Normalized Netlengths: A Measure of Routing Cost for Logic Synthesis

Hirendu Vaishnav and Massoud Pedram

CENG Technical Report 95-17

Department of Electrical Engineering - Systems
University of Southern
Los Angeles, California 90089-2562
(213)740-4458

June 9, 1995

Normalized Netlengths: A Measure of Routing Cost for Logic Synthesis

Hirendu Vaishnav, Massoud Pedram

University of Southern California

Department of Electrical Engineering - Systems

Los Angeles, CA 90089-2562

June 9, 1995

Abstract

In this report, we present a cost function which can be used to minimize the routing in a circuit during technology independent phase of logic synthesis. Instead of estimating the absolute routing cost of a net, this function captures the relative routing costs of nets based on the number of terminals on the nets. Unlike the routing cost functions proposed earlier, our cost function does not require layout-parameters or any tuning of the variables to achieve acceptable estimation of routing cost. We illustrate the usefulness of this function by applying it to the process of extraction in logic synthesis. Our experimental results indicate that minimization of this pin-count based routing measure during extraction improved routing by 10% and chip area by 8% on average at no performance loss.

1 Introduction

With the drive towards deep-submicron devices, the feature sizes have shrunk considerably. The main impact of deep-submicron technology is that the relative contribution of interconnect in determining the circuit area, delay and power dissipation will increase substantially. Even with the current technology, interconnect is the dominating contributer to chip area, delay and power dissipation.

Conventional synthesis techniques attempt to minimize some intermediate measure of chip quality (e.g., total gate area or number of literals for estimating chip area; gate delay for estimating chip delay; power dissipated due to gate capacitances for estimating power) while ignoring the interconnect. Increasing contribution of interconnect to the final chip area, delay and power dissipation, and the inability of existing synthesis techniques to effectively optimize interconnect lead to circuits that do not satisfy constraints after physical design. The only remedy to this is to iterate between logic synthesis and physical design until all constraints are met. However, with the move towards deep-submicron feature sizes, the number of required iterations increase dramatically while the computational cost of each iteration rises rapidly. It is our belief that the solution to the challenge posed by deep-submicron technology is to improve the interaction between synthesis and layout tools and to shift the focus of synthesis tools from traditional "gate optimization" to "interconnect optimization".

The structural and functional freedom available at higher levels of design abstraction can be exploited to greatly impact the interconnect cost of the circuit. However, it is very difficult to accurately estimate the interconnect cost at these higher levels. In this report, we propose a simple routing cost function which accurately characterizes *relative* netlengths of nets based on their pin-count and which can be calculated efficiently and minimized effectively during logic synthesis.

Previous work on netlength estimation can be divided in three categories based on the analysis model: empirical, theoretical and procedural. Empirical models derive expressions for physical characteristics (including expected values of average/total netlengths) by extracting information from actual designs and fitting curves to the data [5]. Most of these approaches are based on *Rent's rule* which is a relationship between the IO and the cell count of a design, i.e., $ioCount = averageCellSize \times cellCount^r$, where r is *Rent's exponent*. Theoretical models produce closed form equations for netlength estimation by making simplifying assumptions about the interconnect structure. Theoretical models are either deterministic which rely directly on the parameters extracted from actual design [4] or are based on stochastic analysis that require fitting curves from actual design data [17, 7]. Procedural models consider more detailed aspect of the placement process and the interconnection structure to improve the accuracy of the predictions [19, 12]. However, all of these works either rely on parameters extracted from actual design, or require fitting curves from actual design data, or require the a priori knowledge of chip

dimensions limiting their applicability for logic synthesis.

Our approach is a combination of procedural and theoretical approaches and characterizes expected netlength variation with respect to the pin-count of the nets. In contrast to above approaches which attempt to estimate absolute netlengths, our main emphasis is to estimate the relative length of pins with different pin-counts. Instead of ignoring routing tree topology (as in [19]) or developing a procedural netlength estimator (as in [12]), we use the worst case routing tree topology for an optimal rectilinear steiner tree [3] to calculate the netlength within a given bounding-box. Also, instead of enumerating pin assignments as in [19] and [12], we use a probabilistic method to estimate the expected length of the box enclosing terminals of a net, thereby simplifying the corresponding formula. The formula is further simplified by making it independent of chip dimensions while keeping the estimation error very low. Indeed, the formula presented here requires only the number of pins on a net to estimate the relative routing contribution of the nets, making it an ideal candidate for a netlength minimizing objective function during logic synthesis. Apart from verifying the accuracy of our relative netlength estimation with the actual netlengths of placed and routed circuits, we use this cost function during logic extraction to minimize the circuit routing.

Logic extraction is the process of identifying subexpressions common to two or more boolean functions, which are then extracted as intermediate nodes in a multi level circuit. The goal of extraction is to minimize the chip area by sharing logic across the network. Literal count has been traditionally used as the objective function during extraction as it correlates well with the gate area. However, minimizing the gate area does not always reduce the chip area. Extraction mechanisms based on literal minimization often produce circuits with large number of fanouts per node. Apart from increasing the routing overhead, this high number of fanouts has other undesirable side-effects, e.g., degradation of performance [20], reducing the effectiveness of retiming and resynthesis mechanisms [10], etc.. Indeed, increased routing overhead may undo any area savings due to literal minimization. Hence, it is imperative that routing cost be considered during extraction.

Recently, a number of researchers have addressed routing optimization during logic synthesis. Saucier et al. [18] proposed a novel mechanism called *lexicographical extraction* to reduce the routing overhead of the circuit by deriving and maintaining an order amongst primary inputs of the circuit during extraction. Pedram et al. [11] proposed a mechanism where an incrementally updated companion placement is used to estimate the routing area and delay during logic synthesis. Vaishnav et al. [13, 21] proposed a mechanism to minimize the routing overhead by avoiding wire crossings during technology decomposition and fanout optimization. However, the problem of identifying accurate routing cost functions that can be used during earlier phases of logic synthesis has been not been fully addressed. In this report, a routing cost function based on the pin-count of a net and the corresponding routing-driven extraction procedures are proposed.

The rest of the report is organized as follows. In Section 2, we present the derivation of our routing measure. In Section 3, we present our approach to minimize the pin-count based objective function during extraction. Section 4 presents our experimental results and discussion. Concluding remarks are presented in Section 5.

2 A Pin-Count Based Routing Cost Function

Many circuit parameters contribute to the routing cost. These parameters can be classified in two categories: parameters which are dependent on the local structure of the boolean network, and those which are dependent on the global structure of the boolean network. Local parameters characterize the routing cost of a node or a net in the circuit based on the local connections and the local subnetwork structure while global parameters characterize the routing cost of the boolean network as a whole. For example, number of pins on the divisor is a local parameter affecting routing cost. (Clearly, a multi-pin net requires larger area to route compared to a two-pin net. It also complicates the subsequent place/route procedures as the resulting net list will contain more global nets.) On the other hand, distribution of nets across the circuit is a global parameter affecting routing. A balanced distribution of nets across the network tends to improve routing by reducing the routing congestion in the network. Two such routing cost functions (based on minimizing the fanout range of the signals or the overlap among fanout ranges of signals) during logic extraction are proposed in [22]. This cost function is more efficient to calculate during extraction yet produces results that are comparable to those of the more elaborate functions proposed in [22]. In this report, a local function characterizing the routing cost based on the pin-count of the nets is presented.

