A Novel Net Weighting Algorithm for Timing-Driven Placement

Tim (Tianming) Kong
Aplus Design Technologies, Inc.
10850 Wilshire Blvd., Suite #370
Los Angeles, CA 90024

Abstract

Net weighting for timing-driven placement has been very popular in industry and academia. It has various advantages such as low complexity, high flexibility and ease of implementation. Existing net weighting algorithms, however, are often *ad-hoc*. There is generally no known *good* net weighting algorithms. In this paper, we present a novel net weighting algorithm based on the concept of path-counting, and apply it in timing-driven FPGA placement application. Theoretically this is the first ever known *accurate*, all-path counting algorithm. Experimental data shows that compared with the weighting algorithm used in state-of-the-art FPGA placement package VPR[1], this new algorithm can achieve the longest path delay reduction of up to 38.8%, 15.6% on average with no runtime overhead and only a 4.1% increase in total wirelength.

1 Introduction

Timing-driven placement has been studied extensively. Existing approaches can be broadly divided into two classes: *path-based* and *net-based*.

A typical path-based approach usually considers all or a subset of paths directly into the problem formulation. The majority of this class of approaches are based on mathematical programming techniques[2][3][4][5]. This class of algorithms usually maintain an accurate timing view during optimization. The major drawback is the relatively high complexity.

Unlike path-based approaches which handle paths directly, net-based approaches usually transform timing information into either net weight or net length (or delay) constraints, and employ a weighted wirelength minimization engine. *Net weighting* [6][7][8][9][10][1] is a technique commonly used in this class. The basic idea is to put a higher weight for nets that are more timing critical. Net weighting techniques have some favorable properties: relatively low complexity, strong flexibility and easy implementation. With more and more complicated timing constraints present in modern circuits, such as multiple clock domains, multiple cycle paths, etc., these advantages make the net weighting method more attractive.

Unfortunately, net weighting is done often in an *ad-hoc* manner. Since circuit timing is inherently path-oriented, a good net weighting algorithm should assign net weights based on path analysis. However, the existence of an exponential number of paths makes it very difficult, if not impossible, to exhaustively analyze all paths in a circuit. Therefore, weighting has to be performed repeatedly. How to stabilize net weighting in order to achieve good convergence is a big open issue.

In this paper, we study the net weighting based timing-driven placement problem, and present a new net weighting algorithm. The main contributions of this work are:

- We developed an *accurate* path counting algorithm. To the best of our knowledge, this is the first known accurate path counting algorithm that considers all paths. Due to the exponential number of paths present in a circuit, accurate all-path counting has generally been considered very difficult.
- Significant performance improvement has been achieved with little loss in total wirelength and no runtime overhead.

The rest of this paper is organized as follows. Section 2 describes background information. Section 3 covers the derivation of our new net weighting algorithm. Experimental data is presented in Section 4, followed by conclusions in Section 5.

2 Background

Timing-driven placement usually employs a static timing analysis engine to compute the delay of the longest path in a circuit. The results of this timing analysis are then used by the placement optimization engine to minimize the longest path delay.

Static timing analysis engines usually model the timing of a circuit as a directed acyclic graph $G(V, E)$: input and output pins of circuit elements are represented as nodes, while connections between pins are modeled as edges. Edges are labeled with delay values. A path starts at a primary input or at an output pin of a memory element, and ends at a primary output or an input pin of a memory element. We will use $\mathcal{P}_\mathcal{I}$ and $\mathcal{P}_\mathcal{O}$ to represent the start pin and terminating pin set of any path, respectively.

The longest path delay can be determined by computing the *arrival time*, iteratively, making use of the following equation:

$$ARR(t) = \begin{cases} 0 & t \in \mathcal{P}_\mathcal{I} \\ \max\limits_{(s,t)\in E} \{ARR(s) + d(s,t)\} & \text{otherwise} \end{cases}$$

where $d(s,t)$ is the delay of edge (s,t). The longest path delay is,

$$T = \max_{t\in\mathcal{P}_\mathcal{O}} ARR(t).$$

Similarly we can also compute the *required arrival time* for each pin using the following equation:

$$REQ(s) = \begin{cases} T & s \in \mathcal{P}_\mathcal{O} \\ \min\limits_{(s,t)\in E} \{REQ(t) - d(s,t)\} & \text{otherwise} \end{cases}$$

We can now compute for each edge the *slack* value, which measures how much additional delay can be added to an

