Routability-Driven Floorplanner With Buffer Block Planning

Chiu-Wing Sham and Evangeline F. Y. Young

***Abstract*—In traditional floorplanners, area minimization is an important issue. However, due to the recent advances in very large scale integration technology, the number of transistors in a design are increasing rapidly and so are their switching speeds. This has increased the importance of interconnect delay and routability in the overall performance of a circuit. We should consider interconnect planning, buffer planning, and routability as early as possible. In this paper, we study and implement a routability-driven floorplanner with congestion estimation and buffer planning. Our method is based on a simulated annealing approach that is divided into two phases: the area optimization and congestion optimization phases. In the area optimization phase, modules are roughly placed according to the total area and wirelength. In the congestion optimization phase, a floorplan will be evaluated by its area, wirelength, congestion, and routability. We assume that buffers should be inserted at flexible intervals from each other for long enough wires and probabilistic analysis is performed to compute the congestion information taken into account the constraints in buffer locations. Our approach is able to reduce the average number of wires at the congested areas and allow more feasible insertions of buffers to satisfy the delay constraints without having much penalty in increasing the area of the floorplan.**

***Index Terms*—Buffer planning, computer-aided design, floor planning, interconnect-driven, physical design, very large scale integration (VLSI).**

I. Introduction

A. Motivation

FLOORPLANNING plays an important role in the physical design of very large scale integration circuits. It plans the shapes and locations of the modules on a chip, the result of which will greatly affect the overall performance of the final circuit. In the past, area minimization was the major concern in floorplan design. Advances in the deep submicrontechnology have brought many changes and challenges to this. As technology continues to scale down, the sizes of transistors are getting smaller and a significant portion of circuit delay is coming from interconnects. In some advanced systems today, as much as 80% of the clock cycle is consumed by interconnects [2]. Area minimization is less important while routability and delay has become the major concern in floorplanning and many other designing steps.

Traditional floorplanners have not paid enough attention to interconnect optimization. This results in a large expansion in area, or even in an unroutable design failing to achieve timing closure after detailed routing. Buffer insertion is one of the most popular and effective techniques to achieve timing closure. It was projected that over 700 K repeaters will be inserted on a single chip in the 70-nm technology [3]. In current practices, buffers are inserted after routing. However buffers also take up silicon resources ($40\times$ to $200\times$ minimum inverter size [5]) 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 very useful. Besides, buffers also contribute delay and area, and their locations should be carefully planned.

B. Previous Works

There are several previous works addressing the interconnect issues in floorplan design. In [2], a floorplan is divided into grids and congestion is estimated at each grid, assuming that each wire is routed in either an L- or Z-shape. They use a simple and fast method to estimate congestion but buffer insertion is not considered. Cong *et al.* define in their paper [4] the term *feasible region* of a net, that is, the largest polygon in which a buffer can be inserted such that the timing constraint can be satisfied. The locations of these feasible regions can be computed and buffers are clustered into blocks in these feasible regions along the channel areas. Sarkar *et al.* [11] add the notion of independence to feasible regions so that the feasible regions of different buffers on a net can be computed independently. They also try to improve routability by considering congestion in their objective function. Tang and Wong [12] propose an optimal algorithm to assign buffers to buffer blocks assuming that only one buffer is needed per net. In [13], a realistic global router is used to evaluate the congestion information of each candidate placement solution. Dragan *et al.* [5] use a multicommodity flow-based approach to allocate buffers to some pre-existing buffer blocks such that the required upper and lower bounds on buffer intervals can be satisfied as much as possible. Alpert *et al.* [1] make use of tile graph and dynamic programming to perform buffer block planning. They propose that buffers should be allowed to be inserted inside macro blocks and their method will distribute buffer sites throughout the layout. Finally, Lou *et al.* [9] apply probabilistic analysis to estimate congestion and routability, and they show that their estimations correlate well with postroute

Manuscript received June 1, 2002; revised September 3, 2002 and December 8, 2002. The work described in theis paper was substantially supported by a grant from the the Research Grants Council of the Hong Kong Special Administrative Region (Project no. CUHK4231/OIE). This paper was recommended by Guest Editor C. J. Alpert.

C.-W. Sham and E. F. Y. Young are with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Shatin, N.T., Hong Kong (e-mail: fyyoung@cse.cuhk.hk).

Digital Object Identifier 10.1109/TCAD.2003.809649

congestion. However, their congestion model does not take into account buffer insertions.

C. Our Contributions

In order to address the interconnect issues in floorplan design, we propose to use a probabilistic method to estimate congestion and routability with buffer insertions. Probabilistic analysis can address the problem in a more global way and can thus give a more accurate estimation. In our model, we assume that buffers are constrained to be inserted for long enough wires such that the distance between adjacent buffers is lying within a range [*low, up*] given by the user. We call this the *variable interval buffer insertion constraint* [1], [5]. For example, according to [8], global repeater rules for a high-end microprocessor design in the 0.25-μm CMOS technology require repeaters at intervals of at most 4500 μm, and we can model this situation by assigning the *low* and *up* appropriately. In our floorplanner, we will divide a floorplan into a two-dimensional grid structure and will estimate congestion at each grid subjecting to the given buffer insertion constraint. Notice that the congestion at each location is dependent on the routes of the wires while the route of a wire is, in turn, dependent on the availability of buffer resources along the route. This availability of buffer resources can be estimated from the amount of empty space and the number of possible buffer insertions at that location. We compute the congestion information assuming that every route is equally likely to be used as long as the buffer insertion constraint can be satisfied. The computation can be performed efficiently by using dynamic programming and a table lookup approach. Besides, we have used a two stage simulated annealing method to speed up the whole floorplanning process. In order to verify the usefulness of our method, we have implemented a simple global router for testing and the experimental results can demonstrate the effectiveness and efficiency of our floorplanner.

This paper is divided into eight sections. In Section II, an overview of our floorplanner will be given. In Sections III–VII, the ideas and the implementation details including buffer planning, congestion estimation, and the two-phase simulated annealing process will be described and explained. Finally, experimental results and comparisons between our floorplanner and a traditional floorplanner will be shown in Section VIII.

II. Overview of Our Floorplanner

In this section, we will give a brief overview of our routability-driven floorplanner. We assume that wires are routed over-the-cell and multibend routing is used. We divide a floorplan into grids and the size of the deadspace in each grid is computed for estimation of the amount of buffer resources. Given a set of m nets and a set of n modules where each module M_i has an area A_i, we want to obtain a nonoverlapping packing of these modules such that the area of the packing, the interconnect and congestion costs are small, every net satisfies its buffer insertion constraint, and every module satisfies its area and aspect ratio constraint.

In our floorplanner, given a candidate floorplan solution, we will first estimate the buffer usage of each net k at each grid, then we can estimate the total buffer usage at each grid,

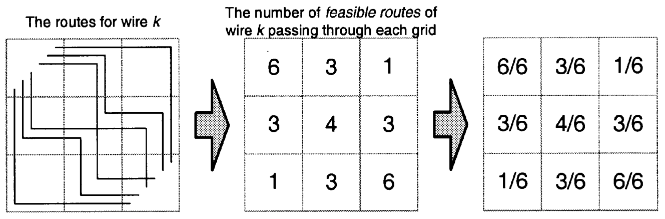


Fig. 1. Commonly used congestion model.

b_usage(x, y). We assume that buffers should be inserted at flexible intervals from each other for long enough wires. By making use of b_usage(x, y) and the estimation on the amount of buffer resources $\big(b$_space(x, y) b_space(x, y) is estimated from the size of the deadspace at $(x, y)\big)$ we can estimate the probability of successful buffer insertion b_success(x, y) at each grid. We can then make use of this information to estimate congestion, congestion(x, y). In order to improve the routability of the design output, our floorplanner will try to reduce the routing congestion at different locations of the layout and to increase the number of nets with successful buffer insertions.