2.1 Effect of Pin-count on Routing Length

To determine the effect of pin-count on the routing length of a net, we performed a simple experiment. Three benchmark examples were area-optimized using SIS, placed using GORDIAN [6], and routed using TimberWolf global router [9] and YACR2 [15] channel router. The graph depicting the average netlength as a function of the number of pins on the net is shown in Figure 1. The figure confirms that pin-count of a net is directly related to the netlength. However, to effectively minimize the total netlength of a circuit during logic synthesis using pin-counts, the netlengths must be characterized in terms of the pin-counts. We propose such a characterization. It should be noted that instead of estimating absolute netlengths, our emphasis is on estimating relative netlengths to allow a comparative selection during early stages of logic synthesis.

Definition 2.1 Equi-perimeter Netlength ratio *of a net with n pins (denoted by $\rho(n)$) is obtained by dividing the netlength of a net with n pins by the netlength of a net*

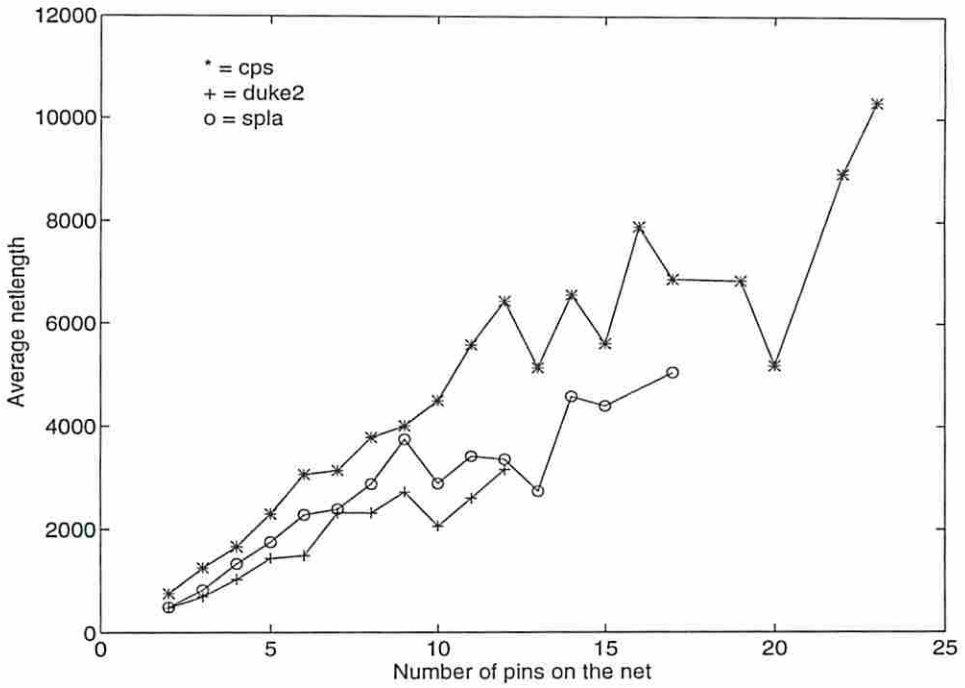
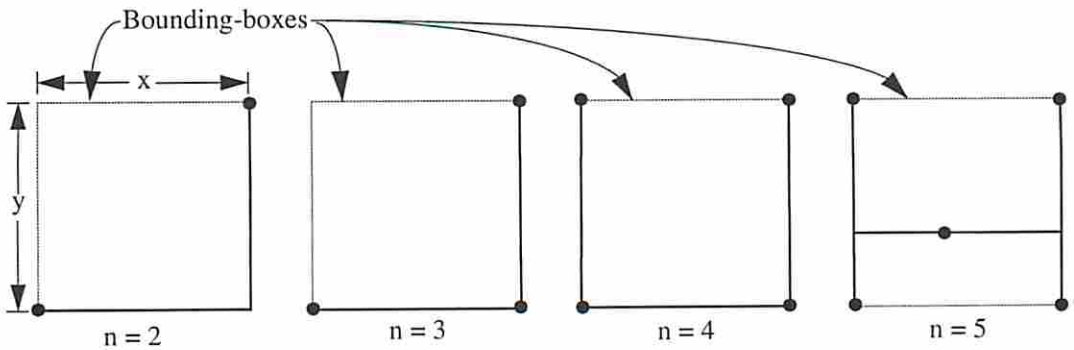


Figure 1: Average netlength for different pin counts for three benchmark circuits.



Pins are denoted by thick circles; Thick lines correspond to the routing trees
When $x = y$: $\rho(2) = \rho(3) = (x+y)/(x+y) = 1$; $\rho(4) = \rho(5) = (x+2y)/(x+y) = 1.5$

Figure 2: An illustration of equi-perimeter netlength ratios. The worst value of ρ occurs when $x = y$.

with 2 pins given that the two nets establish the same bounding-box.

For example, in the worst case, the optimal values of $\rho(2) = \rho(3) = 1$ whereas $\rho(4) = 1.5$ as shown in Figure 2.

Definition 2.2 Perimeter ratio *of a net with n pins (denoted by $\nu(n)$) is obtained by dividing the perimeter of a net with n pins by the perimeter of a net with 2 pins.*

Based on the above, if $perimeter(2)$ denotes the perimeter of a 2-pin net, the netlength $L(n)$ of an n-pin net is given by:

$$L(n) = \frac{perimeter(2)}{2} \cdot \rho(n) \cdot \nu(n)$$

Since $\frac{perimeter(2)}{2}$ is a constant, the relative routing contribution of two nets with pin-count n and n' can be obtained by comparing $\rho(n) \cdot \nu(n)$ with $\rho(n') \cdot \nu(n')$. We refer to this measure of relative routing contribution as normalized netlength of net n.

Definition 2.3 Normalized Netlength *of a net with n pins (denoted by $\eta(n)$) is defined as:*

$$\eta(n) = \rho(n) \cdot \nu(n) \tag{1}$$

In our work, we use the expected values of $\eta(n)$ for different pin-counts n, to drive our routing-driven extraction procedure. To calculate expected value of $\eta(n)$, we need to calculate expected values of $\rho(n)$ and $\nu(n)$.

The placement model adopted in this report assumes that the chip has width W and Height H in terms of horizontal and vertical pin-pitches, i.e., $W = actual\_chip\_width/pin\_pitch_x$ and $H = actual\_chip\_height/pin\_pitch_y$ where pin_pitch_x is the minimum possible center-to-center distance between two adjacent pins in the X-dimension and pin_pitch_y is the minimum possible center-to-center distance between two adjacent pins in the Y-dimension. This results in a chip layout which has grid-lines determined by pin_pitch_x and pin_pitch_y. Further, we assume that each pin has to conform to these grid boundaries, i.e., each pin has to be completely confined within these grid boundaries. We also assume that pins are randomly distributed on the chip, independent of other pins. This implies that we allow more than one pins to share one pin location. However, since we are estimating the bounding-box width and bounding-box height separately, this assumption is necessary because x-projections of more than one pins may coincide even tough they have distinct locations due to different y locations.