0-7803-7607-2/02/$17.00 ©2002 IEEE

```
1.   set F(p) = B(p) = 0 for each pin p;
2.   for each P_I pin p, set F(p) = 1 if critical;
3.   traverse each pin t in topological order
4.       if (t is a critical pin)
5.           for each critical input pin s of t
6.               F(t) += F(s);
8.
9.   set B(p) = 1 for critical P_O pin p;
10.  traverse each pin s in reverse topological order
11.      if (s is a critical pin)
12.          for each critical output pin t of s
12.              B(s) += B(t);
14.
15.  for each edge (s,t), set
16.      GP(s,t) = F(s) × B(t);
```

Table 1: Algorithm GPATH – counting the number of critical paths passing through each edge.

```
1.   set F(p) = B(p) = 0 for each pin p;
2.   for each P_I pin p, set F(p) = 1;
3.   traverse each pin t in topological order
4.       for each input pin s of t
5.           if (ARR(s) + d(s,t) == ARR(t))
6.               set F(t) += F(s);
9.
10.  set B(p) = 1 for critical P_O pin p;
11.  traverse each pin s in reverse topological order
12.      for each output pin t of s
13.          if (REQ(t) − d(s,t) == REQ(s))
14.              set B(s) += B(t);
17.
18.  for each edge (s,t), set
19.      LP(s,t) = F(s) × B(t);
```

Table 2: Algorithm LPATH – counting the number of local critical paths passing through each edge.

edge without increasing the longest path delay of the whole circuit. The slack of a given edge (s,t) can be computed as:

$$slack(s,t) = REQ(t) - ARR(s) - d(s,t).$$

The delay of any path is the total delay of its constituent edges, i.e.,

$$d(\pi) = \sum_{e \in \pi} d(e).$$

Similarly the slack of a path is defined to be the maximum amount of delay which can be added to the path before it becomes critical. Thus a path's slack can be computed as:

$$slack(\pi) = T - d(\pi).$$

3 Path Counting Based Weighting Algorithm (PATH)

A good net weighting algorithm should take the path sharing effect into consideration. Intuitively, if two critical paths share a common segment, the edges in the common segment should receive higher weights. Path counting is a general way to assign net weights with consideration of such effect.

3.1 Critical Path Counting

PATH is motivated by the following traditional critical path counting problem: given a timing graph, after performing timing analysis for edges present in some critical path, compute the total number of critical paths passing each edge.

This problem can be readily solved by the following simple algorithm. For each pin p in the timing graph, we define two variables:

- Forward path $F(p)$ – the number of different critical paths starting from $\mathcal{P}_\mathcal{I}$ elements, terminating at p.
- Backward path $B(p)$ – the number of different critical paths starting from $\mathcal{P}_\mathcal{O}$ elements, terminating at p, if we reverse all signal flow directions.

The number of different critical paths passing through an edge (s,t) can then be computed as:

$$\mathbf{GP}(s,t) = F(s) \times B(t)$$

An efficient algorithm (GPATH) for computing $F(p), B(p)$ and $\mathbf{GP}$ is shown in Table 1.

3.2 Locally Most Critical Path Counting

An immediate extension of GPATH allows us to count for each non-critical edge the number of paths which are most critical with respect to this edge, i.e., the number of paths whose slack is equal to this edge's slack. We still use values $F(p), B(p)$ while they now account for local critical paths only. The algorithm (LPATH) is illustrated in Table 2. We need to point out that a timing driven packing tool, T-VPack[11] used $F(s)+B(t)$ as a tie-breaking factor when assigning weight to edge (s,t).

3.3 Accurate All Path Counting

One could use the number of critical paths or locally most critical paths passing through each edge to assign net weights. The drawback is that they are either inaccurate in handling non-critical paths or not considering them at all.

Our further extension, algorithm PATH as shown in Table 3 is *accurate* for certain type of *discount* functions (to be defined below). The following is a general formulation[1]. for timing optimization:

$$\begin{array}{ll} \min & \mathcal{F}(x) \\ \text{s.t.} & \sum_{e \in \pi} d(e) \le T,\ \forall \pi \in \mathcal{P} \end{array}$$

where $\mathcal{F}(x)$ is any given objective function, $\mathcal{P}$ is the set of all paths present in the design. One could transform the problem into an unconstrained optimization problem for the purpose of timing optimization as the following:

$$\min \quad \mathcal{F}(x) + \sum_{\pi \in \mathcal{P}} \mathcal{D}(slack(\pi), T) d(\pi)$$

and thus naturally assign weight to a particular edge $e(s,t)$ as:

$$\mathbf{PW}_\mathcal{D}(e) = \sum_{\pi \ni e} \mathcal{D}(slack(\pi), T)$$

where $\mathcal{D}(x,y)$ is a given *discount* function. We call $\mathcal{D}(x,y)$ a *discount* function because it is normally given as a monotonically decreasing function with x as we wish to put smaller weight (thus more discount) on paths with larger slack.

