Fast Wire Length Estimation by Net Bundling for Block Placement

Tan Yan Hiroshi Murata

Faculty of Environmental Engineering
The University of Kitakyushu
Kitakyushu, Fukuoka 808-0135, Japan
{yantan, hmurata}@env.kitakyu-u.ac.jp

ABSTRACT

The wire length estimation is the bottleneck of packing based block placers. To cope with this problem, we present a fast wire length estimation method in this paper. The key idea is to bundle the 2-pin nets between block pairs, and measure the wire length bundle by bundle, instead of net by net. Previous bundling method [5] introduces a huge error which compromises the performance. We present an error-free bundling approach which utilizes the piecewise linear wire length function of a pair of blocks. With the function implemented into a lookup table, the wire length can be computed promptly and precisely by binary search. Furthermore, we show that 3-pin nets can also be bundled, resulting in a further speedup. The effectiveness of our method is verified by experiments.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids—*Placement and Routing*; J.6 [**Computer Applications**]: Computer-Aided Engineering—*Computer-aided design (CAD)*

General Terms

Algorithm, Experimentation

Keywords

Wire length estimation, Net bundling, Lookup table

1. INTRODUCTION

To cope with the continuously increasing system size, hierarchical design methodologies are desired and the supporting block placement algorithms are studied. The most extensively studied optimization scheme for the block placement problem is the packing based one. In this scheme, many strategies have been proposed for area optimization [6, 9, 10, 16] while little work is focused on the wire length. Recent research shows that the wire length estimation dominates the runtime when wire length is taken into consideration [5]. This problem becomes even more crucial when dealing with larger circuits. Table 1 illustrates such a trend by showing the comparison between the partitioning results of the smallest (IBM01) and biggest (IBM18) ISPD'98 benchmarks [2] produced by hMetis [14]. It can be seen that the interconnection scale (avg. #pin per blk.) increases dramatically as the circuit size (#cells) enlarges. Therefore, how to restrain the increasing time cost of wire length estimation as the interconnections scale up becomes the major challenge for packing based block placers.

Table 1: The partitioning results of ISPD'98 benchmarks [2] and the profile of MCNC/GSRC benchmarks [15].

IBM01 (12506 cells, 14111 nets)				
#partitions	50	100	200	300
#nets (block level)	1899	2669	3259	3659
#pins (block level)	4819	7628	10327	12627
avg. #pin per blk.	96	76	52	42
IBM18 (210341 cells, 201920 nets)				
#partitions	50	100	200	300
#nets (block level)	15374	21650	27489	31416
#pins (block level)	36498	60158	85143	105723
avg. #pin per blk.	730	602	426	352
MCNC/GSRC benchmarks				
#blocks	49	100	200	300
#nets	408	885	1714	1893
#pins	953	1873	3640	4358
avg. #pin per blk.	19	19	18	15

A recent work [5] mentioned a net bundling method to speed up the wire length estimation. Although it could achieve a quite satisfying speedup, this naive bundling causes a huge error in the estimated wire length because it relocates the pins. This error leads to a regression in performance.

In this work, we propose an *error-free* bundling approach that speeds up the wire length estimation, especially when the interconnection scale is large. Our approach is based on the fact that the total HPWL of the nets spanning between two blocks forms a piecewise linear convex function with respect to the relative position of the two blocks [13, 17]. With this function implemented as a lookup table, we can compute the total wire length promptly and precisely by binary-searching the table instead of scanning the nets one by one. Furthermore, we show that 3-pin nets can be bundled together with 2-pin nets.

The contribution of this paper lies in the following aspects:

- The piecewise linear wire length function is usually exploited for wire length *optimization* in the single-row fixed-ordering cell placement problem [3,13,17]. When facing the order-free block placement problem in which optimization by this function becomes a hard task, researchers turn to the omnipotent simulated annealing (SA) and totally forget about this function. *We show that this function is still useful for wire length estimation in the SA context.*
- By the use of this function, the wire length estimation is greatly sped up, especially when the interconnections scale up.
- We are the first to show that 3-pin nets can be bundled.
- We also present an approximate bundling method as fast as that of [5] but with a much smaller error.

The rest of this paper is organized as follows: Section 2 gives the motivation of this paper. Section 3 outlines the naive bundling approach. Section 4 presents our lookup table based bundling approach. Section 5 analyzes the experimental results and Section 6 concludes this paper.

