INTEGRATION, the VLSI journal 42 (2009) 449–456

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

Handling routability in floorplan design with twin binary trees ☆

Steve T.W. Lai [a], Evangeline F.Y. Young [a,*], Chris C.N. Chu [b]

[a] Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
[b] Department of Electrical and Computer Engineering, Iowa State University, IA 50011-3060, USA

ARTICLE INFO

Article history:
Received 20 June 2008
Received in revised form
27 February 2009
Accepted 4 March 2009

Keywords:
Floorplanning
Routability
Buffer planning
Simulated annealing

ABSTRACT

As technology moves into the deep-submicron era, the complexities of VLSI circuits grow rapidly. Interconnect optimization has become an important concern. Most routability-driven floorplanners [H.M. Chen, H. Zhou, F.Y. Young, D.F. Wong, H.H. Yang, N. Sherwani, Integrated floorplanning and interconnect planning, in: Proceedings of IEEE International Conference on Computer-Aided Design, 1999, pp. 354–357; S. Krishnamoorthy, J. Lou, H.S. Sheng, Estimating routing congestion using probabilistic analysis, in: Proceedings of International Symposium on Physical Design, 2001, pp. 112–117; M. Wang, M. Sarrafzadeh, Modeling and minimization of routing congestion, in: IEEE Asia and South Pacific Design Automation Conference, 2000, pp. 185–190] use grid-based approach that divides a floorplan into grids as in global routing to estimate congestion by the expected number of nets passing through each grid. This approach is direct and accurate, but not efficient enough when dealing with complex circuits containing many nets. In this paper, an efficient and innovative interconnect-driven floorplanner using twin binary trees (TBT) representation [B. Yao, H. Chen, C.K. Cheng, R. Graham, Revisiting floorplan representations, in: Proceedings of International Symposium on Physical Design, 2001, pp. 138–143; E.F.Y. Young, C.C.N. Chu, Z.C. Shen, Twin binary sequences: a non-redundant representation for general non-slicing floorplan, in: Proceedings of International Symposium on Physical Design, 2002, pp. 196–201] is proposed. The estimations are based on the wire densities (number of wires passing through per unit length) on the half-perimeter boundaries of different regions in a floorplan. These regions are defined naturally by the TBT representation. Buffer planning is also considered by deciding if buffers can be inserted successfully for each net. In order to increase the efficiency of our floorplanner, a fast algorithm for the least common ancestor (LCA) problem in Bender and Farach-Colton [The LCA problem revisited, in: Latin American Theoretical INformatics, 2000, pp. 88–94] is used to compute wire density, and a table look-up approach is used to obtain the buffer insertion information. Experimental results show that our floorplanner can reduce the number of unroutable wires. The performance is comparable with other interconnect-driven floorplanners that perform global routing-like operations directly to estimate routability, but our estimation method is much faster and is scalable for large complex circuits.

1. Introduction

In the deep-submicron era, the complexities of VLSI circuits are growing rapidly. Interconnections between modules become longer and denser. As floorplanning is an early step in the VLSI design cycle, an interconnect-optimized floorplan is important for the applicability and performance of the later designing stages and, most importantly, will allow timing closure to be achieved earlier. Interconnect optimization in floorplan design has become ever more important than before.

Some routability-driven floorplanners [3,12,7] were proposed. Most of them used the grid-based approach to measure congestion. 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. In paper [3], a floorplan is divided into grids and congestion is estimated at each grid assuming that all the wires are routed in either L-shape or Z-shape. They use simple-geometry routing to plan the wires in a reasonable amount of time. In paper [12], a realistic global router is used to evaluate congestion of a placement solution. In paper [7], a probabilistic method is proposed to estimate routability. These grid-based approaches have been shown to be effective in reducing interconnect cost but their computational complexities are usually high. A fast and accurate congestion evaluation method will be very useful for interconnect-driven floorplanning.

☆ The work described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. 4181/06).

* Corresponding author.
E-mail address: fyyoung@cse.cuhk.edu.hk (E.F.Y. Young).

0167-9260/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.vlsi.2009.03.001

Buffer insertion is one of the most popular and effective techniques [4] to achieve timing closure in circuit design. In current practices, buffers are inserted after routing. However, buffers also take up silicon resources and cannot be inserted wherever we want. A good planning of the module positions during the floorplanning stage so that buffers can be inserted wherever needed in the later routing stages will be useful. Cong et al. [5] defined in their paper the term *feasible region* of a net. The locations of these regions can be computed and buffers are clustered into blocks in these regions along the channel areas. Sarkar et al. [8] added into the notion of independence to feasible regions so that the feasible regions of different buffers on a net can be computed independently. They also tried to improve routability by considering congestion in their objective function. Tang and Wong [11] proposed an optimal algorithm to assign buffers to buffer blocks. Dragan et al. [6] used a multi-commodity flow-based approach to allocate buffers to some pre-existing buffer blocks such that the required upper and lower bound on intervals between buffers can be satisfied as much as possible. Alpert et al. [1] made use of tile graph and dynamic programming to perform buffer block planning. The papers [9,13]considered congestion and buffer positions simultaneously. They assumed the *variable interval buffer insertion constraint* which is firstly introduced in [9], i.e., buffers are constrained to be inserted for long enough wires such that the distance between adjacent buffers is lying within a range of $[L, U]$ given by the user. This constraint in buffer locations provides flexibilities for the later routing stages and allows the users to specify their requirements accordingly. The paper [9] uses probabilistic approach while the paper [13] computes the best possible buffer locations for each net and estimates congestion based on those buffer locations.