2.2 Estimating Equi-perimeter Netlength Ratios

When the net is routed using a rectilinear steiner tree, Chung and Hwang [3] have exhaustively calculated worst case optimal value of $\rho(n)$ $n \leq 10$, i.e., maximum value of

optimal rectilinear steiner tree netlength under all possible pin configurations within a unit size bounding-box. We reproduce these values of $\rho(n)$ in table 1. They have also shown that

$$lim_{n\to\infty}\rho(n) \quad = \quad \frac{\sqrt{n}+1}{2} \tag{2}$$

n	2	3	4	5	6	7	8	9	10
$\rho(n)$	1	1	3/2	3/2	5/3	7/4	11/6	2	2

Table 1: Worst case equi-perimeter netlengths.

The above implies that when the nets are routed using optimal rectilinear steiner trees, the equi-perimeter netlength ratio is $O(\sqrt{n})$. Next, we justify our choice of this estimation of $\rho(n)$ which gives the worst case equi-perimeter netlength ratio when nets are routed using optimal rectilinear steiner trees.

Ideally, equi-perimeter netlength ratios should be estimated by enumerating all pin configuration in a given bounding-box and averaging the netlengths obtained by routing each such pin configuration with an optimal rectilinear steiner tree. Apart from being computationally expensive, average netlengths calculated by such an analysis will depend on the bounding-box size. The consequence of assuming the worst case pin configuration to calculate $\rho(n)$ is that we might obtain pessimistic estimates. However, the pessimistic values of $\rho(n)$ may be negated to some extent due to our assumption of *optimal steiner* routing trees for each net which gives optimistic estimates since existing routers only approximate optimal steiner trees.

2.3 Estimating Perimeter Ratios

To estimate perimeter ratios, we need to address the following question. How does the net bounding box increase with an increase in pin-count?

Suppose we have a net with n pins. Let $P_{i,i+x}(n)$ denote the probability of all n pins falling within a X-span of $[i, i+x]$. Let $P_{|i,i+x|}(n)$ denote the probability of n pins having a X-span of $[i, i+x]$. Given the chip width W, the probability that the net bounding-box has width x, denoted by $X_x(n)$, is:

$$X_x(n) = \sum_{i=1}^{W-x} P_{|i,i+x|}(n)$$

Based on this, the expected value of the width $W(n)$ of net bounding-box is given by:

$$W(n) = \sum_{x=0}^{x=W-1} xX_x(n) = \sum_{x=0}^{x=W-1} x \sum_{i=1}^{W-x} P_{|i,i+x|}(n) \tag{3}$$

Next, we calculate $P_{|i,i+x|}(n)$. This is given by: probability that all n pins fall within $[i, i+x]$ minus the probability that all pins fall within$[i, i+x-1]$ or within $[i+1, i+x]$, i.e.,:

$$P_{|i,i+x|}(n) \quad = \quad P_{i,i+x}(n) - P_{i,i+x-1}(n) - P_{i+1,i+x}(n) + P_{i+1,i+x-1}(n) \tag{4}$$

The last term in the above equation is added because both $[i, i+x-1]$ and $[i+1, i+x]$ contain $[i+1, i+x-1]$. Thus, by subtracting $P_{i,i+x-1}(n)$ and $P_{i+1,i+x}(n)$, we have subtracted $P_{i+1,i+x-1}(n)$ twice.

When the pins are uniformly distributed, $P_{i,i+x}(n) = P_{j,j+x}(n)$ and $P_{|i,i+x|}(n) = P_{|j,j+x|}(n) \quad \forall\ i, j;\ 1 \leq i, j \leq W - x$. In this case, we can drop the dependence on i from $P_{i,i+x}(n)$ and $P_{|i,i+x|}(n)$, representing them as $P_x(n)$ and $P_{|x|}(n)$, respectively. This reduces equation (3) to:

$$\begin{aligned} W(n) \quad &= \quad \sum_{x=0}^{x=W-1} x \sum_{i=1}^{W-x} P_{|x|}(n) \\ &= \quad \sum_{x=0}^{x=W-1} x(W-x) P_{|x|}(n) \end{aligned} \tag{5}$$

Also, when the pins are uniformly distributed, probability that a pin falls within $[i, i+x]$, given that the maximum possible span is $[1, W]$, is given by $\frac{x+1}{W}$. Since the pin locations are independent of each other, the probability that n pins fall within span $[i, i+x]$, given that the maximum possible span is $[1, W]$, is given by $(\frac{x+1}{W})^n$. Thus, we can further simplify equation (4) to:

$$P_{|x|}(n) = (\frac{x+1}{W})^n - 2(\frac{x}{W})^n + (\frac{x-1}{W})^n = \frac{(x+1)^n - 2x^n + (x-1)^n}{W^n} \tag{6}$$

Substituting this in equation (5) gives us the expected width of the bounding-box as:

$$W(n) \quad = \quad \sum_{x=1}^{x=W-1} x(W-x) \frac{(x+1)^n - 2x^n + (x-1)^n}{W^n} \tag{7}$$

This is a closed form expression estimating the width of the bounding box for a net with n pins under a uniform and independent pin distribution. Based on equation (7),

we present the following lemma which gives the expected bounding-box width of 2-pin nets. The lemma can be proved by simple manipulations of equation (7) after setting $n = 2$.

Lemma 2.1

$$W(2) \quad = \quad \frac{W}{3} \tag{8}$$

Dividing equation (7) by equation (8), we can obtain normalized width, denoted N_w, of bounding-box for each pin-count as given below.

$$N_w(n) \quad = \quad \frac{3}{W} \sum_{x=1}^{x=W-1} x(W-x) \frac{(x+1)^n - 2x^n + (x-1)^n}{W^n} \tag{9}$$

We provide the solution to this equation for different values of W in Table 2.

n	3	4	5	6	7	8	9	10
$W = 10$	1.5	1.798	1.99	2.13	2.237	2.317	2.38	2.429
$W = 10^2$	1.5	1.8	2	2.14	2.25	2.333	2.4	2.454
$W = 10^3$	1.5	1.8	2	2.14	2.25	2.333	2.4	2.454
$W = 10^4$	1.5	1.8	2	2.14	2.25	2.333	2.4	2.454

Table 2: Expected normalized width of the bounding-box.

As equation (7) indicates, expected width of the bounding-box is dependent on the final chip width W, implying that it is not useful during logic synthesis where the final chip dimensions are not known a-priori. However, as can be seen from Table 2, value of $N_w(n)$ is relatively independent of W. Indeed, as we show next, for a given range of W and n, we can bound the error by making (7) independent of W by setting $W = \infty$.