III. Congestion Model

In order to improve the routability of the floorplan solution, a congestion model is used to estimate the wiring information of a candidate solution. By using a good congestion model, the intermediate floorplan solutions can be evaluated accurately with respect to their congestion and routability. The paper [9] shows that their probabilistic congestion model correlates well to postroute results but their work does not consider buffer locations. This motivates us to use a probabilistic approach in our floorplanner. Nevertheless, as clock frequencies reach and exceed the gigahertz level, each top level global net must undergo buffer insertion to maintain signal integrity and reasonable signal delay. The number of buffer insertions is large, but buffers cannot be inserted wherever we want. Buffer planning in the early designing stage is very important and the rest of the design flow will be benefited by a proper planning and usage of the buffer resources. In this section, we assume that all wires are two-pin and we will discuss how we handle multipin nets in Section VII.

A commonly used congestion model is shown in Fig. 1. We can see that the probability for each possible route of a wire k is equal. However, if the probabilities of successful buffer insertion (dependent on the amount of buffer resources) are different at different grids, the probabilities of some routes will be higher if they run through those locations with higher probabilities of successful buffer insertion.

In our approach, we try to address the buffer-insertion issue with a new congestion model. In this model, adjacent buffers are required to be inserted at a distance $x \in [low, up]$ from each other where *low* and up are the lower and upper bounds of the buffer insertion constraint aregiven by the users. In Fig. 2, we assume that both the upper and lower bounds of the buffer insertion constraint is equal to two, i.e., buffers should be inserted at an interval of two grid units from each other. We can see that the probabilities of the routes b, c, d, and e with successful buffer

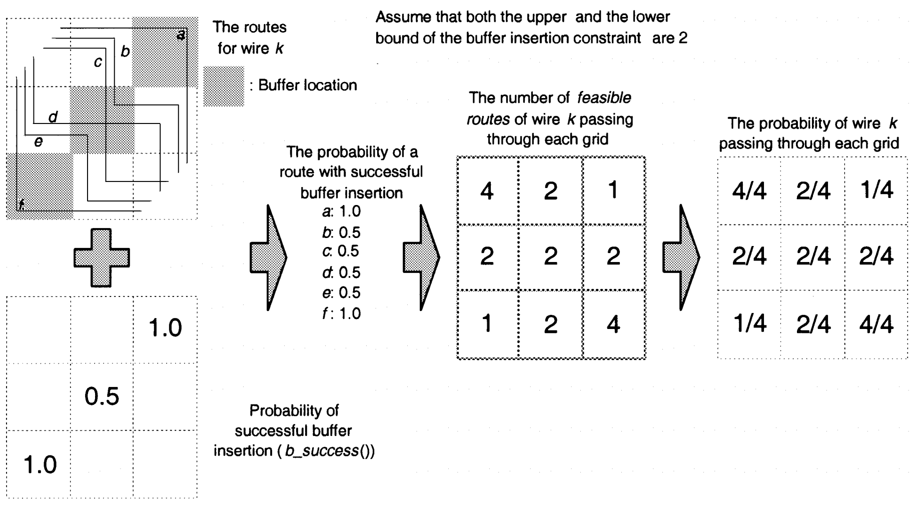

The routes for wire k
a
b
c
d
e
f
: Buffer location
Assume that both the upper and the lower bound of the buffer insertion constraint are 2
The probability of a route with successful buffer insertion
a: 1.0
b: 0.5
c: 0.5
d: 0.5
e: 0.5
f: 1.0
The number of feasible routes of wire k passing through each grid
4 2 1
2 2 2
1 2 4
The probability of wire k passing through each grid
4/4 2/4 1/4
2/4 2/4 2/4
1/4 2/4 4/4
1.0
0.5
1.0
Probability of successful buffer insertion (b_success())

Fig. 2. Simple example of our congestion model.

insertion are only 0.5. Therefore, the total number of feasible routes for this wire is reduced from six to four where the number of feasible routes is computed as the summation of the probabilities of all the possible routes with successful buffer insertion. As a result, the probability of wire k passing through each grid will be different from that computed in Fig. 1. The probabilities of some routes become higher if they run through those locations with higher chances of successful buffer insertion.

In order to compute the congestion information using this model, we need estimates of $b\_\text{usage}(x, y)$ and $b\_\text{success}(x, y)$ at each grid (x, y). (Note that $b\_\text{usage}(x, y)$ is used to compute $b\_\text{success}(x, y)$). In this section, we will show the computation of the congestion information, assuming that $b\_\text{success}(x, y)$ is given for each grid (x, y). We will show how we can estimate $b\_\text{success}(x, y)$ in Section IV.

A. Probabilistic Model With Variable Interval Buffer Insertion Constraint

Given the information on the amount of buffer resources at each grid, we want to calculate the probability $r\_\text{success}(l)$ that a route l can be routed successfully from the source to the sink without violating any buffer insertion constraint. In the example of Fig. 3, the wirelength is six and the upper and lower bounds of the buffer insertion constraint is three and two, respectively. Consequently, there are two feasible ways of buffer insertion for each route. They are shown at the top right corner of Fig. 3 by straightening out a path to one-dimension. We assume that the routes have equal probability of being selected. The probability of the first feasible way of buffer insertion is equal to $b\_\text{success}(x_1, y_1) \times b\_\text{success}(x_2, y_2)$ where (x_1, y_1) and (x_2, y_2) are two and four grid units from the source, respectively, and that of the second one is equal to $b\_\text{success}(x_3, y_3)$ where (x_3, y_3) is three grid units from the source. Given the probability of successful buffer insertion at each grid, the probability of route 1 satisfying the buffer insertion constraint $r\_\text{success}(1)$ can be computed as $(0.2 + 0.5 \times 1)/2 = 0.35$. Similarly, we can calculate $r\_\text{success}(l)$ for all the other routes $l = 2 \ldots 6$.

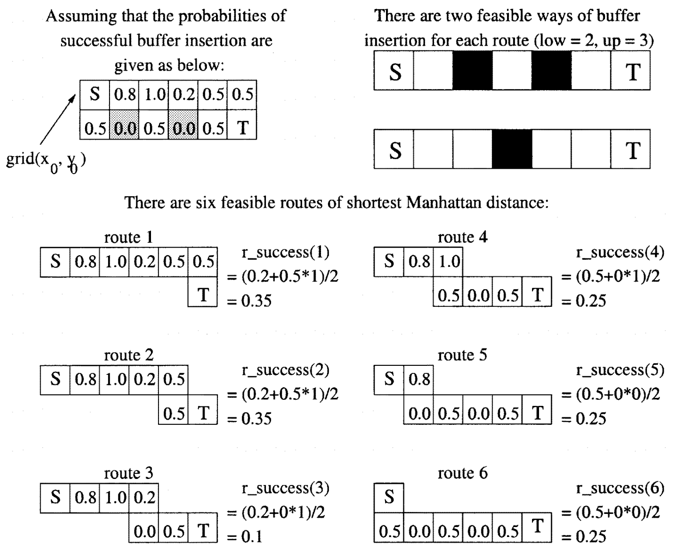

Assuming that the probabilities of successful buffer insertion are given as below:
S 0.8 1.0 0.2 0.5 0.5
0.5 0.0 0.5 0.0 0.5 T
grid(x0, y0)
There are two feasible ways of buffer insertion for each route (low = 2, up = 3)
S T
S T
There are six feasible routes of shortest Manhattan distance:
route 1
r_success(1) = (0.2+0.5*1)/2 = 0.35
route 2
r_success(2) = (0.2+0.5*1)/2 = 0.35
route 3
r_success(3) = (0.2+0*1)/2 = 0.1
route 4
r_success(4) = (0.5+0*1)/2 = 0.25
route 5
r_success(5) = (0.5+0*0)/2 = 0.25
route 6
r_success(6) = (0.5+0*0)/2 = 0.25

Fig. 3. Example of computing $r\_\text{success}(l)$.