1.1. Our contributions

In order to provide a simple and efficient method to estimate congestion, an indirect measurement, *wire density*, is proposed. Instead of estimating congestion by counting the number of wires passing through each grid using a global routing-like approach, we measure congestion by wire density, i.e., the average number of wires passing per unit length of the boundaries of different regions in the floorplan. A floorplan with a higher wire density on average will have a higher chance of having congestion problem. An example is shown in Fig. 1. We use twin binary trees as the floorplan representation because the regions to be considered can be naturally defined by the corresponding twin binary trees (TBT). For a floorplan with n modules, $n-1$ regions are defined by each tree. In order to provide more regions for evaluation, we construct an additional pair of trees, which is the mirror of the original pair of trees. For buffer planning, we adopted the variable interval buffer insertion constraint and buffer planning is considered by deciding if buffers can be inserted successfully for each net. In order to increase the efficiency of our floorplanner, a fast algorithm [2] for the least common ancestor (LCA) problem is used to compute wire densities, and a table look-up approach is used to obtain the buffer insertion information. Experimental results show that our floorplanner can reduce the number of unroutable wires. The performance is comparable with other interconnect-driven floorplanners that perform global routing-like operations directly to estimate routability, but our estimation method is much faster and is scalable for large complex circuits.

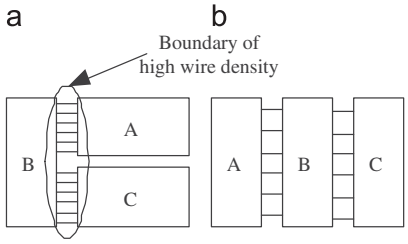

Fig. 1. Higher wire density in floorplan A.

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 Sections 4 and 5, the ideas and implementation details of the congestion evaluation method and the buffer planning method will be described and explained. Experimental results will be shown at the end.

2. Twin binary trees

In our floorplanner, we use twin binary trees as our floorplan representation. The TBT floorplan representation was first proposed in paper [14]. The one-to-one mapping between TBT and mosaic floorplan is shown in [14]. Recall that the definitions of mosaic floorplans and twin binary trees are as follows:

Definition 1. Mosaic floorplan is a floorplan satisfying the following three properties:

1. There is no empty room in the floorplan and each room is assigned to one and only one block. In the floorplan, except the four corners of the chip, the segment intersection forms a T-junction. A T-junction is composed of a non-crossing segment and a crossing segment. The non-crossing segment has one end touching the crossing segment.
2. The topology is equivalent before and after the non-crossing segment of a T-junction slides to adjust the room size.
3. There is no degenerate case where two distinct T-junctions meet at the same point.

Definition 2. The set of twin binary trees $TBT_n \subseteq Tree_n \times Tree_n$ is the set:

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

where $Tree_n$ is the set of all 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 (right) child, a bit 0 (1) is added to the sequence. The first 0 and the last 1 in the sequence are then omitted. If a pair of trees (t_1, t_2) are twin binary to each other, their labellings will be complement to 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 module at the upper right corner of the packing. By connecting the upper right corners of all the modules with the slicelines, t_1 can be constructed by representing the horizontal slicelines by tree edges connecting a parent to its left child, and the vertical slicelines by tree edges connecting 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. It is also observed that the in-order traversal of this pair of trees is the same [15]. An example is shown in Fig. 2. In this example, $\theta(t_1) = 10010$ and $\theta(t_2) = 01101$, so $\theta(t_1) = \theta^c(t_2)$, and their in-order traversals are both *ABCFDE*.

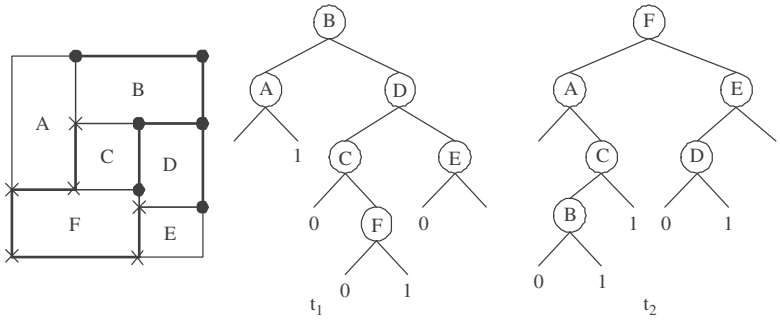

Fig. 2. Construction of TBT.