Theorem 2.2

$$lim_{W \to \infty} \frac{3}{W} \sum_{x=1}^{x=W-1} x(W-x) \frac{(x+1)^n - 2x^n + (x-1)^n}{W^n} \quad = \quad \frac{3(n-1)}{n+1} \tag{10}$$

Proof Using binomial expansion of $(x+1)^n$ and $(x-1)^n$, we can rewrite RHS of equation (7) as:

$$\frac{3}{W^{(n+1)}} \sum_{x=1}^{x=W-1} (xW - x^2)(x^n + C_1{}^n x^{(n-1)} + C_2{}^n x^{(n-2)} + \cdots + 1 - 2x^n + x^n - C_1{}^n x^{(n-1)} + C_2{}^n x^{(n-2)} + \cdots - 1)$$

After cancelling the corresponding terms, this reduces to:

$$\frac{3}{W^{(n+1)}} \sum_{x=1}^{x=W-1} (xW - x^2)(2{C_2}^n x^{(n-2)} + {C_4}^n x^{(n-4)} + \cdots) \tag{11}$$

For the time being, let us focus on the first term, i.e.,:

$$\frac{3}{W^{(n+1)}} \sum_{x=1}^{x=W-1} (xW - x^2)2{C_2}^n x^{(n-2)}$$

The above can be simplified to:

$$\frac{6{C_2}^n}{W^{(n+1)}}(W \sum_{x=1}^{x=W-1} x^{(n-1)} - \sum_{x=1}^{x=W-1} x^n)$$

As $W \to \infty$, $\sum_{x=1}^{x=W-1} x^{(n-1)} \to \frac{W^n}{n}$ and $\sum_{x=1}^{x=W-1} x^n \to \frac{W^{(n+1)}}{n+1}$. Substituting this in the above, we get:

$$\frac{6{C_2}^n}{W^{(n+1)}}(W\frac{W^n}{n} - \frac{W^{(n+1)}}{n+1}) = \frac{3(n-1)}{n+1}$$

Similarly, for the second term in equation (11) we get:

$$\frac{6{C_4}^n}{W^{(n+1)}}(W\frac{W^{(n-2)}}{n-2} - \frac{W^{(n-1)}}{n-1}) = \frac{n(n-3)}{12W^2}$$

Thus, the second term and all subsequent terms in equation (11) have some positive power of W in denominator, implying that the value of subsequent terms is 0 as $W \to \infty$.

■

n	3	4	5	6	7	8	9	10
$W = 10$	1.5	1.798	1.99	2.13	2.237	2.317	2.38	2.429
$W = \infty$	1.5	1.8	2	2.14	2.25	2.333	2.4	2.454
$\%Error$	0	0.1	0.2	0.3	0.5	0.7	0.8	1.0

Table 3: $N_w(n)$ and the percentage error for $W = 10$ and $W = \infty$.

Thus, when W is assumed to be ∞, $\nu(n)$ is given by:

$$\nu(n) \quad = \quad \frac{3(n-1)}{n+1} \tag{12}$$

Corollary 2.3 *For $W \geq 10$, by using $W = \infty$ (i.e., equation (10)), the error in estimation of $\nu(n)$ is bounded by 1% by when $n \leq 10$, and by 4.7% when $n \leq 40$.*

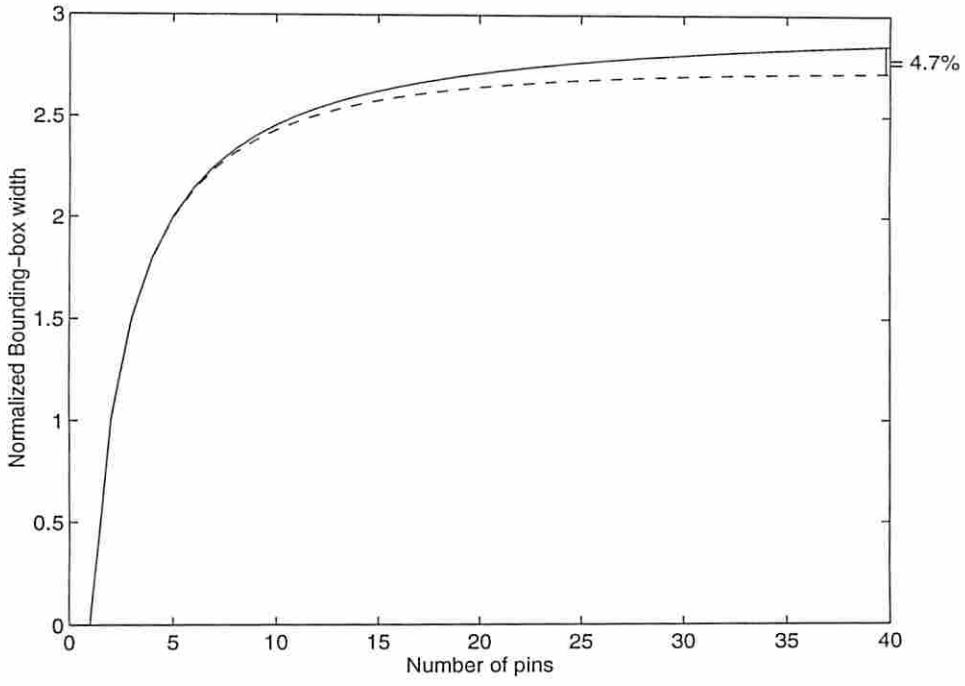


Figure 3: Effect of W on $\nu(n)$, $N_w(n)$ and $N_h(n)$. Solid line corresponds to $W = \infty$ whereas dotted line corresponds to $W = 10$.

The above corollary is a direct results of equation (10) and the values presented in Table 3. As can be seen in Table 3, for $W \geq 10$, error in the value of $W(n)$ is bounded by 1% for $n \leq 10$ if we use $W = \infty$ to estimate $W(n)$. Also, for $n \leq 40$ the error in $W(n)$ is bounded by 4.7% as seen in Figure 3. Indeed, the error bound can be reduced to negligible amount (i.e., less than 0.01% for $n \leq 10$ and less then 0.05% when $n \leq 40$) if it is guaranteed that $W \geq 100$. In our case, we have chosen $W \geq 10$ to allow the same numbers to remain applicable for H also since in many standard cell designs H corresponds to the number of rows which may be as little as 10.

The analysis required to obtain the expected height of the bounding-box is identical and leads to the same results as in Table 2. Thus, expected values of normalized height $N_h(n)$ and perimeter ratio $\nu(n)$ are also identical to the values of $N_w(n)$ presented in Table 2.

2.4 Estimating Normalized Netlengths

As indicated in equation (1), given the equi-perimeter netlength ratios and perimeter ratios, we can calculate the normalized netlengths. The following table characterizes the increase in netlength with respect to an increase in the pin-count of the net. This

table is generated by multiplying the equi-perimeter netlength ratio for each pin-count (characterized by Table 1 and equation (2)) with the corresponding perimeter ratio (characterized by equation (12)). To generate Table 4, we chose $W = \infty$, which guarantees that we have an error bound of at most 1% when $n \leq 10$ and an error bound of at most 4.7% when $n \leq 40$ for $W \geq 10$. As our experimental results verify, in general, more than 95% of the total netlength is due to nets with pin-count of 10 or less. Thus, an inaccuracy greater than 1% for nets with pin-count greater than 10 does not affect the overall netlength estimation.

n	2	3	4	5	6	7	8	9	10
$\eta(n)$	1	1.5	2.7	3	3.57	3.94	4.28	4.8	4.91

Table 4: $\eta(n)$ as a function of n.

The table gives the value of $\eta(n)$ for $n \leq 10$. For $n > 10$, value of $\eta(n)$ can be calculated using the following equation.

$$\eta(n) \quad = \quad \frac{3}{2} \cdot (\sqrt{n} + 1) \cdot \frac{n-1}{n+1} \tag{13}$$

Table 4 and equation (13) characterize the relative netlengths of nets with different pin-counts. This function only depends on n and hence, can be used during logic synthesis. In particular, we can minimize

$$R(N) \quad = \quad \sum_{i=2}^{n_{max}} |N_i| \eta(i) \tag{14}$$

where N_i represents the set of nets with i pins and n_{max} is the maximum pin-count.