Thus, the total number of feasible routes for a wire k is equal to $\sum_{l \in L_k} r\_\text{success}(l)$ where L_k is the set of all routes for wire k, and the number of feasible routes passing through the grid (x, y) is equal to $\sum_{l \in L_k(x,y)} r\_\text{success}(l)$ where $L_k(x, y)$ is the set of all routes for wire k passing through the grid at (x, y). For the example in Fig. 3, $\sum_{l \in L_k} r\_\text{success}(l)$ is equal to $3 \times 0.25 + 2 \times 0.35 + 0.1 = 1.55$. Consider the grid at $(x_0 + 1, y_0)$, routes 1–5 will pass through this grid, so the total number of feasible routes passing through this grid is equal to $2 \times 0.25 + 2 \times 0.35 + 0.1$, i.e., 1.3. Consider the grid at $(x_0, y_0 + 1)$; only route 6 will pass through this grid. Thus, the total number of feasible routes passing through this grid is equal to $r\_\text{success}(6)$, i.e., 0.25.

Consequently, the congestion information $F(x, y, k)$ at grid (x, y) for wire k can be calculated as

$$F(x, y, k) = \frac{\sum\limits_{l \in L_k(x,y)} r\_\text{success}(l)}{\sum\limits_{l \in L_k} r\_\text{success}(l)}$$

where $F(x,y,k)$ is the probability of wire k passing through the grid at (x,y). For example, we can compute $F(x_0+1, y_0, k) = 1.3/1.55$ and $F(x_0, y_0+1, k) = 0.25/1.55$. We can see that $F(x_0+1, y_0, k) + F(x_0, y_0+1, k)$ is equal to one because the routes of wire k must pass through either the grid at (x_0+1, y_0) or (x_0, y_0+1). Finally, the expected number of wires passing through a grid at (x,y) is computed as

$$\text{congestion}(x,y) = \sum_{\text{all wire } k} F(x,y,k).$$

In order to calculate $F(x,y,k)$ efficiently, we use dynamic programming. At the beginning, we are given the probability of successful buffer insertion at each grid. We will then create an array $\text{prob}[0 \ldots up]$ at each grid with size equals one plus the upper bound of the buffer insertion constraint, *up*. Assuming that wire k starts from grid (x_0, y_0) and ends at grid (x_t, y_t) where $x_t > x_0$ and $y_t > y_0$, we will initialize $g_0(x_0, y_0, k).\text{prob}[0]$ to one and the remaining values of the array to zero. The value of $g_0(x,y,k).\text{prob}[i]$ represents the total number of feasible routes for wire k from the source to (x,y) such that the previous buffer is inserted at a grid of distance i units in front of (x,y). We can look at the example in Fig. 3 to illustrate the meaning of this notation. If we consider the fourth grid in the second row, i.e., $x = x_0 + 3$ and $y = y_0 + 1$, and $k = 2$. Only routes 3–6 will pass through this grid, and each route has two feasible ways of buffer insertion. Since $k = 2$, we only consider those with the previous buffer inserted at a distance of two grid units in front. There are only four of them with probabilities 1.0, 1.0, 0.0, and 0.0, respectively. Therefore, $g_0(x_0+3, y_0+1).\text{prob}[2] = 2.0$. The value of the array at each grid can be computed dynamically row by row according to the recursive equation (1), found at the bottom of the page.

Consequently, the value $g_0(x_t, y_t, k).\text{prob}[0]$ can be obtained which represents the total number of feasible routes for wire k from the source (x_0, y_0) to the sink (x_t, y_t). Then, we will repeat the same steps from the sink (x_t, y_t) to the source (x_0, y_0) to calculate $g_1(x,y,k).\text{prob}[i]$ for $0 \le i \le up$ with (2), found at the bottom of the page.

Thus, the total number of feasible routes passing through a grid at (x,y) and a buffer is inserted at (x,y) and can be computed as

$$F_1(x,y,k) = \frac{g_0(x,y,k).\text{prob}[0] \times g_1(x,y,k).\text{prob}[0]}{b\_\text{success}(x,y)}.$$

In addition, the total number of feasible routes passing through a grid at (x,y) and no buffers is inserted at (x,y) can be computed as

$$F_0(x,y,k) = \sum_{i,j} g_0(x,y,k).\text{prob}[i] \times g_1(x,y,k).\text{prob}[j]$$

where $i, j \neq 0$ and low $\le i + j \le up$.

Finally, as the total number of feasible routes running from the source (x_0, y_0) to the sink (x_t, y_t) is $g_0(x_t, y_t, k).\text{prob}[0]$ or $g_1(x_0, y_0, k).\text{prob}[0]$ (note that $g_0(x_t, y_t, k).\text{prob}[0]$ is equal to $g_1(x_0, y_0, k).\text{prob}[0]$), the probability that wire k will pass through the grid at (x,y) can be computed as

$$F(x,y,k) = \frac{F_0(x,y,k) + F_1(x,y,k)}{g_0(x_t, y_t, k).\text{prob}[0]}.$$

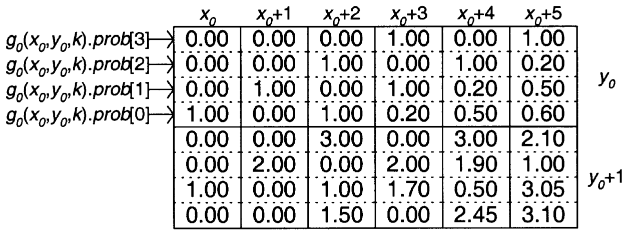
Consider the example in Fig. 3 again: we can compute $g_0(x,y,k).\text{prob}[i]$ and $g_1(x,y,k).\text{prob}[i]$ for $0 \le i \le up$ by (1) and (2). The results are shown in Figs. 4 and 5, respectively. In this case, $F(x_0+1, y_0, k)$ can be calculated as follows:

$$\begin{aligned}
&F_0(x_0+1, y_0, k)\\
&\quad = g_0(x_0+1, y_0, k).\text{prob}[1]\\
&\qquad \times g_1(x_0+1, y_0, k).\text{prob}[1]\\
&\qquad + g_0(x_0+1, y_0, k).\text{prob}[1]\\
&\qquad \times g_1(x_0+1, y_0, k).\text{prob}[2]\\
&\qquad + g_0(x_0+1, y_0, k).\text{prob}[2] \times g_1(x_0+1, y_0, k).\text{prob}[1]\\
&\quad = 1.0 \times 1.0 + 1.0 \times 1.6 + 0.0 \times 1.0\\
&\quad = 2.6\\
&F_1(x_0+1, y_0, k)\\
&\quad = \frac{g_0(x_0+1, y_0, k).\text{prob}[0] \times g_1(x_0+1, y_0, k).\text{prob}[0]}{0.8}\\
&\quad = 0.0\\
&F(x_0+1, y_0, k)\\
&\quad = \frac{F_0(x_0+1, y_0, k) + F_1(x_0+1, y_0, k)}{g_0(x_t, y_t, k).\text{prob}[0]}\\
&\quad = \frac{2.6 + 0.0}{3.1}\\
&\quad = \frac{1.3}{1.55}.
\end{aligned}$$

From the example, we can see that the result of $F(x_0+1, y_0, k)$ obtained by dynamic programming is the same as that obtained