3. Overview of our floorplanner

Our floorplanner is based on simulated annealing using the TBT floorplan representation. Given a candidate floorplan solution, the total wire length of the nets is either estimated by the half-perimeter bounding box approach or the minimum spanning tree approach. The congestion cost is then estimated from the wire densities which are computed as the number of nets passing per unit length of the boundaries of different regions in the floorplan. These regions are defined by the TBT naturally and hierarchically. The computations of the wire densities 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 computation. 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. For buffer planning, a table is constructed that records whether a net connecting one location to another can have all its buffers inserted successfully. We only need to construct such a table once for each candidate floorplan solution and it can be used repeatedly for all the nets. The buffer insertion cost is then estimated according to the number of nets with unsuccessful buffer insertion.

4. Congestion estimation

In order to estimate the wiring congestion of a floorplan solution efficiently, an indirect but effective evaluation method is used. This approach aims at measuring congestion by the wire densities (number of nets passing per unit length) on the boundaries of different regions in a floorplan.

Definition 3. 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.

For the packing shown in Fig. 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 $R(r)$ where r is the root represents the whole packing and there will be no nets passing through the boundary to outside. We calculate the wire density of $R(i)$ as follows:

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

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

We choose TBT as the floorplan representation in our floorplanner because the regions for evaluation can be defined naturally by TBT. Besides, a lot of fast and simple tree algorithms can be used in our congestion evaluation process. The computations of N_i and P_i will start from the leaf nodes and follow the post-order traversal of the tree to compute the terms at each node i. By dynamic programming, the information computed at the child nodes can be used to compute the N_i and P_i at the parent node very efficiently.

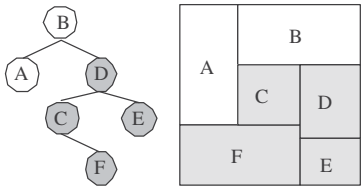

Fig. 3. Region $R(D)$.

4.1. Computation of N_i

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

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

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 (given by the netlist specification) and M'_i is an offset for adjustments due to net sharing between module i, region $R(l(i))$ and region $R(r(i))$. There are two kinds of net sharing, called net completion and net merging. A net is completed at i if it connects module i with some modules in $R(l(i)) \cup R(r(i))$ or it connects some modules in $R(l(i))$ with some modules in $R(r(i))$, and it is not connected to any module outside $R(i)$. Net merging is defined similarly except that the net will continue to go outside $R(i)$ to connect some modules there. M'_i can be computed as

$$M'_i = \sum_{j=2}^{3}(j-1)m_j^i + \sum_{j=2}^{3} j \cdot c_j^i \tag{3}$$

where m_j^i and c_j^i are the numbers of net merging and completion at i, respectively, due to net sharing between j out of the three locations: module i, region $R(l(i))$ and region $R(r(i))$. Note that the value of j can be 2 or 3 only because a net merged or completed at i can either come from module i, region $R(l(i))$ or region $R(r(i))$. In the following, we call a portion of a net connection a subnet.

The adjustment for net merging m_j^i is needed because the repeated countings of one single net in N'_i, $N_{l(i)}$ and $N_{r(i)}$ will overestimate the term N_i. For $j=2$, two subnets of a single net coming from module i, $R(l(i))$ or $R(r(i))$ are merged. For $j=3$, three subnets of a single net coming from module i $R(l(i))$ and $R(r(i))$ are merged.

The term m_j^i is multiplied by $j-1$ because we need to keep one counting in N_i. An example is shown in Fig. 4. In Fig. 4, we consider the situation when module D is reached 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 net between module D and $R(E)$ and one net between $R(C)$ and $R(E)$, so $m_2^D = 3$. There is also one net merged between module D, $R(C)$ and $R(E)$, so $m_3^D = 1$.

Similarly, the adjustment for net completion c_j^i is needed because the repeated countings of one single net in N'_i, $N_{l(i)}$ and $N_{r(i)}$ will over-estimate the term N_i. The value j in c_j^i has the same meaning as that in m_j^i. The term c_j^i is multiplied by j because the net has completed and all the countings should be eliminated. In Fig. 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 $c_2^D = 1+2+3 = 6$. There is also one net completed between module D, $R(C)$ and $R(E)$, so $c_3^D = 1$. Finally, $M'_D = m_2^D + 2m_3^D + 2c_2^D + 3c_3^D = 3 + 2(1) + 2(6) + 3(1) = 20$.

In Fig. 4, N_D is computed as $N'_D + N_C + N_E - M'_D$, where $N'_D = 10$ (by counting the number of nets connected to module D in the netlist specification), $N_C = 13$, $N_E = 11$ and $M'_D = 20$. As a result, $N_D = 10 + 13 + 11 - 20 = 14$. There are 14 nets passing through the boundary of $R(D)$ to outside. However, the term M'_i will vary for different packings, a naive method to compute M'_i will impose an $O(mn)$ run time complexity where m is the total number of nets and n 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 problem to compute M'_i. Details of the implementation will be given in Section 4.4.

