computation of P_i

The term P_i, which is the normalized half-perimeter of $R(i)$, can also be computed easily by following the post-order traversal of the tree. As the tree edges of a TBT represent the width and height of the rooms occupied by the modules, we will separate the half-perimeter P_i of region $R(i)$ into the horizontal (P^h_i) and vertical (P^v_i) portions to make the operation simple. The pseudo-code is given as follows:

HalfPerimeter(tree t)
1. *For* $j = 1$ *to n where* $(\pi(1), \pi(2), \ldots, \pi(n))$ *is the post-order traversal of t*
2. $\quad i = \pi(j)$
3. $\quad$ *If i is a leaf node*
4. $\quad\quad P^h_i = w_i$
5. $\quad\quad P^v_i = h_i$

6. *If i has left child $l(i)$ only*
7. $P_i^h = w_i + P_{l(i)}^h$
8. $P_i^v = max(h_i, P_{l(i)}^v)$
9. *If i has right child $r(i)$ only*
10. $P_i^h = max(w_i, P_{r(i)}^h)$
11. $P_i^v = h_i + P_{r(i)}^v$
12. *If i has both left and right child, $l(i)$ and $r(i)$*
13. $P_i^h = max((w_i + P_{l(i)}^h), P_{r(i)}^h)$
14. $P_i^v = max((h_i + P_{r(i)}^v), P_{l(i)}^v)$
15. $P_i = \frac{P_i^h}{chip\_width} + \frac{P_i^v}{chip\_height}$

In the pseudo-code, w_i and h_i are the width and height of the room occupied by module i. The computation of $P(i)$ is divided into four cases. Line 3-5 is the case where module i is a leaf node as in figure 5(a). Figure 5(b) shows the case of line 6-8 where module i has a left child $l(i)$ only. Figure 5(c) shows the case of line 9-11 where module i has a right child $r(i)$ only. Line 12-14 is the last case where module i has both left child $l(i)$ and right child $r(i)$ as in figure 5(d). Finally, on line 15, P_i^h and P_i^v are normalized by the chip width and chip height respectively to maintain a uniform order of magnitude. As dynamic programming is applied in the computation, the time complexity of *HalfPerimeter(t)* to compute the normalized half-perimeters of all the $(n-1)$ regions is only $O(n)$.

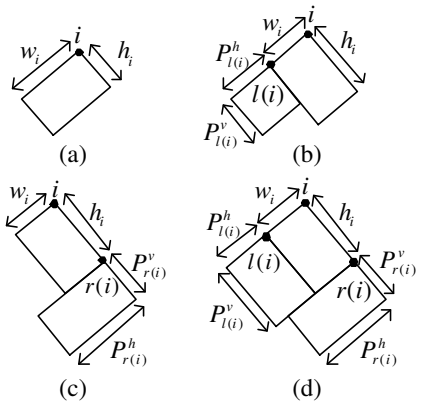

Figure 5. Cases in P_i computation.

4.3. Usage of mirror TBT

After discussing the computation of N_i and P_i, we can evaluate the wire density for t_1 and t_2. By the characteristic of the TBT representation, the C_i computed from t_1 represent the wire densities of the boundaries facing the upper right direction, while those computed from t_2 represent the wire densities of the boundaries facing the lower left direction. Each tree can give $n-1$ statistical samples for the wire density evaluation where n is the number of modules. In order to increase the effectiveness of our congestion model, a pair of *mirror TBT*, which are based on the original pair of TBT, are constructed. The mirror TBT can be imagined as the TBT constructed from a packing which is rotated 90° counterclockwise. Together with the mirror TBT, our congestion model can give $4(n-1)$ wire density values which consider in four routing directions (upper right for t_1, lower left for t_2, upper left for t_3 and lower right for t_4). As sufficient statistical samples are considered, the routability of a packing can be estimated correctly.