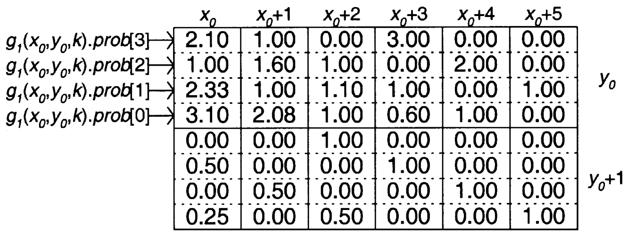
$$g_0(x,y,k).\text{prob}[i] = \begin{cases} g_0(x-1,y,k).\text{prob}[i-1] + g_0(x,y-1,k).\text{prob}[i-1], & \text{for } 1 \le i \le \text{up} \\ \sum\limits_{j=\text{low}}^{up} g_0(x,y,k).\text{prob}[j] \times b\_\text{success}(x,y), & \text{for } i = 0 \end{cases} \tag{1}$$

$$g_1(x,y,k).\text{prob}[i] = \begin{cases} g_1(x+1,y,k).\text{prob}[i-1] + g_1(x,y+1,k).\text{prob}[i-1], & \text{for } 1 \le i \le up \\ \sum\limits_{j=\text{low}}^{\text{up}} g_1(x,y,k).\text{prob}[j] \times b\_\text{success}(x,y), & \text{for } i = 0 \end{cases} \tag{2}$$



	x_0	x_0+1	x_0+2	x_0+3	x_0+4	x_0+5	
$g_0(x_0,y_0,k).prob[3]$	0.00	0.00	0.00	1.00	0.00	1.00	y_0
$g_0(x_0,y_0,k).prob[2]$	0.00	0.00	1.00	0.00	1.00	0.20	
$g_0(x_0,y_0,k).prob[1]$	0.00	1.00	0.00	1.00	0.20	0.50	
$g_0(x_0,y_0,k).prob[0]$	1.00	0.00	1.00	0.20	0.50	0.60	
	0.00	0.00	3.00	0.00	3.00	2.10	y_0+1
	0.00	2.00	0.00	2.00	1.90	1.00	
	1.00	0.00	1.00	1.70	0.50	3.05	
	0.00	0.00	1.50	0.00	2.45	3.10	

Fig. 4. Results of $g_0(x, y, k).\mathrm{prob}[i]$ for $0 \le i \le$ up.



	x_0	x_0+1	x_0+2	x_0+3	x_0+4	x_0+5	
$g_1(x_0,y_0,k).prob[3]$	2.10	1.00	0.00	3.00	0.00	0.00	y_0
$g_1(x_0,y_0,k).prob[2]$	1.00	1.60	1.00	0.00	2.00	0.00	
$g_1(x_0,y_0,k).prob[1]$	2.33	1.00	1.10	1.00	0.00	1.00	
$g_1(x_0,y_0,k).prob[0]$	3.10	2.08	1.00	0.60	1.00	0.00	
	0.00	0.00	1.00	0.00	0.00	0.00	y_0+1
	0.50	0.00	0.00	1.00	0.00	0.00	
	0.00	0.50	0.00	0.00	1.00	0.00	
	0.25	0.00	0.50	0.00	0.00	1.00	

Fig. 5. Results of $g_1(x, y, k).\mathrm{prob}[i]$ for $0 \le i \le$ up.

by counting the routes from Fig. 3. As a result, we can compute $F(x, y, k)$ effectively.

In this section, $r\_\mathrm{success}(l)$ is calculated assuming that all possible ways of buffer insertion are equally likely to be used. However, it is also reasonable to select the route with the highest probability of successful buffer insertion. To achieve this, we can simply replace (1) and (2) with (3) and (4).

B. Time Complexity

According to this congestion estimation model, we need to scan the array at each grid from the source to the destination for each net. Therefore, the time complexity is $O(m \times k \times up^2)$ where k is the number of grids scanned for each net, m is the number of nets, and up is the upper bound of the buffer insertion constraint. If the floorplan is divided into $K \times K$ grids, we need to scan at most K^2 grids for each net. Moreover, the upper bound in the buffer insertion constraint is also bounded by the length of the wires ($2K$). Therefore, the time complexity of this congestion estimation method is $O(m \times K^4)$. However, the value of up, though upper bounded by $2K$, is much smaller than $2K$ (four to ten grids) even when the floorplan is very large. As a result, the method is efficient in practice.

IV. Buffer Planning

In this section, we will show how we can estimate the amount of buffer usage $b\_\mathrm{usage}(x, y)$ and the probability of successful buffer insertion $b\_\mathrm{success}(x, y)$ at each grid (x, y). Such information will be used in the calculation of the congestion information as discussed in Section III.

A. Estimation of Buffer Usage

According to the variable interval buffer insertion constraint, buffers can be inserted at flexible locations as long as adjacent buffers are at a distance $x \in [\mathrm{low}, \mathrm{up}]$ from each other. There are, thus, several feasible ways of buffer insertion satisfying the constraint for each route. The probability that a route l of wire k will insert a buffer at a grid (x, y), $b\_\mathrm{insert}(x, y, l, k)$, should be calculated in order to estimate the amount of buffer usage $b\_\mathrm{usage}(x, y)$. Given a possible route l of wire k with source S and sink T and the buffer insertion constraint $[\mathrm{low}, \mathrm{up}]$, the total number of feasible ways of buffer insertion $\mathrm{total}(l)$ can be obtained. We assume that all feasible ways of buffer insertion are equally likely to be used in the routing stage. The probability that a grid (x, y) is required to have a buffer inserted for a route l of wire k can be calculated as

$$b\_\mathrm{insert}(x, y, l, k) = \frac{b\_\mathrm{count}(d, l)}{\mathrm{total}(l)}$$

where d is the Manhattan distance of the grid (x, y) from the source S, $\mathrm{total}(l)$ is the total number of feasible ways of buffer insertion for l, and $b\_\mathrm{count}(d, l)$ is the number of feasible ways of buffer insertion for route l that will insert a buffer at a grid of distance d from the source S.

In the example of Fig. 6, there are four feasible ways of buffer insertion satisfying the buffer insertion constraint, so $\mathrm{total}(l)$ is equal to four. Consider the grid $(x_0 + 4, y_0)$, its distance from the source S is equal to four, so $d = 4$. There is only one feasible way of buffer insertion for route l that will insert a buffer at the grid of distance four from the source, so $b\_\mathrm{count}(4, l)$ is equal to one and the probability that grid $(x_0 + 4, y_0)$ is required to have a buffer inserted can be computed as

$$\begin{aligned} b\_\mathrm{insert}(x_0 + 4, y_0, l, k) =& \frac{b\_\mathrm{count}(4, l)}{4} \\ =& 0.25. \end{aligned}$$

We can compute $b\_\mathrm{insert}(x, y, l, k)$ by calculating $b\_\mathrm{cnt}(d)$ where $b\_\mathrm{cnt}(d)$ is the number of feasible ways of buffer insertion between two grids of distance d apart such that buffers are inserted at both ends. The probability of requiring a buffer at a grid (x, y) of distance d from the source in route l can be calculated as

$$b\_\mathrm{insert}(x, y, l, k) = \frac{b\_\mathrm{cnt}(d) \times b\_\mathrm{cnt}(\mathrm{length} - d)}{b\_\mathrm{cnt}(\mathrm{length})}$$

where $length$ is the distance from the source S to the sink T of route l, $b\_\mathrm{cnt}(d)$ computes the number of feasible ways of buffer insertion from the source to grid (x, y), $b\_\mathrm{cnt}(\mathrm{length} - d)$ computes the number of feasible ways of buffer insertion from grid (x, y) to the sink and $b\_\mathrm{cnt}(\mathrm{length})$ computes the total

$$g_0(x, y, k).\mathrm{prob}[i] = \begin{cases} g_0(x-1, y, k).\mathrm{prob}[i-1] + g_0(x, y-1, k).\mathrm{prob}[i-1], & \text{for } 1 \le i \le up \\ \max_{\mathrm{low} \le j \le up}\{g_0(x, y, k).\mathrm{prob}[j] \times b\_\mathrm{success}(x, y)\}, & \text{for } i = 0 \end{cases} \quad (3)$$

$$g_1(x, y, k).\mathrm{probprob}[i] = \begin{cases} g_1(x+1, y, k).\mathrm{prob}[i-1] + g_1(x, y+1, k).\mathrm{prob}[i-1], & \text{for } 1 \le i \le up \\ \max_{\mathrm{low} \le j \le up}\{g_1(x, y, k).\mathrm{prob}[j] \times b\_\mathrm{success}(x, y)\}, & \text{for } i = 0 \end{cases} \quad (4)$$