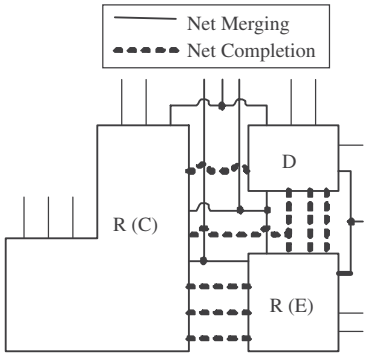

Fig. 4. An example of computing N_D.

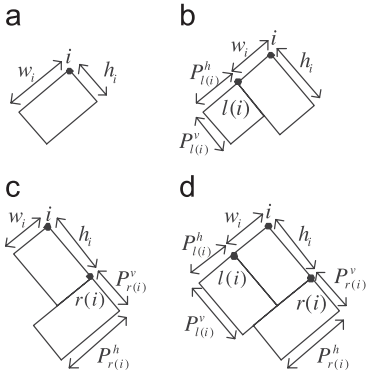

Fig. 5. Cases in the computation of P_i.

4.2. Computation of P_i

The term P_i, 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 widths and the heights of the rooms occupied by the modules, we will also separate the half-perimeter P_i of region $R(i)$ into a horizontal portion P_i^h and a vertical portion P_i^v to make the operation simple. The pseudo-code is given as follows:

ComputeP (tree t)
Assume that $(\pi(1), \pi(2), \ldots, \pi(n))$ *is the post-order traversal of t*
1. *For* $j = 1$ *to n*
2. $i = \pi(j)$
3. *If i is a leaf node*
4. $P_i^h = w_i$
5. $P_i^v = h_i$
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, respectively. The computation of $P(i)$ is divided into four cases. Lines 3–5 is the case in which module i is a leaf node as shown in Fig. 5(a). Fig. 5(b) shows the case when module i has a left child $l(i)$ only (lines 6–8). Fig. 5(c) shows the case when module i has a right child $r(i)$ only (lines 9–11). Lines 12–14 is the last case in which module i has both left child and right child, $l(i)$ and $r(i)$, as in Fig. 5(d). Finally, P_i^h and P_i^v are normalized by the chip width and the chip height, respectively, on line 15 to maintain a uniform order of magnitude. As dynamic programming is applied in the computation, the time complexity of *ComputeP(t)* to compute the normalized half-perimeters of all the $n-1$ regions is only $O(n)$.

4.3. Mirror TBT

According to the definition of the TBT representation, the C_i computed from t_1 represent the wire densities of the boundaries facing the lower left direction, while those computed from t_2 represent the wire densities of the boundaries facing the upper right direction. Each tree can provide $n-1$ statistical samples for wire density evaluation where n is the number of modules. In order to increase the effectiveness of our evaluation method, a pair of *mirror TBT*, which are constructed from the original pair of TBT, are considered. The mirror TBT is the TBT constructed from a packing which is rotated by 90° counterclockwise. Together with the mirror TBT, our evaluation method computes $4(n-1)$ wire density values considering all four routing directions. As sufficient statistical samples are considered, the routability of a packing can be estimated correctly.

4.4. Efficient calculation of N_i

In this section, a detailed explanation of how the LCA algorithm can be used to compute the term N_i efficiently 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. In our approach, instead of computing M'_i for each module i separately, we are going to compute all the M'_is

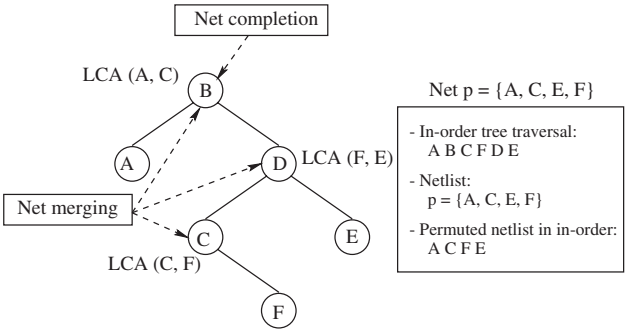

Fig. 6. Using LCA to compute N_i.

simultaneously by visiting each net one by one. The M_i's are all initialized to zero at the beginning. For each net visit, we will update the values of some M_i's. To consider the effect of a net on the M_i values, we can look at an example shown in Fig. 6. In this example, the net p will affect the values of M_B', M_C' and M_D' at nodes B, C and D, respectively, since p will merge and complete at those nodes (so adjustments are needed). Net p will merge at nodes B, C and D, and finally complete at B. In general, the nodes where adjustments are needed are $LCA(u, v)$, where u and v are some modules connected by the net, e.g., $LCA(C, F) = C$, $LCA(E, F) = D$ and $LCA(A, E) = B$. However, we cannot get the correct LCAs where adjustments are needed by 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 shown in Fig. 6 are $LCA(A, C) = B$, $LCA(C, E) = D$ and $LCA(E, F) = D$, which are not correct. The following lemma is useful for finding the correct set of LCAs where adjustments are needed due to a net.