Due to the existence of an exponential number of paths, for general discount function $\mathcal{D}$ it is very difficult if not impossible, to compute $\mathbf{PW}(e)$ both efficiently and accurately.

[1]Here we ignore potentially other constraints, such as non-overlap constraints.

1.	set $F(p) = B(p) = 0$ for each pin p;
2.	for each $\mathcal{P}_\mathcal{I}$ pin p, set $F(p) = 1$;
3.	traverse each pin t in topological order
4.	for each input pin s of t
5.	$F_s(s,t) = ARR(t) - ARR(s) - d(s,t)$;
6.	compute $discount = \mathcal{D}(F_s(s,t), T)$;
7.	$F(t)$ += $discount \times F(s)$;
10.	
11.	for each $\mathcal{P}_\mathcal{O}$ pin p, set $B(p) = 1$;
12.	traverse each pin s reverse topological order
13.	for each output pin t of s
14.	$B_s(s,t) = REQ(t) - d(s,t) - REQ(s)$;
15.	compute $discount = \mathcal{D}(B_s(s,t), T)$;
16.	$B(s)$ += $discount \times B(t)$;
19.	
20.	for each edge (s,t), compute
21.	$\mathbf{AP}(s,t) = F(s) \times B(t) \times \mathcal{D}(slack(s,t), T)$;

Table 3: Algorithm PATH – accurately computing the impact of all paths passing through each edge.

One important contribution of this work is the following result:

Theorem 3.1 *For any exponential discount function $\mathcal{D}$, i.e., if there exists a positive constant number a, $\mathcal{D}(x,y) = a^{-\frac{x}{y}}$, $\mathbf{PW}_\mathcal{D}(s,t) = \mathbf{AP}_\mathcal{D}(s,t)$ as computed by algorithm PATH shown in Table 3.*

Proof. First of all, notice that we use the following two concepts in this algorithm:

- Forward local slack for edge $e(s,t)$, which is computed as $F_s(s,t) = ARR(t) - ARR(s) - d(s,t)$;
- Backward local slack for edge $e(s,t)$, computed as: $B_s(s,t) = REQ(t) - d(s,t) - REQ(s)$

Forward local slack represents the relative arrival time criticality with respect to the current sink pin t: t may have several fanin edges; each of them has different forward local slack. The one(s) with zero forward local slack is (are) the most timing critical one(s) for improving the timing for t, any increase in edge delay will lead to an increase of t's arrival time; while those with positive local slack can afford some delay increase without sacrifice of t's arrival time. Backward local slack similarly represents the relative required arrival time criticality with respect to a source pin s.

Let's examine the contribution of any path π to $\mathbf{AP}_\mathcal{D}$. Suppose π consists of the following edges: e_1, e_2, ..., e_k, $e(s,t)$, e_{k+1}, e_{k+2}, ..., e_m. The contribution of π to $\mathbf{AP}_\mathcal{D}(e)$ is:

$$\begin{aligned}\mathbf{AP}_\mathcal{D}(\pi, e) = \; & \prod_{i=1}^{k} \mathcal{D}(F_s(e_i), T)) \times \mathcal{D}(slack(e), T) \\ & \times \prod_{i=k+1}^{m} \mathcal{D}(B_s(e_i), T)\end{aligned}$$

Notice that

$$slack(\pi) = \sum_{i=1}^{k} F_s(e_i) + slack(e) + \sum_{i=k+1}^{m} B_s(e_i)$$

and,

$$\mathcal{D}(x_1, y) \times \mathcal{D}(x_2, y) = \mathcal{D}(x_1 + x_2, y)$$

we have:

$$\begin{aligned}\mathbf{AP}_\mathcal{D}(\pi, e) &= \mathcal{D}(\textstyle\sum_{i=1}^{k} F_s(e_i) + slack(e) + \sum_{i=k+1}^{m} B_s(e_i), T) \\ &= \mathcal{D}(slack(\pi), T)\end{aligned}$$