5. Implementation

5.1. Efficient calculation of N_i

In this section, a detailed explanation of using the LCA algorithm to compute N_i will be given. Recall from section 4.1 that the major difficulty of computing N_i is the high computational cost of computing the term M_i'. Instead of computing M_i' for each module i one by one, we are going to compute all M_i' incrementally by visiting each net one by one. Let's look at the example in figure 6. In this example, we need to find the nodes B, C and D where adjustments are needed due to net merging and completion of net p. Net p will merge at node B, C and D, and finally complete at B. The nodes where adjustments are needed are $LCA(u,v)$, where (u,v) are some module pairs in a net. For a net with k modules, $k-1$ LCAs should be found for adjustments. It is observed that we cannot get the correct LCAs where adjustments are needed by just picking the module pairs arbitrarily. For example, the LCAs obtained by simply selecting the three adjacent module pairs from the original net specification of p in figure 6 are $LCA(A,C) = B$, $LCA(C,E) = D$ and $LCA(E,F) = D$ which are not the correct set of LCAs $\{B,C,D\}$ where adjustments are needed. Therefore, the following lemma is used to find the correct set of LCAs where adjustments are needed for a net p.

Lemma 1 *Given a tree t with n nodes (representing n modules) and a net p connecting k modules $(m_1, m_2, \ldots, m_k)$. The set of nodes L_p in t where two or more subnets of p meet (adjustment is needed) is*

$$L_p = \bigcup_{i=1}^{k-1} \{LCA(m_{\pi(i)}, m_{\pi(i+1)})\}$$

where $(m_{\pi(1)}, m_{\pi(2)}, \ldots, m_{\pi(k)})$ is a permutation of the k modules obtained by following the pre-order traversal of t. (In figure 6, the permutation of the modules connected by p following the pre-order traversal is $(ACFE)$ and $L_p = \{B,C,D\}$.)

Proof: The proof is done by induction on the depth of the tree t. The pre-order traversal of t of depth $n+1$ can be expressed as AB_nC_n where A is the root, B_n and C_n represent the pre-order traversal of the left subtree of A rooted at B and the right subtree of A rooted at C with depth smaller than or equal to n respectively, and n is the

larger value of the depths of the left and right subtree of A. Because of the lacking of space, we will show the proof for the case where A have both left and right subtrees only. The cases where A has left subtree or right subtree only can be proved similarly.

When $n = 1$, the pre-order traversal of t is (AB_1C_1) as shown in figure 7(a). For the tree t where A have both left and right subtrees, B_1 and C_1 represent B and C respectively and the pre-order traversal is ABC. Consider the case for a net $p = \{C,B,A\}$, the subnets of p will meet (twice) at node A. The permutated p is $\{A,B,C\}$, and the LCAs found according to the lemma are correct since $LCA(A,B) = A$ and $LCA(B,C) = A$. The cases where $p = \{B,A\}$, $p = \{C,A\}$ and $p = \{C,B\}$ can be proved similarly. Hence, the proposition is true when $n = 1$.

Assume that the proposition is true when $n = k$, and the pre-order traversal of t is AB_kC_k as shown in figure 7(b). When $n = k+1$, the pre-order traversal of t will be $AB_{k+1}C_{k+1}$. We can re-write it as $A(BD_kE_k)(CF_kG_k)$ as in figure 7(c). Let B^f and B^l be the first and last node of the permutated subnet of p in BD_kE_k respectively, and C^f be the first node of the permutated subnet of p in CF_kG_k. For the case where net p resides in the left and right subtrees of A and node A, the LCAs found from the left and right subtrees of A are correct according to the inductive hypothesis. There is one more node that the subnets of p will meet (twice), which is A, and it will be found correctly according to the lemma since $LCA(A,B^f) = A$ and $LCA(B^l,C^f) = A$. The cases where net p resides in the left or right subtree of A completely, p resides in the left subtree of A and node A, p resides in the right subtree of A and node A, and p resides in the left and right subtrees of A but not node A can be argued similarly. Hence, the proposition is true for $n = k+1$. *Q.E.D.*

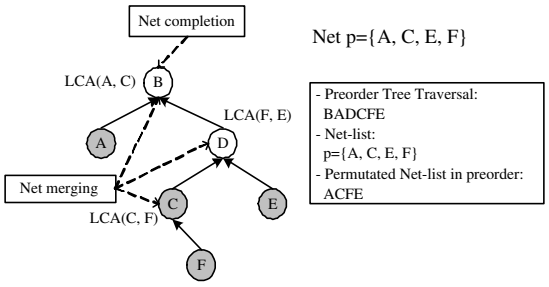

Figure 6. Using LCA to compute N_i.