Lemma 1. *Given a tree t with n nodes representing n modules and a net p connecting k modules $m_1, m_2, \ldots, m_k$ in the tree t. The set of nodes L_p in t where two or more subnets of p meet (so 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 in-order traversal of t. (For example, in Fig. 6, the permutation of the modules connected by p following the in-order traversal of the tree is ACFE and $L_p = \{B, C, D\}$ since $LCA(A, C) = B$, $LCA(C, F) = C$ and $LCA(F, E) = D$.)

Proof. The proof can be done by induction on the number of pins of net p. The proof for the base case that the net has only two pins is obvious. Assume that the statement is true for all nets with k or less pins. Consider a net p connecting $k+1$ modules permuted as $m_{\pi(1)}, m_{\pi(2)}, \ldots, m_{\pi(k+1)}$ by following the in-order traversal of the tree t. Consider a net p' that connects the k modules $m_{\pi(1)}, m_{\pi(2)}, \ldots, m_{\pi(k)}$. According to the inductive hypothesis, $L_{p'} = \bigcup_{i=1}^{k-1} \{LCA(m_{\pi(i)}, m_{\pi(i+1)})\}$. It is obvious that $L_{p'} \subset L_p$. There is one more subnet in p that is not in p', which is the subnet connecting to module $m_{\pi(k+1)}$. Consider the positions of module $m_{\pi(k+1)}$ and module $m_{\pi(k)}$ in t. There are only two cases as $m_{\pi(k+1)}$ follows $m_{\pi(k)}$ in the in-order traversal. In the first case, node $m_{\pi(k)}$ is an ancestor of node $m_{\pi(k+1)}$. In this case, the subnet from $m_{\pi(k+1)}$ will merge with the subnet from $m_{\pi(k)}$ at node $m_{\pi(k)}$. The claim is correct since $LCA(m_{\pi(k)}, m_{\pi(k+1)}) = m_{\pi(k)}$.

In the second case, node $m_{\pi(k)}$ is not an ancestor of node $m_{\pi(k+1)}$. Let $x = LCA(m_{\pi(k)}, m_{\pi(k+1)})$. Consider the subtree t_1 rooted at $l(x)$ and the subtree t_2 rooted at $r(x)$. Notice that $m_{\pi(k)}$ is in t_1, $m_{\pi(k+1)}$ is in t_2 and there is no other modules in t_2 except $m_{\pi(k+1)}$ that is connected by p. In this case, the subnet from $m_{\pi(k+1)}$ will merge with the subnet connecting the modules in t_1 at node x. The claim is also correct since $LCA(m_{\pi(k)}, m_{\pi(k+1)}) = x$ □

After obtaining the set L_p for each net p, we can update the values of the corresponding M_i's. As shown in Fig. 6, M_B', M_C' and M_D' will be incremented by 1 because net p will be merged at those nodes. Finally, the value of M_j' for the shallowest module j (in terms of its depth in the tree) among all the modules in the set L_p will be further incremented by 1 because the net will be completed there. For the example shown in Fig. 6, the shallowest module in L_p is B, so M_B' is further increased by 1. We repeat this process of finding LCAs and updating M_i's for each net. Finally, we can apply Eq. (2) to compute all the N_i values for wire density computation.

In paper [2], 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*) technique for the RMQ problem, the LCA problem can be solved in constant time after a preprocessing time of $O(n \log n)$ using dynamic programming.

5. Buffer planning

Our floorplanner tries to plan the module positions such that the variable interval buffer insertion constraint is satisfied, i.e., buffers are constrained to be inserted for long enough wires such that the distance x between adjacent buffers (or between the source or sink node and an adjacent buffer) is bounded, i.e., $x \in [L, U]$, where L and U are given by the users. We divide a candidate floorplan solution into a two-dimensional grid structure and two look-up tables are constructed to store the routability information between every pair of grids that has a net connection between them. We will then count the number of *blocked nets*, i.e., a net that cannot have all its buffer inserted successfully, by looking up the two tables and this counting will be optimized in the objective function of the annealing process.

5.1. Look-up table and feasible grids

In order to perform buffer planning efficiently, we will not compute the exact buffer locations for each net. Instead, we are only interested in knowing whether a net has a good chance of having all its required buffers inserted successfully. Two look-up tables are constructed using dynamic programming that stores this routability information between every pair of grids that has a net connection between them. One table considers the nets running from (to) upper left to (from) lower right, while the other one considers those nets running from (to) lower left to (from) upper right. We define a list F of *feasible grids* as follows:

Definition 4. A grid G is a feasible grid if

1. G contains a terminal of a net, or
2. G has sufficient empty space to hold α buffers,

where α is a user specified constant.