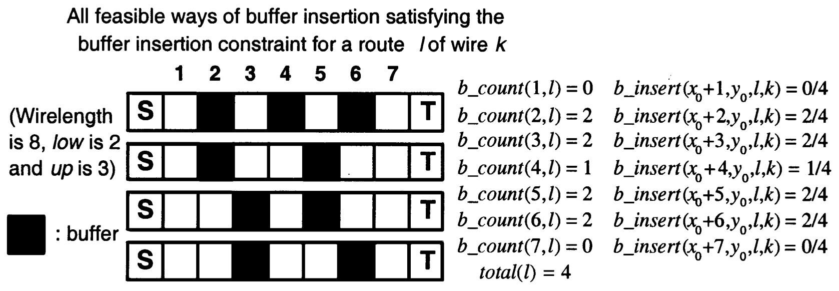

All feasible ways of buffer insertion satisfying the buffer insertion constraint for a route l of wire k
1 2 3 4 5 6 7
(Wirelength is 8, low is 2 and up is 3)
S
T
: buffer
b_count(1,l) = 0 b_insert(x0+1,y0,l,k) = 0/4
b_count(2,l) = 2 b_insert(x0+2,y0,l,k) = 2/4
b_count(3,l) = 2 b_insert(x0+3,y0,l,k) = 2/4
b_count(4,l) = 1 b_insert(x0+4,y0,l,k) = 1/4
b_count(5,l) = 2 b_insert(x0+5,y0,l,k) = 2/4
b_count(6,l) = 2 b_insert(x0+6,y0,l,k) = 2/4
b_count(7,l) = 0 b_insert(x0+7,y0,l,k) = 0/4
total(l) = 4

Fig. 6. Example of calculating the probability of buffer insertion at each grid.

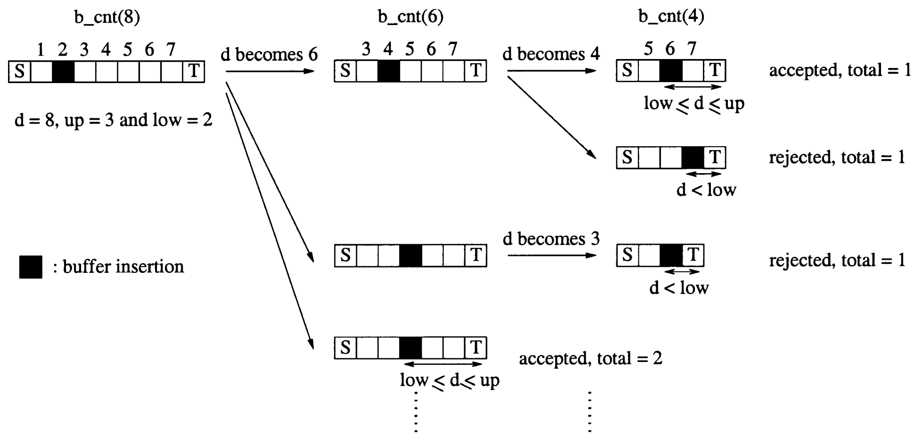


Fig. 7. Example of the forward recursive method.

number of feasible ways of buffer insertion from the source to the sink.

In order to compute the probability of buffer insertion $b\_\text{insert}(x, y, l, k)$ at each grid (x, y) for every route l and wire k efficiently, two methods based on dynamic programming are used. In both methods, $b_\text{cnt}(d)$s are calculated, saved and reused. They are the forward recursive method and the backward recursive method as described below.

1) Forward recursive method: The forward recursive method works by searching and counting the number of feasible ways of buffer insertion satisfying the buffer insertion constraint. For each successful way of buffer insertion found, the number accumulated will be increased by one. Finally, the total number of feasible ways of buffer insertion between two points of distance d apart will be obtained. In general, the function will be called recursive until all feasible ways of buffer insertion are found. An example is shown in Fig. 7.

For the example in Fig. 7, d is equal to eight and the upper and lower bounds of the buffer insertion constraint are three and two, respectively ($up = 3$ and $low = 2$). At the beginning, the function $b\_\text{cnt}(8)$ is called. We will then try to insert the first buffer at the grid $(x_0 + 2, y_0)$ which is at a distance low from the source S. The function $b\_\text{cnt}(6)$ is then called recursively. (Note that after the return of the call $b\_cnt(6)$, $b\_\text{cnt}(5)$ will be called because we can also insert the first buffer at the grid $(x_0 + 3, y_0)$ which is at a distance of three from the source S.) When $b\_\text{cnt}(6)$ is called, we will try to insert the second buffer at the grid $(x_0 + 4, y_0)$ which is at a distance *low* from the first buffer. The function $b\_\text{cnt}(4)$ is then called recursively again. This process will be repeated until $b\_\text{cnt}(2)$ is called and a "1" will be returned because the final value of d $(d = 2)$ is lying between *low* and *up*. In general, the function $b\_\text{cnt}(d)$ will be called recursively until all the feasible ways of buffer insertion are found. The formula is shown below

$$b\_\text{cnt}(d) = \begin{cases} \sum_{i=\text{low}}^{\text{up}} b\_\text{cnt}(d-i), & \text{if up} < d \\ 1 + \sum_{i=\text{low}}^{d-1} b\_\text{cnt}(d-i), & \text{if } low \le d \le up \\ 0, & \text{if } d < low. \end{cases}$$

By dynamic programming, i.e., calculating $b\_\text{cnt}(1), b\_\text{cnt}(2), \ldots, b\_\text{cnt}(d)$ iteratively, the time complexity to compute $b\_\text{cnt}(i)\ \forall\ 1 \le i \le d$ by this forward recursive method is $O(d \times up)$.

2) Backward recursive method: The backward recursive method is a faster method but it has a limitation that the lower bound in the buffer insertion constraint must be one. It counts the number of infeasible ways of buffer insertion by looking at the first position where a violation occurs. Finally, it computes the total number of infeasible ways of buffer insertion, f_total. Since the total number of ways of buffer insertion is 2^{d-1}, the total number of successful buffer insertion satisfying the buffer insertion constraint s_total, can be calculated as $2^{d-1} - f\_\text{total}$. An example is shown in Fig. 8.