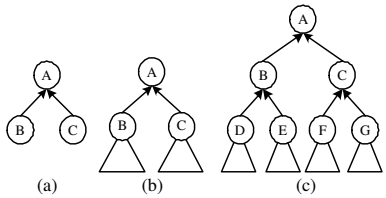

Figure 7. Proof of our M'_i computation.

After obtaining the set L_p of a net p, we can update the value of the corresponding M'_i. As shown in figure 6, M'_B, M'_C and M'_D will be incremented by 1 because net p will be merged when they are visited. Finally, M'_i of the shallowest module i in the set L_p will be further incremented by 1 because the net is going to be completed there. In figure 6, this shallowest module is B. The same operation will be performed for each net to compute all M'_i. Finally, we can apply equation (2) to compute all N_i for wire density computation.

5.2. Solving the LCA problem efficiently

In paper [1], an efficient and simple LCA algorithm is proposed. It reduces the LCA problem to the *Range Minimum Query (RMQ)* problem. By applying the *Sparse Table (ST)* algorithm for the RMQ problem, the LCA problem can be solved in constant time with a preprocessing time of $O(nlogn)$ using dynamic programming.

5.3. Cost Function

The cost function for the simulated annealing process of our floorplanner is shown as follows:

$$cost = A + \alpha W + \beta C \tag{4}$$

where A is the chip area of the floorplan, W is the total wire length estimated, C is the summation of all the wire density values of the floorplan, and α and β are the weights to describe the importance of these three terms. In our floorplanner, α and β are set such that the ratio of the importance of these three terms are $A : W : C = 2 : 2 : 1$.

5.4. Complexity

Efficiency is one of the major advantages of our wire density congestion model. Recall from equation (1) that the computation of the wire density W_i is divided into two parts, N_i and P_i. The operations needed to compute N_i for all i are the construction of the LCA sparse table ($O(nlogn)$), the computation of M'_i for all i ($O(k)$) where k is the total number of pins in all nets, and the computation of equation (2) ($O(n)$), so the time complexity of computing N_i will be $O(nlogn) + O(k) + O(n) = O(nlogn) + O(k)$. Usually, the magnitude of k is much greater than that of n, so we treat the time complexity of computing N_i as $O(k)$. Secondly, the time complexity of computing P_i for all i is $O(n)$. As a result, the time complexity of our congestion estimation method is $O(k)$ only.

6. Experimental results

We have implemented two floorplanners for testing. One is a traditional floorplanner without considering congestion, the other one is a routability-driven floorplanner using our wire density model. Both floorplanners are based on the TBT floorplan representation and the simulated annealing approach. Three MCNC benchmarks $\{ami33, ami49, playout\}$

are used. In addition, three data sets $\{n2000, n2500, n3000\}$ are created to demonstrate the performance of our floorplanner for complex circuits. The detailed specifications of the data sets are shown in table 1. The experiments are performed using a PC with a Pentium III 1GHz processor and 1GB memory. We use a simple global router to evaluate the performance of the floorplanners.

Experimental results are shown in table 2. The term *unroutable wire* is the wire that cannot be routed in the shortest Manhattan distance due to the limitation of the wire capacity at each grid in the global router. The term *congestion* is the average number of nets per μm^2 of the top 10% most congested grids reported by the router. We use the data in paper [7] to compute the parameters in the router. We use the feature values of the $0.18\mu m$ technology for *ami*33 and *ami*49. For the other data sets, we use the feature values of the $0.13\mu m$ technology. The results show a significant reduction in the number of unroutable wires. The results in congestion are similar for the two floorplanners. It shows that our floorplanner is actually more efficient in distributing the wires uniformly since more wires are being routed (the number of unroutable wires has decreased) without increasing the congestion measures. Besides, the wire capacity of each grid is actually bounded to avoid over-congestion in the router and the results show that the wiring capacity of each grid is more fully utilized by our floorplanner. Another important result from the experiment is that the runtime of our floorplanner for a complex circuit with three thousand nets ($n3000$) is less than five minutes. It demonstrates both the effectiveness and efficiency of our wire density model in reducing the interconnect cost.