However, before adopting this objective function for optimization during logic synthesis, in the next Section, we seek experimental verification of the accuracy of η.

2.5 Evaluation of the Netlength Cost Function

To verify the accuracy of our normalized netlength cost function we compared the expected relative netlength with the actual average netlength obtained on the benchmark circuits generated as explained in Section 2.1. Since we are only capturing the relative netlength contribution and not the absolute netlength, we need to fit the curve produced by Table 4 to the actual curve by multiplying the values in Table 4 by some normalizing constant. The constant used here is the total netlength of all nets divided by the total η of these nets. This is presented in Figure 4.

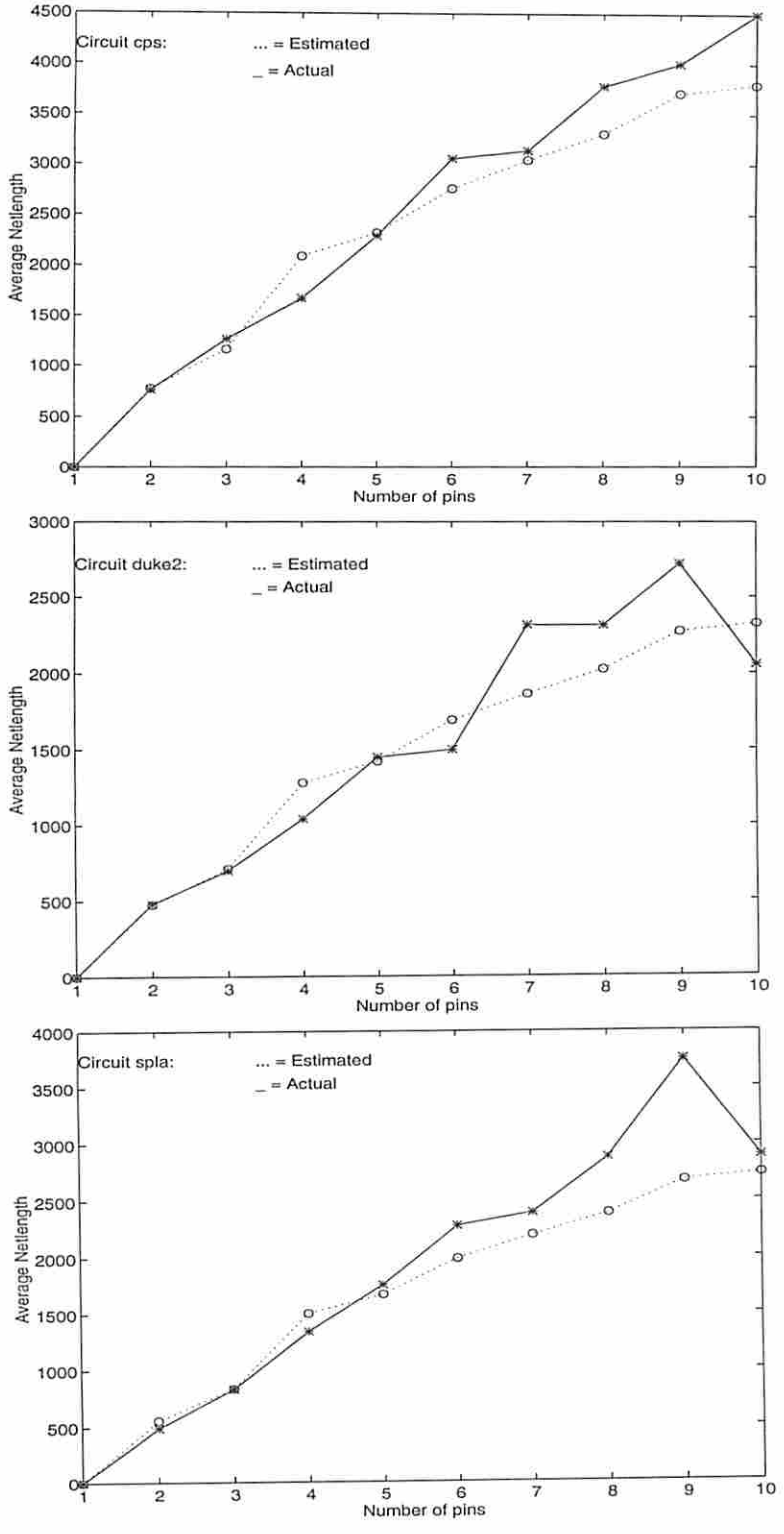


Figure 4: Estimated Vs. Actual average netlengths.

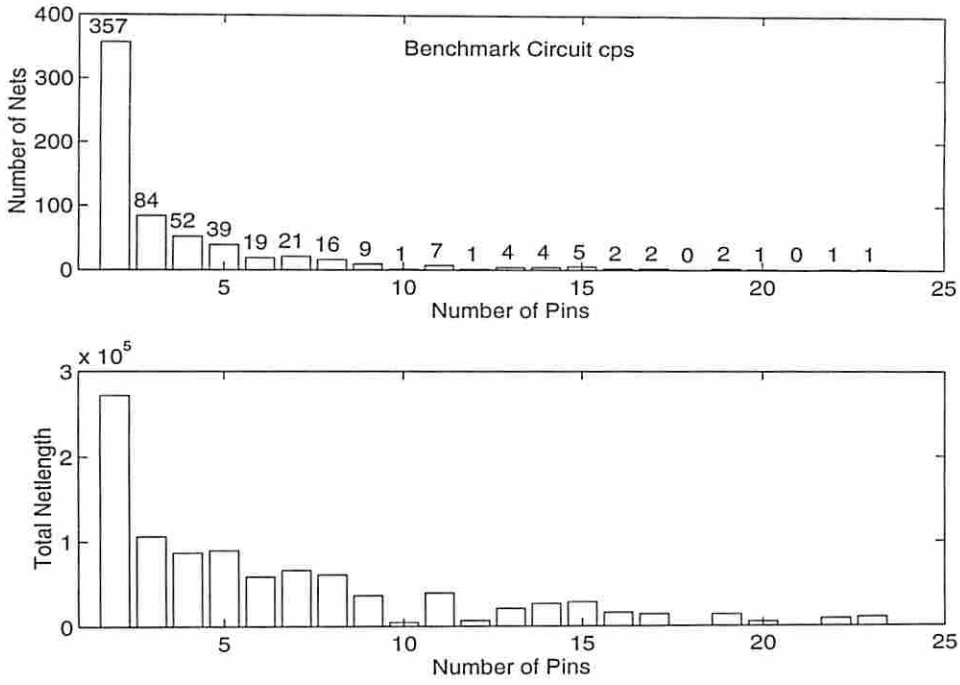


Figure 5: Number of nets with different pin count and sum of their netlengths in circuit cps.

As can be seen from Figure 4, our estimation tracks the actual netlengths quite well. For higher value of n, the actual netlength deviations increase with respect to the expected netlength. This can be attributed to the following. The assumption in calculation of $\rho(n)$ that an optimal rectilinear steiner tree is used to route a net does not hold for larger nets, with larger nets being routed with increasingly suboptimal routing trees by the existing routing tools. However, As can be seen in Figure 5, almost 90-95% of the nets in a circuit have a pin-count of 10 or less. Also, these nets contribute more than 80% of the total netlength of the circuit. Thus, an increased deviation of expected netlengths from the actual netlengths for nets with higher pin-count has does not affect the total netlength significantly.