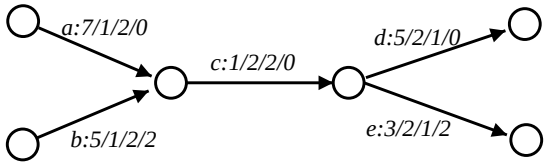

Figure 1: A simple example showing that counting of $\epsilon-$critical paths could be inaccurate. The label at each edge (s,t) is: $delay(s,t)/F(s)/B(t)/slack(s,t)$. For $\epsilon = 2$, there are only 3 $\epsilon-$critical paths passing through c: $\{a,c,d\}, \{a,c,e\}$ and $\{b,c,d\}$ with slack value of 0, 2, and 2 respectively. A simple counting algorithm might include path $\{b,c,e\}$, thus give a wrong answer of 4.

Therefore, summing over all paths, we establish our main result:

$$\mathbf{PW}_\mathcal{D}(s,t) = \mathbf{AP}_\mathcal{D}(s,t) \qquad \square$$

Here we would like to point out the inaccuracy of a commonly used path counting method: identifying all the edges whose slack is no more than a certain threshold ϵ, and get the so-called $\epsilon-$critical network. Then count the total number of paths passing through each edge in this network. A similar algorithm is used in [12] for detailed placement.

This seemingly innocent algorithm is actually inaccurate: it may overcount the number of $\epsilon-$critical paths passing through certain edges. The example in Figure 1 demonstrates how this may happen. In this example, there are 3 $\epsilon-$critical paths, however if we follow the above mentioned algorithm, we will count 4 paths for edge c.

The reason of such inaccuracy is because although all edges in the $\epsilon-$critical network are $\epsilon-$critical, not all paths in the network are $\epsilon-$critical. In the worst case, a path with an arbitrarily large slack could be counted, as long as the slack of every edge in this path is no more than ϵ. In the extreme case, the slack of a path with n edges could be as large as $n\epsilon$.

4 Experimental Results

A placement with good (estimated) timing might not be routable. Thus a good timing-driven placement engine should also perform well in improving routability (or reducing total wirelength). We have implemented our new weighting algorithm in C language and integrated it in state-of-the-art timing-driven FPGA placement tool VPR[1]. Please note we targeted at FPGA placement applications because we had access to the tool. Our method can be applied to ASIC designs as well.

To guarantee a fair comparison, we have downloaded the source code of VPR 4.3 from [13], and only modified the net weighting part. Together with the source code of VPR, we also downloaded the architecture file and the 20 largest MCNC benchmark circuits used by VPR. This experiment is done on a UltraSparcII workstation.

During placement, VPR will simultaneously minimize the longest path delay and total bounding box wirelength. We set the balance factor between these two objectives to be 0.5, i.e., roughly speaking, VPR will spend equal efforts in minimizing these two objectives. For all other algorithmic parameters, we have used the default values in VPR.

Considering the randomness present in the simulated annealing algorithm used in VPR, we ran each circuits ten times. The results are reported in Tables 4 and 5. Delay

circuit				PATH	
name	block	net	grid	delay	win
ex5p	1135	1072	33 x 33	93.20%	4
tseng	1221	1099	33 x 33	97.20%	4
apex4	1290	1271	36 x 36	95.10%	7
misex3	1425	1411	38 x 38	90.20%	10
alu4	1544	1536	40 x 40	85.50%	10
diffeq	1600	1561	39 x 39	109.20%	0
dsip	1796	1599	54 x 54	61.30%	10
seq	1826	1791	42 x 42	76.40%	10
apex2	1919	1916	44 x 44	84.30%	9
s298	1941	1935	44 x 44	93.50%	5
des	2092	1847	63 x 63	84.70%	10
bigkey	2133	1936	54 x 54	64.60%	10
frisc	3692	3576	60 x 60	90.90%	6
spla	3752	3706	61 x 61	80.80%	10
elliptic	3849	3735	61 x 61	83.00%	10
ex1010	4618	4608	68 x 68	76.60%	10
pdc	4631	4591	68 x 68	74.10%	10
s38417	6541	6435	81 x 81	93.30%	4
s38584.1	6789	6485	81 x 81	92.90%	3
clma	8527	8445	92 x 92	76.90%	10
ave*				84.40%	7.6

Table 4: Longest path delay comparison of VPR and PATH. Average value for performance gain is computed as geometric mean.

results reported in Table 4 for each algorithm are the best among ten runs, scaled by the best results obtained by original VPR. For each circuit, total wirelength results among these ten runs have very little difference, thus we report the geometric mean of ten runs in Table 5.