Therefore, a grid is feasible if it is the starting or the ending point of a wire connection or it has sufficient space to hold at least a certain number of buffers. Since we will not compute the exact buffer locations nor count the number of buffer insertions in order to achieve a fast evaluation, the parameter α is used to control the estimation process so that the routability of a floorplan solution can be predicted more accurately.

Definition 5. Given a floorplan with a set F of feasible grids, a routability look-up table L is a two-dimensional table:

$$L[i,j] = \{0 \text{ or } 1 | i,j \in F\}$$

such that $L[i,j] = 0$ if a net or subnet connecting grids i and j is blocked, and $L[i,j] = 1$ otherwise.

Recall that a grid containing a terminal of a net is also a feasible grid, we can thus determine if a net is blocked instantly by checking with the look-up tables. Two look-up tables L_1 and L_2 are used to consider two different routing directions. L_1 considers the nets routing from (to) upper left to (from) lower right while L_2 considers the nets routing from (to) lower left to (from) upper right.

5.2. Construction of the look-up tables

We can construct a look-up table by scanning each feasible grid once by dynamic programming. In the following, we will explain the construction based on table L_1. L_2 can be constructed similarly. L_1 is a $q \times q$ table of bit 0 or 1 where q is the size of the set F of feasible grids. Now we assume that the grids in F are totally ordered in a non-descending order of their shortest Manhattan distances from the lower left corner of the floorplan and then in a non-descending order of their shortest distances from the lower boundary when two or more grids have the distance from the lower left corner. We denote this ordered list as $F(1), F(2), \ldots, F(q)$. L_1 can be constructed by visiting these feasible grids in the above order.

Each grid $F(i)$ keeps a reachable list $R(i)$ which is empty at the beginning. When we visit a grid $F(i)$, two steps are performed, the forward step and the backward step. In the forward step, we will find all the grids $F(j)$s which are on the upper left of $F(i)$ and at a distance x from $F(i)$ where $x \in [L,U]$. We will put $F(i)$ in the reachable lists of all those $F(j)$s. In the backward step, we will update the look-up table L_1 according to the reachable list of $F(i)$. For each grid $F(k)$ in the reachable list of $F(i)$, L_1 will be updated as follows:

$L_1[i] = L_1[i] \bigcup \vec{u}_k$
if $F(k)$ does not have sufficient space to hold α buffers
$L_1[i] = L_1[i] \bigcup \vec{u}_k \bigcup L_1[k]$
otherwise

where $L_1[i]$ denotes the entries on row i of table L_1 and $\vec{u}_k$ is a unit vector of q bits with the kth bit set to 1 and the other $q-1$ bits set to 0. This is because if grid $F(k)$ has sufficient empty space, it can act as an intermediate buffer location for a net or subnet connecting $F(i)$ and a grid reachable from $F(k)$, then we need to update $L_1[i]$ according to $L_1[k]$. Notice that $L_1[k]$ is already found when $F(i)$ is visited since $F(k)$ is on the lower right of $F(i)$. After visiting all the grids in F, the construction of L_1 will be completed.

6. Cost function of the annealing process

The simulated annealing process is divided into two phases. In the first phase, the candidate solutions are still very far away from the final solution, so detailed and complicated evaluations are not needed. The cost function in phase one is

$$cost = A + \beta_1 W + \beta_2 D$$

where A is the area of the smallest bounding rectangle of the floorplan, W is the total wirelength estimated by the half-perimeter estimation method, D is the sum of all the wire density values in the floorplan and β_1 and β_2 are the weights. The weights β_1 and β_2 are set in such a way that the ratio of importance of the terms A, W and D are 2:2:1. This ratio of importance is obtained from experiments.

The second phase starts after 40% of the total number of iterations. In the second phase, more accurate estimations are performed and buffer planning is considered. The cost function in this phrase is

$$cost = A + \gamma_1 W' + \gamma_2 D + \gamma_3 B$$

where W' is the total wirelength estimated by the minimum spanning tree approach, B is the total number of blocked nets and γ_1, γ_2 and γ_3 are the weights. These weights are set such that the ratio of importance of the terms A, W', D and C is 2:2:1:1. Again, this ratio of importance is found from experiments.

7. Run time complexity

Efficiency is one of the major advantages of our interconnect-driven floorplanner. In the following, we will analyze the run time complexity of the congestion estimation process and the buffer planning process. Recall from Eq. (1) that the computation of a wire density W_i is divided into two parts, the computation of N_i and the computation of P_i. The operations needed to compute N_i for all i include (i) the construction of the LCA sparse table which takes $O(n \log n)$ time where n is the number of modules, (ii) the computation of M'_i for all i by looking up the LCA sparse table which takes $O(k)$ time where k is the total number of terminals in the netlists, and (iii) the computation of Eq. (2) for all the modules which takes $O(n)$ time, so the total run time of computing all N_i will be $O(n \log n + k)$. On the other hand, the run time complexity of computing P_i for all i is $O(n)$. As a result, the total time taken of the whole congestion estimation process is $O(n \log n + k)$.