3 PIX: Pin-count Based Extraction

The relationship derived between the routing cost (i.e., total netlength) and the pin-count in the previous section can be exploited during logic synthesis to guide the optimization procedures. In this report, we target the operation of logic extraction to minimize the objective function in equation (14). Before proposing our routing-driven approach for extraction, i.e., extraction to minimize total netlength, we present a brief

introduction to extraction.

3.1 Background

Initial work on extraction was reported by Roth and Karp [16] and by Lawler [8]. Both these approaches apply boolean techniques for extraction. However, computational complexity of these techniques render them impractical for large circuits. Hence, recent approaches use the following procedure along with algebraic division methods to extract common divisors [2, 1, 14].

```
Algorithm 1 Extract(N)
N is a given boolean network
01 begin
02   Generate candidate divisors G
03   do
04     BestLiteralSavings = 0
05     BestDivisor = NULL
06     ForEachDivisor D ∈ G
07       if LiteralSavings(D) ≥ BestLiteralSavings
08         BestDivisor = D
09         BestLiteralSavings = LiteralSavings(D)
10     Divide the boolean equations by BestDivisor
11     Update the weight of remaining candidate divisors
12   While A candidate divisor of sufficient merit exist
13 end
```

First, all candidate divisors that can be extracted from nodes in the network are generated and common divisors are detected. A divisor may consist of a single cube or multiple cubes. A multiple cube divisor which cannot be further divided by a single cube, i.e., a cube-free divisor, is called a kernel. The quotient of the division process is also referred to as a *cokernel.* A cost value is then assigned to each divisor. This cost value has traditionally been the literal savings of the divisor. A divisor that has the best cost is then selected and introduced in the network. After dividing the fanout nodes of a divisor by the selected divisor, the cost of the remaining candidate divisors might change. Hence, it is necessary to update the weight of remaining candidate divisors after each extraction.

For example, consider $f = abeg + aceg + deg + h$. This function can be divided by $d_1 = b + c, d_2 = ab + ac + d$, and $d_3 = ab + ac$ to yield $f = ad_1eg + deg + h, f = d_2eg + h$,

and $f = d_3eg + deg + h$, respectively. In the above example d_1 and d_3 are double divisors while d_2 is a three cube divisor. Also, d_2 is not a kernel as it can be further divided by the single cube a. Cokernels of kernel d_1 and d_2 above are aeg and eg respectively. It should be noted that a cokernel is, in fact, a single cube divisor of the function. A kernel is of level 0 if it contains no kernels except itself. A kernel is of level n if it contains at least one kernel of level $n-1$ and no kernels of level n of higher. Since d_2 above can be represented as $ad_1 + d$, d_2 is of level 1. However, d_1 is a kernel of level 0.

Kernel based decomposition and a mechanism to derive and extract common kernels (based on rectangle covering problem) was proposed by Brayton et al. [1]. They also proposed a modified mechanism to identify common single-cube divisors in a network. Rajski and Vasudevamurthy [14] simplified the problem by considering only cube-free double divisors, i.e., kernels with two cubes and single divisors with two literals[1]. This approach (which is also referred to as the "fast-extract" approach) improves the run time while not sacrificing the quality of resultant circuits. Since the pin-count based extraction algorithm proposed in this report is based on "fast-extract" mechanism proposed in [14], we first present a brief description of their work.

A double divisor (denoted by $D_{i,j,k}$ where i = *number of literals in first cube*, j = *number of literals in the second cube* and k = *total number of variables appearing in divisor D*) is generated by intersecting two cubes with at least one literal in common and then taking the sum of product of the remaining literals in each cube. Literals belonging to the intersection form the cokernel of the corresponding occurrence of the divisor. Consider the example presented above, i.e., $f = abeg + aceg + deg + h$. Here, $c_1 = \{a, b, e, g\}, c_2 = \{a, c, e, g\}, c_3 = \{d, e, g\}$ and $c_4 = \{h\}$. All bases and double divisors for this function can be identified by performing following cube intersections: $b_1 = c_1 \cap c_2 = \{a, e, g\} \Rightarrow d_1 = b + c; b_2 = c_1 \cap c_3 = \{e, g\} \Rightarrow d_2 = ab + d; b_3 = c_2 \cap c_3 = \{e, g\} \Rightarrow d_3 = ac + d$. As can be observed, if a function has C cubes, all double divisors of that function could be identified in $O(C^2)$. Given a circuit with n nodes and with C_{max} being the maximum number of cubes in SOP representation of any node, total number of double divisors identified is $O(nC_{max}{}^2)$. The restriction to double divisors reduces the possible number of divisors from exponential (in case of kernel extraction) to polynomial simply by reducing the granularity of each divisor. Also, circuits corresponding to divisors with more cubes can still be generated by extracting a double divisor containing an already extracted divisor. For example, in the above, a three cube divisor $d = ab + ac + d$ can be emulated by first extracting $d_2 = ab + d$ and then extracting $d_4 = d_2 + ac$. In case of single divisors, all possible two literal single cube divisors are generated by performing a pairwise intersection of the set of cubes in which the literals appear. This is achieved by performing an intersection of two columns in the literal-cube matrix [1]. For example, in the above, cube-set of literal e is given by $\{c_1, c_2, c_3\}$ whereas the cube-set of g is also given by $\{c_1, c_2, c_3\}$. Thus, eg is a valid

[1]The quotient corresponding to a double divisor is referred as the *base* in their work. We follow their convention and refer to a cokernel as base from now on.

two literal divisor with the set of occurrences given by $\{c_1, c_2, c_3\}$.

3.2 Routing-Driven Extraction

For the pin-count based routing cost described in Section 2 (i.e., equation (14)), we implemented a corresponding extraction procedure. The greedy scheme of algorithm 1 was used with all divisors restricted to be either double divisors, or single divisors with two literals as in [14].

From the list of candidate divisors, conventional approaches choose a divisor which provided the best literal savings while we choose "good" divisor that optimizes the routing cost. A "good" divisor is selected from the list of candidate divisors based on their literal savings potential to ensure that the reduction of routing area is not achieved at the cost of a substantial increase in the active area. Since the basic algorithm is greedy, though we minimize equation (14) at each step, it is not guaranteed that we produce a circuit with minimum value of equation (14). However, just as in the case with the literal savings objective function, we expect that it will tend to minimize equation (14).

The strategy to select "good" divisors from the list of candidate divisors is as follows: if maximum literal savings from any divisor is M, select all divisors with literal savings within $p\%$ of M. In our experiments, we used divisors with top 25% literal savings.

A generic mechanism to perform a routing driven extraction based on the above pin-count based cost function is given in algorithm 2. Line 8 of the algorithm was implemented to report the routing cost based on equation (14). It should be noted that if complexity of $NetLnCost(D)$ is same as $LiteralSavings(D)$, algorithm 2 has the same complexity as algorithm 1. In the next subsection, we describe how we calculate $NetLnCost(D)$ of a candidate divisor D.