On average, our new weighting algorithm, namely PATH, achieved a delay reduction of 15.6%, while the cost of total wirelength increase was only 4.1%. It is worth pointing out that this new weighting algorithm is very robust in terms of performance optimization. In Table 4 we have also shown under column "win" how many times we can achieve better performance than the best results of VPR. On average, 76% runs of PATH can outperform the best results of original VPR. Another point we need to make is that implementing this new weighting algorithm does not add to the overall time complexity of VPR at all, thus the performance gains achieved here are essentially *free* of run time overhead.

5 Conclusions and Future Work

We have presented a new weighting algorithm based on path counting, which is the first known accurate all-path counting algorithm. Experimental comparison with an existing net weighting algorithm used in state-of-the-art timing-driven placement package VPR[1] has shown very favorable results.

We also experimented the weighting algorithms used in [10] and [12]. Unfortunately the results are worse than those of original VPR in our experiments. Since both [10] and [12] are based on a quadratic placement framework, we conjecture that those weighting algorithms might only be *good* in certain context.

6 Acknowledgment

The author gratefully acknowledges Prof. Jason Cong of UCLA VLSI CAD LAB and the anonymous reviewers for their valuable comments.

circuit	VPR	PATH	ratio
ex5p	181.1	181.4	100.2%
tseng	102.6	101.4	98.8%
apex4	196.3	200.7	102.2%
misex3	199.2	209.2	105.0%
alu4	203.8	209.9	103.0%
diffeq	159.0	155.4	97.7%
dsip	197.7	233.3	118.0%
seq	261.1	280.5	107.4%
apex2	280.8	291.0	103.6%
s298	228.3	225.8	98.9%
des	252.7	263.8	104.4%
bigkey	213.1	231.0	108.4%
frisc	589.1	573.1	97.3%
spla	636.4	686.3	107.8%
elliptic	511.4	527.6	103.2%
ex1010	676.7	688.3	101.7%
pdc	941.2	1027.6	109.2%
s38417	688.4	746.7	108.5%
s38584.1	676.9	689.9	101.9%
clma	1507.2	1600.4	106.2%
geo-mean	337.1	350.9	104.1%

Table 5: Wirelength comparison of VPR and PATH.

References

[1] A. Marquardt, V. Betz, and J. Rose, "Timing-driven placement for FPGAs," in *ACM Symposium on FPGAs*, pp. 203–213, 2000.

[2] T. Hamada, C. K. Cheng, and P. M. Chau, "Prime: a timing-driven placement tool using a piecewise linear resistive network approach," in *Proc. ACM/IEEE Design Automation Conference*, pp. 531–536, 1993.

[3] A. Srinivasan, K. Chaudhary, and E. S. Kuh, "RITUAL: Performance driven placement algorithm for small cell ICs," in *IEEE/ACM International Conference on Computer-Aided Design*, pp. 48–51, 1991.

[4] M. Jackson and E. S. Kuh, "Performance-driven placement of cell based IC's," in *Proc. ACM/IEEE Design Automation Conference*, pp. 370–375, 1989.

[5] T. Gao, P. M. Vaidya, and C. L. Liu, "A performance driven macro-cell placement algorithm," in *Proc. ACM/IEEE Design Automation Conference*, pp. 147–152, 1992.

[6] M. Burstein and M. N. Youssef, "Timing influenced layout design," in *Proc. ACM/IEEE Design Automation Conference*, pp. 124–130, 1985.

[7] M. Marek-Sadowska and S. P. Lin, "Timing driven placement," in *IEEE/ACM International Conference on Computer-Aided Design*, pp. 94–97, 1989.

[8] B. M. Riess and G. G. Ettelt, "Speed: fast and efficient timing driven placement," in *International Symposium on Circuits and Systems*, pp. 377–380, 1995.

[9] R. S. Tsay and J. Koehl, "An analytic net weighting approach for performance optimization in circuit placement," in *Proc. ACM/IEEE Design Automation Conference*, pp. 620–625, 1991.

[10] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," in *Proc. ACM/IEEE Design Automation Conference*, pp. 269–274, 1998.

[11] A. Marquardt, V. Betz, and J. Rose, "Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density," in *ACM Symposium on FPGAs*, pp. 37–46, 1999.

[12] M. Senn, U. Seidl, and F. Johannes, "High quality deterministic timing driven FPGA placement," in *ACM Symposium on FPGAs*, 2002.

[13] http://www.eecg.toronto.edu/~vaughn/challenge/challenge.html, "The FPGA place-and-route challenge."