For buffer planning, we need to visit each feasible grid once to construct the two routability look-up tables. In each visit, we need to perform a forward step and a backward step, the run times of which are both dependent on the maximum size of a reachable list. The size of a reachable list is a linear function of L, U and g where $g \times g$ is the size of the grid structure into which the floorplan is divided. Since L and U are small constants specified by the users, the run time of each visit can be written as $O(g)$. The run time to construct the two look-up tables is thus $O(g|F|)$ where F is the set of feasible grids. After constructing the look-up tables, we need to check all the nets to count the number of blocked nets and this will take another $O(k)$ time. Therefore the total run time for the whole buffer planning process is $O(g|F| + k)$.

8. Experimental results

We have implemented our floorplanner and compared its performance with a traditional floorplanner and two related previous works, [9,13], since they also assumed the variable interval buffer insertion constraint. The traditional floorplanner is a simulated annealing-based floorplanner that considers area and total wirelength (using half-perimeter estimation) only in the cost function. In order to evaluate the performance of the floorplanners, a simple global router is implemented to route the final floorplan solutions. In this simple global router, each multi-pin net is first decomposed into a set of 2-pin nets. These 2-pin nets are then routed one after another by maze routing. Rip-up and re-route will be done to route as many nets as possible. A wire is *unroutable* if it cannot be routed successfully due to congestion (each grid has a wiring capacity constraint) or buffer insertion failure. Three MCNC benchmarks, *ami32*, *ami49* and *playout*, are used. (There are totally six MCNC benchmarks but the other three

Table 1
Data set specifications.

Data	Number of modules	Number of terminals	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 2
Comparisons between different floorplanners.

Data	Dead space (%)	Wire Length (10^3 μm)	Number of unroutable wires	Run time (s)
ami33				
Traditional[a]	10.31	21.25	15.03	21.05
[9][b]	11.80	20.59	9.63	678.45
[13][c]	12.39	23.21	3.88	290.69
Our floorplanner[a]	11.63	22.18	6.88	653.75
ami49				
Traditional	10.87	386.45	14.53	25.54
[9]	10.80	379.80	10.00	789.46
[13]	11.24	384.55	6.75	369.18
Our floorplanner	18.67	398.39	8.72	688.73
playout				
Traditional	10.25	284.78	170.54	30.64
[9]	10.38	290.56	115.88	3498.23
[13]	11.74	274.74	94.50	912.21
Our floorplanner	16.83	298.90	107.60	720.44
n2000				
Traditional	11.56	102.60	581.75	35.41
[13]	15.35	114.65	416.47	1316.81
Our floorplanner	17.72	114.42	430.17	761.50
n2500				
Traditional	14.08	141.09	887.35	37.54
[13]	16.51	155.37	675.61	1701.46
Our floorplanner	20.26	173.21	694.46	774.83
n3000				
Traditional	16.37	177.77	1299.75	46.95
[13]	18.45	194.47	970.45	1916.85
Our floorplanner	20.88	208.80	988.40	832.05

[a] Pentium IV 1.4 GHz processor and 512 MB memory.
[b] Pentium IV 1.2 GHz processor and 512 MB memory.
[c] Pentium IV 1.4 GHz processor and 256 MB memory.

are very simple and they will not have any routability problem.) In additions, we have generated three heavily connected data sets, *n2000*, *n2500* and *n3000*, to demonstrate the performance of the floorplanners. The detailed specifications of these data sets are shown in Table 1.

Table 2 compares our floorplanner with the traditional floorplanners and those in paper [9] and [13]. We used the data in paper [10] to compute the parameters for the router. For *ami33* and *ami49*, feature values of the 0.18 μm technology were used, while for the other data sets, feature values of the 0.13 μm technology were used. Notice that the numbers of unroutable wires reported in Table 2 are not integers because they are the average obtained by performing the experiments several times. Results show that the number of unroutable wires can be reduced significantly by applying our fast congestion estimation and buffer planning method. In terms of reduction in the number of unroutable wires, our floorplanner is better than the floorplanner in [9] but not as good as the one in [13]. However, if we compare the run times, our floorplanner is found to be the most suitable one for large complex circuits because our run time grows very slowly. We did not compare with the floorplanner in [9] for the three complicated data sets because their results cannot be obtained in a reasonable amount of time.

9. Conclusion

In this paper, we presented an interconnect-driven floorplanner that considers both wire congestion and buffer planning. The major advantages of our floorplanner are its accuracy and its scalability to large complex circuits. We measure congestion as the average number of wires passing through the boundaries of different regions of the floorplan. Buffer planning is performed by constructing routability look-up tables from which we can determine approximately the number of blocked nets due to unsuccessful buffer insertions. Experimental results have shown that the performance of our floorplanner is comparable with other interconnect-driven floorplanners that perform global routing-like operations directly to estimate routability. However, our method is good for large complex circuits since the run time of our floorplanner will grow much slower than that of the other floorplanners when the complexity of the circuit increases.