For the example in Fig. 8, d is equal to eight and the upper and lower bounds of the buffer insertion constraint are three and one, respectively ($up = 3$ and $low = 1$). At the beginning, $b\_\text{cnt}(8)$ is called. We will first count the number of ways of buffer insertion such that the first violation occurs at the leftmost possible grid. This occurs when the first buffer is inserted after

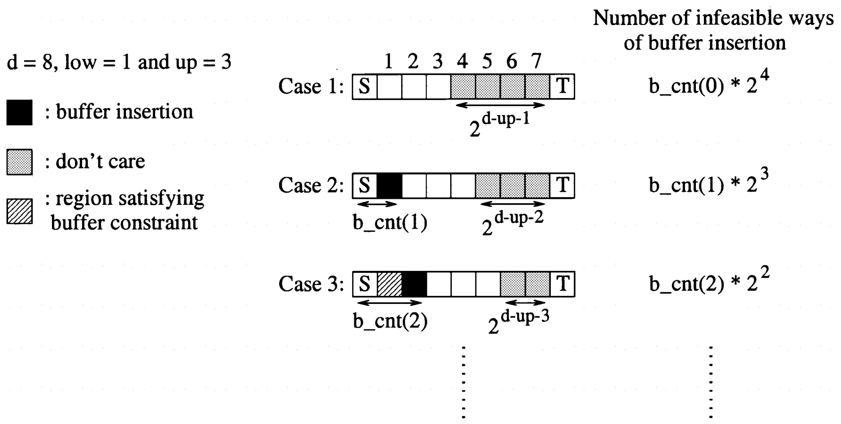


Fig. 8. Example of the backward recursive method.

the grid (x_0+3, y_0) because the interval between the source S and the first buffer is greater than up and the buffer insertion constraint is violated. All of the possible ways of inserting the remaining buffers are infeasible. As a result, the total number of infeasible ways of buffer insertion in this case is equal to 2^{d-up-1}, i.e., 2^4. If the first violation occurs at the grid (x_0+4, y_0), the total number of infeasible ways of buffer insertion is equal to $b\_\mathrm{cnt}(1)\times 2^{d-up-2}$, i.e., $b\_\mathrm{cnt}(1)\times 2^3$. Notice that since the first violation occurs at the grid (x_0+4, y_0), those insertions in front must be correct and we have made use of the recursive call $b\_\mathrm{cnt}(1)$ to count those feasible insertions in front. In this way, the function will be called recursively to obtain $b\_\mathrm{cnt}(d)$. The general formula is shown as

$$b\_\mathrm{cnt}(d)=\begin{cases}1, & \text{if } d=0\\ 2^{d-1}-\sum_{i=0}^{d-\mathrm{up}-1} 2^{d-i-\mathrm{up}-1}\times b\_\mathrm{cnt}(i), & \text{if } d>0.\end{cases}$$

By dynamic programming, i.e., calculating $b\_\mathrm{cnt}(1), b\_\mathrm{cnt}(2), \ldots, b\_\mathrm{cnt}(d)$ iteratively, the time complexity to compute $b\_\mathrm{cnt}(i)\ \forall\ 1\le i\le d$ by this backward recursive method is $O(up+d^2)$.

Assuming that the time required for each recursive call is similar for the two methods, we find that the backward recursive method is faster than the forward one when $up > d/5$. As a result, we can combine the two methods to find the total number of feasible ways of buffer insertion satisfying the buffer insertion constraint efficiently. The combined method is shown as follows:

```
Combined method: $b\_\mathrm{cnt}(d)$
Input: Distance between two target grids, $d$
Output: $b\_\mathrm{cnt}(d)$
If $(up > d/5)$ and $(low = 1)$
 Backward recursive method
else
 Forward recursive method
Return.
```

According to the combined method, given the total length (length) of a route l of a wire k running from a source S to a sink T and the distance (d) from the source to a grid at (x, y), we can compute the total number of feasible ways of buffer insertion from S to T such that a buffer is inserted at (x, y) as

$$b\_\mathrm{cnt}(d)\times b\_\mathrm{cnt}(\mathrm{length}-d).$$

Therefore, the probability that a buffer is required at grid (x, y) for route l of wire k is

$$b\_\mathrm{insert}(x,y,l,k)=\frac{b\_\mathrm{cnt}(d) * b\_\mathrm{cnt}(\mathrm{length}-d)}{b\_\mathrm{cnt}(\mathrm{length})}.$$

Note that these values can be computed, saved, and reused for wires of the same length.

B. Estimation of Buffer Resources

In order to obtain the congestion information, we need to compute the probability of buffer insertion at each grid. We have shown in the above section the computation of the probability that a route l of wire k will require a buffer at a grid (x, y), i.e., $b\_\mathrm{insert}(x,y,l,k)$. The total number of buffer requirements at grid (x, y), $b\_\mathrm{usage}(x, y)$, can then be calculated as

$$b\_\mathrm{usage}(x,y)=\sum_{\text{all wire } k \text{ and all route } l} b\_\mathrm{insert}(x,y,l,k).$$

We use dynamic programming to calculate b_usage. Since we need to compute $b\_\mathrm{insert}(\cdot)$ at each grid from the source to the destination (which has at most K^2 grids) for each two-pin net, the time complexity is $O(m\times K^2)$. The probability of buffer insertion at grid (x, y), $b\_\mathrm{success}(x, y)$, can then be calculated as

$$b\_\mathrm{success}(x,y)=\min\left\{1, \frac{b\_\mathrm{space}(x,y)}{b\_\mathrm{usage}(x,y)}\right\}$$

where $b\_\mathrm{space}(x, y)$ is the maximum number of buffers that can be inserted at grid (x, y) and is dependent on the amount of empty space in grid (x, y). From the equation, we can see that if the amount of empty space at grid (x, y) is large enough to accommodate all the possible buffer insertions, the value of $b\_\mathrm{success}(x, y)$ is equal to one. Otherwise, it is equal to $\big(b\_\mathrm{space}(x,y)/b\_\mathrm{usage}(x,y)\big)$.

In order to ensure that every wire contributes equally to buffer usage, we can normalize the probabilities such that the contribution to $\sum_{x,y} b\_\mathrm{usage}(x,y)$ from any wire k is always equal to one, even if multiple routes are available for wire k.

V. Two-Phase Simulated Annealing

In our floorplanner, we will use sequence pair to represent the packings and apply simulated annealing for optimization. Simulated annealing is an iterative and nondeterministic optimization technique. In simulated annealing, all the moves that result in a decrease in cost will be accepted. The moves that result in an increase in cost are accepted with a probability that decreases over the iterations. We have used the same set of moves as in [10]. The algorithm is shown as

```
Simulated Annealing
$temp = init\_temperture$
$place = init\_placement$
$iter = 0$
while $temp > final\_temperature$ do
 while $iter < max\_iter\_no$ do
  $new\_place = \mathrm{PERTURB}(place)$
  $\triangle C = \mathrm{COST}(new\_place) - \mathrm{COST}(place)$
  if $(\triangle C < 0)$ or $\mathrm{RANDOM}(0,1) > e^{-k\triangle C/temp}$
   $place = new\_place$
  $iter = iter + 1$
  $temp = \mathrm{SCHEDULE}(temp)$.
```

In our design, the simulated annealing process is divided into two phases. They are the area optimization phase and the congestion optimization phase. In the area optimization phase, the congestion information is less meaningful because the locations of the modules are still far from their final positions. Thus, the cost function used in this phase does not include the congestion cost nor the buffer insertion cost. The cost function is

$$\mathrm{Cost}_0 = \mathrm{Area} + \alpha \times \mathrm{Wire}$$

where Area is the area of the floorplan and Wire is the total wirelength (its computation will be discussed in Section VI) and α is the weight.

In the congestion optimization phase, the cost function is

$$\mathrm{Cost}_1 = \mathrm{Area} + \alpha \times \mathrm{Wire} + \beta \times M\_\mathrm{weight}$$

where Area is the area of the floorplan, Wire is the total wirelength, M_weight is the average number of wires in the top 4% most congested grids, and α and β are the weights.

In the transitional period from the area optimization phase to the congestion optimization phase, we need to recalculate the temperature because the order of magnitude of the costs might change a lot and the acceptance rate could drop or rise unexpectedly, if we did not adjust the temperature. As a result, we need to use a new temperature to maintain the acceptance rate. To achieve this, we will first obtain the mean value of $\triangle\mathrm{Cost}$ from a number of random walks using the old cost function, and obtain the mean value of $\triangle\mathrm{new\_Cost}$ from a number of random walks using the new cost function. The new temperature $\mathrm{new\_}T$ can then be computed as

$$\mathrm{new\_}T = \frac{\triangle\mathrm{new\_Cost}}{\triangle\mathrm{Cost}} \times \mathrm{old\_}T.$$

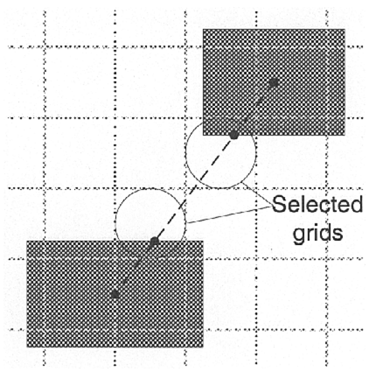


Fig. 9. Intersection-to-intersection estimation.