Algorithm 2 NetLn_Extract($\mathcal{N}, \mathcal{P}$)
$\mathcal{N}$ is a given boolean network
$\mathcal{P}$ is % value to identify good divisors
01 **begin**
02 Generate candidate divisors G
03 **do**
04 BestLiteralSavings = 0
05 BestNetLnCost = $-\infty$
06 BestDivisor = **NULL**
07 **ForEachDivisor** D $\in$ G
08 **if** LiteralSavings(D) $\geq 1 - \mathcal{P}/100$*BestLiteralSavings
09 **if** LiteralSavings(D) $\geq$ BestLiteralSavings
10 BestLiteralSavings = LiteralSavings(D)
11 **if** NetLnCost(D) $\leq$ BestNetLnCost
12 BestDivisor = D
13 BestNetLnCost = NetLnCost(D)
14 Divide the boolean equations by BestDivisor
15 Update the cost of remaining candidate divisors
16 **While** a candidate of sufficient merit exists
17 **end**

3.3 Netlength Cost of a Divisor

The netlength cost of a divisor is calculated as the change in the total netlength of the circuit due to the extraction. Suppose as a result of extraction, a node D with a_D outputs is introduced in the original circuit N. Let the resultant circuit be denoted by N'. The circuit routing cost before extraction is given by equation(14). Then we need to calculate $\Delta(D)$ where $\Delta(D) = R(N') - R(N)$. Apart from the additional routing cost of a new net corresponding to the output of D, pin-counts of already existing nets might also change. Hence, $\Delta(D) \neq \eta(a_D)$. However, by identifying all the nets whose pin-count might change we can calculate $\Delta(D)$ efficiently as explained next.

Assume that a double divisor D consists of set of literals $L_D = \{l_1, l_2, \ldots, l_m\}$, each of which appears $a_1, a_2, \ldots, a_m$ times in the circuit before the extraction of D. If the new divisor D appears a_D times in the circuit after the extraction, then appearance of each $l_i \in L_D$ is reduced from a_i to $a_i - a_D + 1$, which will reduce the routing cost of these literals. However, a new node D with $a_D + 1$ pins is introduced. Apart from this, if B_k denotes the set of literals in the base of the kth appearance of divisor D, where $1 \leq k \leq a_D$, then for every literal ${l_j}^k \in B_k, 1 \leq k \leq a_D$, the original appearance of the literal (denoted ${a_j}^k$) is reduced by one (since D is a double divisor).

For every candidate divisor we calculate the routing cost before extraction as:

$$\sum_{l_i \in L_D} \eta(a_i + 1) + \sum_{l_j{}^k \in B_k, 1 \leq k \leq a_D} \eta(a_j{}^k + 1)$$

After the extraction of a double divisor, routing cost is:

$$\sum_{l_i \in L_D} \eta(a_i - a_D + 2) + \sum_{l_j{}^k \in B_k, 1 \leq k \leq a_D} \eta(a_j{}^k) + \eta(a_D + 1)$$

Hence, the change in routing cost due to the extraction, namely, $\Delta(D)$, is given by

$$\begin{aligned} \Delta(D) \quad = \quad & \sum_{l_i \in L_D} \{\eta(a_i + 1) - \eta(a_i - a_D + 2)\} + \\ & \sum_{l_j{}^k \in B_k, 1 \leq k \leq a_D} \{\eta(a_j{}^k + 1) - \eta(a_j{}^k)\} - \eta(a_D + 1) \end{aligned} \tag{15}$$

Or in case of a single divisor

$$\Delta(D) \quad = \quad \sum_{l_i \in L_D} \{\eta(a_i + 1) - \eta(a_i - a_D + 1)\} - \eta(a_D + 1) \tag{16}$$

This extraction procedure based on pin count based routing measure attempts to minimize the routing cost of the final circuit by choosing a divisor with maximum route savings during each iteration of greedy heuristic. The experimental results are presented in next Section.

It should be noted that the traditional area cost, namely, the number of literals saved due to extraction, can be obtained from our divisor costs (i.e., equations (15) and (16)) simply by removing the η parameter (along with 1 pin corresponding to the net source introduced for calculating η) as shown below.

In case of double divisors, we can reduce equation (15) as follows:

$$\begin{aligned} & \sum_{l_i \in L_D} \{\eta(a_i + 1) - \eta(a_i - a_D + 2)\} + \sum_{l_j{}^k \in B_k, 1 \leq k \leq a_D} \{\eta(a_j{}^k + 1) - \eta(a_j{}^k)\} - \eta(a_D + 1) \\ \rightsquigarrow \quad & \sum_{l_i \in L_D} \{a_i - a_i + a_D - 1\} + \sum_{l_j{}^k \in B_k, 1 \leq k \leq a_D} \{a_j{}^k - a_j{}^k + 1\} - a_D \\ = \quad & \sum_{l_i \in L_D} \{a_D - 1\} + \sum_{l_j{}^k \in B_k, 1 \leq k \leq a_D} \{1\} - a_D \\ = \quad & |L_D| * (a_D - 1) + \sum_{k, 1 \leq k \leq a_D} \{|B_k|\} - a_D \end{aligned}$$

which is exactly the literal savings of a double divisor.

Likewise, equation (16) can be reduced as follows:

$$\sum_{l_i \in L_D} \{\eta(a_i + 1) - \eta(a_i - a_D + 1)\} - \eta(a_D) \quad \rightsquigarrow \quad \sum_{l_i \in L_D} \{a_i - a_i + a_D\} - a_D$$
$$= \quad |L_D| * a_D - a_D = a_D * (|L_D| - 1)$$

Thus, the only difference in calculating equations (15) and (16) with respect to the literal savings objective function is that, we need to know variables $a_i, \forall l_i \in L_D$ and ${a_j}^k, \forall {l_j}^k \in B_k, 1 \leq k \leq a_D$. In other words, we need to know the original number of occurrences in the circuit for the literals appearing in the divisor as well as for literals appearing in the base of each occurrence of the candidate divisor. This information can be pre-calculated before each extraction iteration in $O(E)$ time along with the circuit update in line 13 and/or 14 and stored at each node in the circuit. This implies that time complexity of $NetLnCost(D)$ is same as the time complexity of $LiteralSavings(D)$. Thus, the overall time complexity of algorithm 2 is identical to the time complexity of algorithm 1.

4 Experimental Results and Discussion

We implemented the *PIn-count based eXtraction* (PIX) procedure within the SIS synthesis environment. Before presenting results with our extraction algorithm, we produce in Table 5, the results obtained by optimizing circuits in SIS using the "script.rugged" (which uses "fast_extract" to perform extraction). To generate our results, the circuits were mapped using the SIS mapper with LIB2 gate library in delay mode, placed using GORDIAN [6] and routed using TimberWolf global router [9] and YACR2 detailed router [15]. We report *post-route* route area, chip area and circuit delay. Results presented in Tables 5 are used as a basis for comparison with our experiments.

In our experiments with PIX, traditional extraction operations in *script.rugged*, namely, "fast_extract", was replaced by PIX. We refer to this modified scripts as "script.rugged.pix". Other operations in *script.rugged* were kept intact.