References

[1] C.J. Alpert, J. Hu, S.S. Sapatnekar, P.G. Villarrubia, A practical methodology for early buffer and wire resource allocation, in: Proceedings of the 38th ACM/IEEE Design Automation Conference, 2001, pp. 189–194.
[2] M.A. Bender, M. Farach-Colton, The LCA problem revisited, in: Latin American Theoretical INformatics, 2000, pp. 88–94.
[3] H.M. Chen, H. Zhou, F.Y. Young, D.F. Wong, H.H. Yang, N. Sherwani, Integrated floorplanning and interconnect planning, in: Proceedings of IEEE International Conference on Computer-Aided Design, 1999, pp. 354–357.
[4] J. Cong, Challenges and opportunities for design innovation in nanometer technologies, SRC Working Papers, 1997.
[5] J. Cong, T. Kong, D.Z. Pan, Buffer block planning for interconnect-driven floorplanning, in: Proceedings of IEEE International Conference on Computer-Aided Design, 1999, pp. 358–363.
[6] F.F. Dragan, A.B. Kahng, S. Muddu, A. Zelikovsky, Provably good global buffering using an available buffer block plan, in: Proceedings of the International Conference on Computer-Aided Design, 2000, pp. 104–109.
[7] S. Krishnamoorthy, J. Lou, H.S. Sheng, Estimating routing congestion using probabilistic analysis, in: Proceedings of International Symposium on Physical Design, 2001, pp. 112–117.
[8] P. Sarkar, V. Sundararaman, C.K. Koh, Routability-driven repeater block planning for interconnect-centric floorplanning, in: International Symposium on Physical Design, 2000, pp. 186–191.
[9] C.W. Sham, E.F.Y. Young, Routability driven floorplanner with buffer block planning, in: Proceedings of International Symposium on Physical Design, 2002, pp. 50–55.
[10] D. Sylvester, K. Keutzer, Getting to the bottom of deep submicron, in: Proceedings of the International Conference on Computer-Aided Design, 1998, pp. 203–211.
[11] X. Tang, D.F. Wong, Planning buffer locations by network flows, in: International Symposium on Physical Design, 2000, pp. 180–185.
[12] M. Wang, M. Sarrafzadeh, Modeling and minimization of routing congestion, in: IEEE Asia and South Pacific Design Automation Conference, 2000, pp. 185–190.
[13] K.K.C. Wong, E.F.Y. Young, Fast buffer planning and congestion optimization in interconnect-driven floorplanning, in: Proceedings of 8th Asia and South Pacific Design Automation Conference, 2003, pp. 411–416.
[14] B. Yao, H. Chen, C.K. Cheng, R. Graham, Revisiting floorplan representations, in: Proceedings of International Symposium on Physical Design, 2001, pp. 138–143.

[15] E.F.Y. Young, C.C.N. Chu, Z.C. Shen, Twin binary sequences: a non-redundant representation for general non-slicing floorplan, in: Proceedings of International Symposium on Physical Design, 2002, pp. 196–201.

Steve T.W. Lai: Steve T.W. Lai received his B.Sc. and M.Phil. degrees in Computer Science and Engineering, in 2001 and 2003, respectively, from the Chinese University of Hong Kong, Shatin, N.T., Hong Kong. His research interests include floorplanning and algorithms.

Evangeline F.Y. Young: Evangeline Young received her B.Sc. degree and M.Phil. degree in Computer Science from The Chinese University of Hong Kong (CUHK). She received her Ph.D. degree from The University of Texas at Austin in 1999. Currently, she is an associate professor in the Department of Computer Science and Engineering in CUHK. Her research interests include algorithms and CAD of VLSI circuits. She is now working actively on floorplanning, placement and routing. Dr. Young has served on the technical program committees of several major conferences including ICCAD, ASP-DAC, ISPD and GLSVLSI.

Chris C.N. Chu: Chris Chu received the B.S. degree in Computer Science from the University of Hong Kong, Hong Kong, in 1993. He received the M.S. degree and the Ph.D. degree in Computer Science from the University of Texas at Austin in 1994 and 1999, respectively. Dr. Chu is currently an Associate Professor in the Electrical and Computer Engineering Department at Iowa State University. His area of expertises include CAD of VLSI physical design, and design and analysis of algorithms. His recent research interests are performance-driven interconnect optimization and fast circuit floorplanning, placement, and routing algorithms. He received the IEEE TCAD best paper award in 1999 for his work in performance-driven interconnect optimization. He received the ISPD best paper award in 2004 for his work in efficient placement algorithm. He received the Bert Kay Best Dissertation Award for 1998-1999 from the Department of Computer Sciences in the University of Texas at Austin. Dr. Chu has served on the technical program committees of several major conferences including ICCAD, ISPD, ISCAS, DATE, ASP-DAC, and SLIP. He has also served as an organizer for the ACM SIGDA Ph.D. Forum.