By using $\mathrm{new\_}T$, the acceptance rate can be maintained and the change in cost function can be performed smoothly.

VI. Wirelength Estimation

In the floorplanning stage, the positions of the input/output (I/O) pins in the modules are not yet fixed. Since a module will usually cover a number of grids, the assumption on the locations of the I/O pins will affect the results significantly. We need to reasonably estimate the positions of the I/O pins in order to accurately compute the interconnect and congestion costs.

In traditional floorplanners, half-perimeter estimation using the center of a module as the location of that module is most commonly used. However, if we consider congestion and buffer insertion, center-to-center estimation is difficult to be applied. It is because, if a module M is large, center-to-center estimation will increase the congestion estimation at the center of M significantly. Congestion may be overestimated and this will reduce the accuracy of the whole method. In addition, buffer may be required to be inserted at the grids covered by M. Therefore, many routes will become impossible as buffers cannot be inserted on macro blocks. As a result, we adopt a more appropriate method for estimating the positions of the I/O pins on a module that can be used in our routability-driven floorplanner.

In order to distribute the I/O pins into the grids appropriately, intersection-to-intersection method is used. Consider a net connecting two modules A and B, we will first draw a line from the center of one module to another. The two intersecting points will be found on the edges of the modules and the I/O pins will be placed at the grids containing the intersecting points as in Fig. 9.

This is an appropriate method for our floorplanner since the estimated positions of the I/O pins will be similar to those in the final design and the buffer locations can be estimated more accurately.

VII. Handling Multipin Nets

In order to handle multipin nets, we need to decompose a multipin net into a set of two-pin nets. There are several methods to decompose a multipin net into two-pin nets such as using the minimum spanning tree (MST) method, or the rectilinear steiner tree (RST) method. MST runs faster but it may overestimate the congestion because of the overlapping net segments. However, this conservative estimation will not affect the resultant packing significantly because the total length of an MST can be reduced at most by 6%–9% by removing all the overlapping net segments to obtain a corresponding RST [7]. Since the runtime of an RST algorithm is usually much slower than that of an MST algorithm, MST is a better choice for estimation purpose in the early floorplanning stage. As a result, we apply MST to handle multipin nets in our floorplanner.

VIII. Simple Global Router for Evaluation

We have implemented a simple global router to evaluate the performance of the floorplanner. Given a floorplan input, the router will divide the floorplan into a grid structure. Multipin nets are decomposed into two-pin nets by using the MST method and the two-pin nets will be routed one after another by dynamic programming based on the following criteria in decreasing order of priority:

- a route passing through less-congested grids;
- a route requiring less buffers to be inserted;
- a route passing through grids with more empty space left such that buffers can still be inserted for other wires.

In the dynamic programming, when a wire w is required to be routed from the grid (x_0, y_0) to the grid (x_t, y_t) where $x_t > x_0$ and $y_t > y_0$, w must pass through either the grid $(x_t - 1, y_t)$ or $(x_t, y_t - 1)$ if shortest Manhattan distance is used. If the route from (x_0, y_0) to $(x_t - 1, y_t)$ is better than the route from (x_0, y_0) to $(x_t, y_t - 1)$ based on our routing criteria above, w will pass through $(x_t - 1, y_t)$ rather than $(x_t, y_t - 1)$. We can, thus, choose the better one between these two to form part of the best route from (x_0, y_0) to (x_t, y_t). In this way, we can find the best route from the source to each grid dynamically row by row by comparing the routes coming from the grid above and the route coming from the grid on the left.

However, there are limitations on the number of wires in each grid (*wiring capacity*). If a wire can be routed from its source to its sink in its shortest Manhattan distance without violating the buffer insertion constraint and the wiring capacity constraint, the wire is said to be *routable*, otherwise, it is called an *unroutable* wire. For each routable net, we will try to minimize the maximum congestion along its route, minimize the number of buffers used, and maximize the amount of remaining buffer resources. After a net is routed, the information on congestion and the remaining buffer resources in each grid will be updated accordingly.

IX. Experimental Results

We have implemented two floorplanners, a traditional floorplanner $F1$ based on simulated annealing without considering congestion or buffer planning and a routability-driven floorplanner $F2$ based on the two-phase simulated annealing method using the probabilistic model discussed in the above sections for congestion estimation and buffer planning. The floorplan output is evaluated by the simple global router described in Section VIII. The two floorplanners are compared with respect to the total area, wirelength, congestion and the number of unroutable wires. Congestion is measured as the average number of wires passing through the top 4% most congested grids, and an unroutable wire is a wire that cannot be routed in its shortest Manhattan distance without violating the buffer insertion constraint and the wiring capacity constraint of each grid. The data sets used are $ami33$, $ami49$, and $playout$. The circuit characteristics and parameters used in the three data sets are shown in Table I. Each result shown in Tables II and III is an average value obtained by performing the experiment eight times. The value of low and up are estimated analytically as follows:

TABLE I
Circuit Characteristics and Parameters Used in the Experiments

	$ami33$	$ami49$	$playout$
No. of modules	33	49	62
(No. of nets, no. of two-pin nets)	(123, 265)	(408, 504)	(1611, 1946)
g_unit (μm)	600	400	500
Wiring capacity	6	18	60
$[low, up]$	[3,6]	[4,8]	[4,7]

$$\text{low} = \frac{\sqrt{\frac{4*(R_b*C_b+D_b)}{R_o*C_o}}}{(2*g\_\text{unit})}$$

$$\text{up} = \frac{\sqrt{\frac{4*(R_b*C_b+D_b)}{R_o*C_o}}}{(g\_\text{unit})}$$

where $g_$unit is the length of a grid, R_b is the buffer resistance, C_b is the buffer capacitance, D_b is the intrinsic delay of a buffer, R_o is the unit wire resistance, and C_o is the unit wire capacitance. The above equations are derived from the Elmore delay model [6] by assuming that $R_D = R_B$ and $C_L = C_B$. The derivation will be shown in Appendix I. Notice that the dimensions of the modules are enlarged by a factor of ten for $ami33$ and $playout$ in order to demonstrate the effect of buffer block planning. In addition, the wiring capacities are chosen such that unroutable wires will appear in our global routing test of those data sets.

Table II shows the results of the experiments. We can see that $F1$ performs better than $F2$ in area optimization, but the differences are very small. On the other hand, $F2$ can reduce the total wirelength, congestion and the number of unroutable wires significantly. This demonstrates the effectiveness of our probabilistic congestion model in reducing interconnect cost in floorplan design.

Table III compares the runtime of our routability-driven floorplanner based on the traditional single-phase and two-phase simulated annealing approach. We can see that the runtime can be shortened by 1.5–5 times for these three data sets. This demonstrates the effectiveness of the two-phase approach and we can expect this approach to be useful for solving floorplanning problems of even larger scale. In general, the runtime can be further reduced if we start the second stage of the annealing process later.