Before presenting our experimental results, let us discuss an issue that demanded attention during the implementation. When a node/divisor has multiple occurrences in the same node, the question arises whether we should count this as one pin or as multiple pins to calculate our pin-count based routing cost. We believe that it is more appropriate to consider multiple occurrences in the same node as a single pin on the corresponding net. This amounts to using the number of fanouts of a node to calculate $\eta(n)$ instead of identifying and accounting for each occurrence of the node in each of its fanout. The main justification for this is that after the circuit is mapped onto the library gates, even for multiple occurrence in the gate (i.e., an "XOR" gate) connection to only

Ckt	Route Area	Gate Area	Chip Area	Delay
b12	165	844	249	9.63
cps	10787	1299	12086	56.86
duke2	1797	474	2271	23.77
ex1010	40066	3112	43178	205.78
ex4	1357	518	1876	13.83
misex3c	2064	550	2614	46.67
pdc	1951	504	2455	20.95
rd84	275	148	423	14.28
spla	3313	727	4040	26.12
Z5xp1	332	140	472	36.77
Z9sym	691	221	912	15.04
alu4	2968	628	3597	32.59
apex2	1289	382	1671	18.88
apex3	18217	1755	19972	78.10
clip	342	146	488	16.45
misex3	3881	813	4694	33.47
seq	17071	1917	18989	61.02

Table 5: Results after routing for circuits optimized using "script.rugged"

a single pin is required. Also, with this approach, we are giving an extra incentive to our extraction procedure to extract a divisor that reduces the immediate support of the fanouts by completely extracting out some variables. Our experimental results verify that using the number of fanouts is indeed more appropriate than using the number of occurrences. An additional advantage of this is that now we need not calculate $\eta(n)$ for the literals in the base of a divisor because the number of fanouts for the literals in the base remain unchanged subsequent to an extraction.

Circuit	Route Area	Gate Area	Chip Area	Delay
b12	1.01	1.10	1.04	1.07
cps	1.03	1.11	1.04	0.76
duke2	0.90	0.88	0.90	1.23
ex1010	0.93	0.94	0.93	1.02
ex4	0.96	1.00	0.97	0.98
misex3c	0.92	1.02	0.94	0.95
pdc	0.80	0.89	0.82	1.05
rd84	0.97	0.91	0.95	1.11
spla	0.90	0.93	0.91	1.03
Z5xp1	0.79	0.96	0.84	1.00
Z9sym	0.70	0.80	0.73	0.95
alu4	0.91	1.01	0.93	0.93
apex2	0.86	0.95	0.88	1.10
apex3	0.95	0.98	0.95	1.07
clip	0.84	0.98	0.88	0.96
misex3	0.94	0.96	0.94	1.14
seq	0.92	1.05	0.93	0.85
AVERAGE	0.90	0.97	0.92	1.01

Table 6: Results using "script.rugged.pix"

The results of our approach are given in table 6. Each entry in the table is normalized with respect to the corresponding entry in table 5. As can be seen, on average, we obtained 10% improvement in routing and 8% improvement in chip area for about the same delay.

5 Concluding Remarks

In this report, we presented a simple cost function that captures the relative lengths of nets based on their pin-counts. Unlike the routing cost functions proposed earlier, our cost function does not require layout-parameters or any tuning of the variables, and thus, is ideal for minimization of routing contribution during logic synthesis. A comparison of the relative netlengths estimated by our cost function with actual netlengths verify that it indeed captures relative netlengths accurately. We further illustrated an application of this cost function by minimizing it during the operation of logic extraction. As experimental results indicate, minimizing the proposed objective function improves the routing and chip area at no degradation in delay.

Our future work will focus on developing other routing cost functions for logic synthesis and on experimental evaluation and and comparison of various functions. In particular, we will focus on characterizing circuits for which extraction based on some routing cost functions is very effective. Also, an increase in the active gate area could lead to an increase in the number of rows required to maintain unit aspect ratio. For large circuits, this often led to significant improvements in routing area. A characterization of the exact effect of number of rows on the routing area is thus needed. We intend to explore this topic further.

References

[1] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. Multi-level logic optimization and the rectangular covering problem. In *Proceedings of the IEEE International Conference on Computer Aided Design*, Nov. 1987.

[2] R. K. Brayton and C. McMullen. The decomposition and facorization of Boolean expressions. In *Proceedings of the International Symposium on Circuits and Systems*, pages 49–54, Rome, May 1982.

[3] F. R. K. Chung and F. K. Hwang. The largest minimal rectilinear Steiner trees for a set of n points enclosed in a rectangle with given perimeter. *Networks*, 9:19–36, 1979.

[4] W. E. Donath. Hierarchical structure of computers. Technical Report RC 2392, IBM T. J. Watson Research Center, March 1969.

[5] W. E. Donath. Placement and average interconnection lengths of computer logic. *IEEE Transactions on Circuits and Systems*, CAS-26(4):272–277, April 1979.

[6] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich. GORDIAN: VLSI placement by quadratic programming and slicing optimization. *IEEE Transactions on Computer-Aided Design*, CAD-10:356–365, March 1991.

[7] F. J. Kurdahi and A. C. Parker. Techniques for area estimation of VLSI layouts. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-8(1):81–92, January 1989.

[8] E. L. Lawler. An approach to multilevel Boolean minimization. *Journal of the Association for Computing Machinery*, 11, July 1964.

[9] K. W. Lee and C. Sechen. A new global router for row-based layout. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 180–183, November 1988.

[10] S. Malik, E. M. Sentovich, R. Brayton, and A. Sangiovanni-Vincentelli. Retiming and resynthesis: Optimizing sequential circuits using combinational techniques. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 10, pages 74–84, January 1991.

[11] M. Pedram and N. Bhat. Layout driven logic restructuring / decomposition. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 134–137, November 1991.

[12] M. Pedram and B. T. Preas. Interconnection length estimation for optimized standard cell layouts. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 390–393, October 1989.

[13] M. Pedram and H. Vaishnav. Technology decomposition using optimal alphabetic trees. In *Proceedings of the European Conf. on Design Automation*, pages 573–577, March 1993.

[14] J. Rajski and J. Vasudevamurthy. The testability-preserving concurrent decomposition and factorization of boolean expressions. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 11, pages 778–793, June 1992.

[15] J. Reed, A. Sangiovanni-Vincentelli, and M. Santamauro. A new symbolic channel router: YACR2. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 208–219, July 1985.

[16] J. Roth and R. Karp. Minimization over boolean graphs. *IBM Journal of Research and Development*, 6(2):227–238, April 1962.

[17] S. Sastry and A. C. Parker. Stochastic models for wirability analysis of gate arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-5(1):52–65, January 1986.

[18] G. Saucier, J. Fron, and P. Abouzeid. Lexicographical expressions of boolean functions with applications to multilevel synthesis. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 12, pages 1642–1654, November 1993.

[19] C. Sechen. Average interconnection length estimation for random and optimized placements. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 190–193, November 1987.

[20] Herve Touati. *Performance Oriented Technology Mapping.* PhD thesis, University of California, Berkeley, 1990.

[21] H. Vaishnav and M. Pedram. Routability-driven fanout optimization. In *Proceedings of the 30th Design Automation Conference*, pages 230–236, June 1993.

[22] Hirendu Vaishnav and Massoud Pedram. Minimizing the routing cost during logic extraction. In *Proceedings of the 32nd Design Automation Conference*, June 1995.