Data	Number of Modules	Number of IO Pins	Number of Nets
ami33	33	42	123
ami49	49	22	408
playout	62	192	1611
n2000	60	200	2000
n2500	75	200	2500
n3000	90	200	3000

Table 1. Specifications of the data sets.

7. Conclusion

In this paper, we present a new congestion model using wire density as a measurement. We use TBT as the floorplan representation because the regions for evaluation can be defined by the TBT representation naturally, the fast and simple tree algorithms, for example, the LCA algorithm, can facilitate the efficiency of our congestion model. By using the regions defined by the TBT and the mirror TBT, sufficient samples can be taken for congestion evaluation. The time complexity of the whole congestion estimation method is linear with respect to the total number of pins in all nets. Experiments have shown that this congestion evaluation model is efficient and effective when dealing with complex circuits. The number of unroutable wires can be greatly reduced in a short time.

Data	Dead Space (%)	Wire Length ($10^3\mu m$)	Number of Unroutable Wires	Congestion (# of nets per μm^2)	Runtime (s)
Floorplanner with wire density control					
ami33	10.10	6.34	0.00	0.83	70.60
ami49	13.76	30.16	0.20	1.00	117.46
playout	12.25	63.64	7.48	8.31	146.94
n2000	13.47	104.35	99.82	12.71	176.20
n2500	16.10	147.69	242.67	13.67	235.76
n3000	16.76	184.76	386.37	14.09	288.34
Floorplanner without considering congestion					
ami33	8.50	6.13	0.03	0.86	25.14
ami49	12.46	30.26	0.60	1.07	42.20
playout	10.84	62.97	9.20	8.66	57.82
n2000	11.62	99.09	142.20	12.79	62.36
n2500	13.93	141.18	314.34	13.46	82.63
n3000	14.54	177.77	476.24	13.82	105.35

Table 2. Experimental results of our floorplanner.

References

[1] M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Latin American Theoretical INformatics*, pages 88–94, 2000.

[2] H. M. Chen, H. Zhou, F. Y. Young, D. F. Wong, H. H. Yang, and N. A. Sherwani. Integrated floorplanning and interconnect planning. In *Proc. Int. Conf. On CAD*, pages 354–357, 1999.

[3] 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. Int. Conf. On CAD*, pages 8–12, 2000.

[4] P. Hung and M. J. Flynn. Stochastic congestion model for VLSI systems. Technical Report CSL-TR-97-737, Stanford University, 1997.

[5] J. Lou, S. Krishnamoorthy, and H. S. Sheng. Estimating routing congestion using probabilistic analysis. In *Int. Symp. Physical Design*, pages 112–117, 2001.

[6] C. W. Sham and E. F. Y. Young. Routability driven floorplanner with buffer block planning. In *Int. Symp. Physical Design*, pages 50–55, 2002.

[7] D. Sylvester and K. Keutzer. Getting to the bottom of deep submicron. In *Proc. Int. Conf. On CAD*, pages 203–211, 1998.

[8] M. Wang and M. Sarrafzadeh. Modeling and minimization of routing congestion. In *Proc. of Asian-Pacific DAC*, pages 185–190, 2000.

[9] B. Yao, H. Chen, C. K. Cheng, and R. Graham. Revisting floorplan representations. In *Int. Symp. Physical Design*, pages 138–143, 2001.

[10] E. F. Y. Young, C. C. N. Chu, and Z. C. Shen. Twin binary sequences: A non-redundant representation for general non-slicing floorplan. In *Int. Symp. Physical Design*, pages 196–201, 2002.