X. Conclusion

In this paper, we presented a new method to compute the congestion information in floorplanning taking into account buffer planning with variable interval buffer insertion constraint. The estimations are based on the supply and demand analysis of the

TABLE II
COMPARISONS BETWEEN $F1$ AND $F2$

	*ami*33		*ami*49		*playout*	
Floorplanners	$F1$	$F2$	$F1$	$F2$	$F1$	$F2$
[1]Area ($10^3\mu m^2$)	129344.00	129661.00	39454.41	39754.35	99058.00	99254.00
Wirelength ($10^3\mu m$)	206.40	205.90	399.75	379.80	3067.60	2755.50
[2] Congestion	2.27	2.21	0.13	0.12	25.60	24.44
Unroutable 2-pin wires	14.75	9.63	13.13	9.88	163.88	115.88
[3] Runtime (s)	126.69	1333.90	152.63	3573.79	497.89	11680.80

[1]Dimensions of the mudules in *ami*33 and *playout* are multiplied by ten. [2]Average number of nets per $10^3\mu$m^2 for the top 4% most congested grids. [3]Experiments are performed using Pentium IV 1.2 GHz with 512 MB of memory.

TABLE III
COMPARING THE RUNTIME OF THE SINGLE-PHASE AND TWO-PHASE FLOORPLANNER

	Single-phase Floorplanner	Two-phase Floorplanner
*ami*33	2124.00s	1333.90s
*ami*49	9805.00s	3573.79s
playout	54541.00s	11680.80s

routing and buffer resources, where the supply is determined by the floorplan solution (the amount of empty space) and the demand is determined by the interconnect structure. Computations of the congestion information in our model are quite complicated but they can be performed efficiently by dynamic programming. Experimental results show that our floorplanner can reduce the interconnect cost efficiently without having much penalty in area. In addition, the runtime can be improved significantly without degradation in performance by the two-phase simulated annealing approach.

APPENDIX

According to the Elmore delay model [6], we can compute the delay when no buffer is inserted (D_{nb}) and the delay when a buffer is inserted at the middle of the wire (D_{wb}) by the following equations:

$$\begin{aligned}D_{nb} =& R_D\left(c_0lh + c_fl + C_L\right)\\ &+ \frac{r_0l}{h}\left(\frac{c_0lh}{2}+\frac{c_fl}{2}+C_L\right)\\ D_{wb} =& R_D\left(\frac{c_0lh}{2}+\frac{c_fl}{2}+C_B\right)\\ &+\frac{r_0l}{2h}\left(\frac{c_0lh}{4}+\frac{c_fl}{4}+C_B\right)+D_B\\ &+R_B\left(\frac{c_0lh}{2}+\frac{c_fl}{2}+C_L\right)\\ &+\frac{r_0l}{2h}\left(\frac{c_0lh}{4}+\frac{c_fl}{4}+C_L\right)\end{aligned}$$

where R_D is the driver resistance, C_L is the load capacitance, R_B is the buffer resistance, C_B is the buffer capacitance, D_B is the buffer delay, h is the wire width, l is the wirelength, and r_0, c_0, and c_f are unit wire resistance, unit wire capacitance, and unit fringing capacitance, respectively.

In order to compute the upper bound up, we assume that $R_D = R_B$ and $C_L = C_B$. Buffer is needed when $D_{nb} > D_{wb}$. By rewriting the above equations, we have

$$\begin{aligned}&D_{nb} - D_{wb} > 0\\ &R_D(c_0lh + c_fl + C_L)\\ &\quad+\frac{r_0l}{h}\left(\frac{c_0lh}{2}+\frac{c_fl}{2}+C_L\right)\\ &\quad-\left(R_D\left(\frac{c_0lh}{2}+\frac{c_fl}{2}+C_B\right)\right.\\ &\qquad+\frac{r_0l}{2h}\left(\frac{c_0lh}{4}+\frac{c_fl}{4}+C_B\right)+D_B\\ &\qquad+R_B\left(\frac{c_0lh}{2}+\frac{c_fl}{2}+C_L\right)\\ &\qquad\left.+\frac{r_0l}{2h}\left(\frac{c_0lh}{4}+\frac{c_fl}{4}+C_L\right)\right)>0\\ &\frac{r_0c_0l^2}{4} - R_BC_B - D_B > 0\\ &l > \sqrt{\frac{4*(R_B*C_B+D_B)}{R_o*C_o}}.\end{aligned}$$

Since a buffer will be inserted if the wire segment is longer than up, we can put

$$up = \frac{\sqrt{\frac{4*(R_B*C_B+D_B)}{R_o*C_o}}}{g\_\mathrm{unit}}.$$

We compute low as half of up

$$\mathrm{low} = \frac{\sqrt{\frac{4*(R_B*C_B+D_B)}{R_o*C_o}}}{2\times g\_\mathrm{unit}}.$$

REFERENCES

[1] C. J. Alpert, J. Hu, S. S. Sapatnekar, and P. G. Villarrubia, "A practical methodology for early buffer and wire resource allocation," in *Proc. 38th ACM/IEEE Design Automation Conf.*, 2001, pp. 189–194.

[2] H. M. Chen, H. Zhou, F. Y. Young, D. Wong, H. H. Yang, and N. Sherwani, "Integrated floorplanning and interconnect planning," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1999, pp. 354–357.

[3] J. Cong, "Challenges and opportunities for design innovations in nanometer technologies," in Frontiers in Semiconductor Research: A Collection of SRC Working Papers, 1997.

[4] J. Cong, T. Kong, and D. Z. Pan, "Buffer block planning for interconnect driven floorplanning," in *Dig. Tech. Papers, IEEE Int. Conf. Computer-Aided Design*, 1999, pp. 358–363.

[5] F. E. Dragan, A. B. Kahng, S. Muddu, and A. Zelikovsky, "Provably good global buffering using an available buffer block plan," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 2000, pp. 104–109.

[6] W. C. Elmore, "The transient response of damped linear network with particular regard to wide band amplifiers," *J. Appl. Physics*, vol. 19, pp. 55–63, 1948.

[7] J. M. Ho, G. Vijayan, and C. K. Wong, "A new approach to the rectilinear steiner tree problem," in *Proc. 26th ACM/IEEE Design Automation Conf.*, 1989, pp. 161–166.

[8] A. B. Kahng, S. Muddu, E. Sarto, and R. Sharma, "Interconect tuning strategies for high performance ICS," in *Proc. Desig, Automation, Test Eur. Conf. Exhib.*, 1998, pp. 471–478.

[9] J. Lou, S. Krishnamoorthy, and H. S. Sheng, "Estimating routing congestion using probabilistic analysis," in *Proc. Int. Symp. Physical Design*, 2001, pp. 112–117.

[10] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-packing- based module placement," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1995, pp. 472–479.

[11] P. Sarkar, V. Sundararaman, and C. K. Koh, "Routability-driven repeater block planning for interconnect-centric floorplanning," in *Proc. Int. Symp. Physical Design*, 2000, pp. 186–191.

[12] X. P. Tang and D. E. Wong, "Planning buffer locations by network flows," in *Proc. Int. Symp. Physical Design*, 2000, pp. 186–191.

[13] M. Wang and M. Sarrafzadeh, "Modeling and minimization of routing congestion," in *Proc. ASP-ACM/IEEE Design Automation Conf.*, 2000, pp. 185–190.

Chiu-Wing Sham received the B.Eng.(Hons.) and M.Phil. degrees in computer engineering, in 2000 and 2002, respectively, from the Chinese University of Hong Kong, Shatin, N.T., Hong Kong, where he is currently pursuing the Ph.D. degree in computer engineering.

His research interests include interconnect-driven floorplanning, floorplan representations, and flexible objects placement problems.

Evangeline F. Y. Young received the B.Sc. and M.Phil. degrees in computer science from the Chinese University of Hong Kong, Shatin, N.T., Hong Kong, in 1991 and 1993, respectively, and the Ph.D. degree from the University of Texas, Austin, in 1999.

Currently, she is an Assistant Professor with the Department of Computer Science and Engineering, Chinese University of Hong Kong. Her research interests include algorithms and computer-aided design of very large scale intregation circuits. She is now working actively on floorplan design optimization, circuit partitioning, circuit retiming, and packing representation.