2. MOTIVATION

The block placement problem is to place a set of hard rectangular blocks without any overlap while optimizing the area and the wire length. One solution to this problem is the packing based method, in which the placement is encoded and the code is iteratively perturbed and evaluated in an SA scheme. During the evaluation, the code is first decoded to its corresponding placement and then the area cost and the wire length cost of this placement are computed.

In this work, we use the fixed-outline criterion [1,11] for area estimation. Under this criterion, the area cost is regarded as zero when all the blocks fit into the given outline. Otherwise, some penalty is charged for the exceeding part. This fixed-outline context eliminates the disturbance of area optimization when certain level of compactness is achieved and helps the placer focus on the wire length optimization.

Wire length is usually evaluated in several models, such as Half Perimeter Wire Length (HPWL), Minimum Spanning Tree (MST) and Rectilinear Steiner Minimal Tree (RSMT). Owing to its fast speed and high consistency with the RSMT model [12], HPWL becomes a standard estimation model for block placers. We also base our study on this model.

Since perturbing the code usually takes only $O(1)$ for most of the representations, *evaluation* time becomes the key. Area estimation can be done in $O(b)$ to $O(b \log b)$, depending on the representation used, where b denotes the number of blocks. HPWL estimation is possible in $O(p)$ time, where p denotes the total number of pins. Which one dominates the runtime depends on the average #pin per block (the value of p/b). In [5], it is observed that over 80% of the runtime is spent on wire length estimation because $p/b \gg 1$ in the tested benchmarks. Of course, the p/b value may vary between different design instances. In practice, it is determined by several methodological factors, such as the tractable size of intra-block implementation [18], the tractable number of blocks in the block placers [7], the number of levels in the hierarchical [1] or multi-level framework [8], etc. However, as can be seen from Table 1, the p/b value inevitably increases as the chip scale increases. *Furthermore, this table also suggests that the MCNC/GSRC [15] benchmarks, on which [5]'s work is based, greatly underestimate the p/b value for larger circuits (such as IBM18).* This means that wire length estimation for large designs may take far more than 80% of the runtime. Therefore, the runtime issue of wire length estimation in block placers is a critical problem for large circuits.

3. THE NAIVE BUNDLING

A naive solution to speed up the wire length estimation is to replace a set of nets by a single heavy net. Following methods fall into this category.

3.1 Bundling by Block Center

In [5], a technique called "conglomerate parallel 2-pin nets into a heavily weighted net" is introduced to reduce the number of 2-pin nets for its embedded placer. Here, the "parallel 2-pin nets" means the 2-pin nets that connect one pair of blocks. Before packing, all the 2-pin nets connecting a pair of blocks are replaced with one representative net with the weight same as the number of the nets bundled. The two pins of this heavy net is assumed to be located at the center of each block.

During the estimation, the weighted wire length of the representative net is used to estimate the total wire length of all the bundled nets. (Multi-pin nets are estimated in the conventional net-by-net fashion.) Fig. 1 gives an illustration of this approach. Due to our observation that usually over

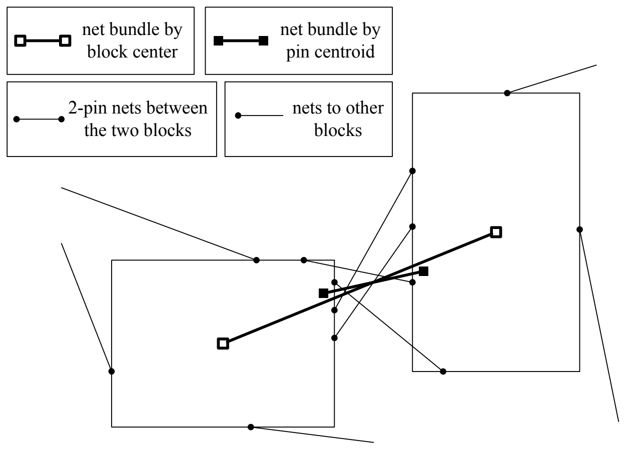

net bundle by block center
net bundle by pin centroid
2-pin nets between the two blocks
nets to other blocks

Figure 1: Four 2-pin nets connecting a pair of blocks are bundled into one net with a weight of four.

80% of the nets are 2-pin nets in industrial and academic benchmarks (refer to Table 2) and that many 2-pin nets are "parallel", this approach achieves a quite satisfying speedup.

Table 2: The percentage of 2/3-pin nets in MCNC/GSRC benchmarks.

nets	*ami33*	*ami49*	*n100*	*n200b*	*n300*
2-pin	84.6%	82.4%	88.9%	88.4%	72.4%
2-pin + 3-pin	92.7%	95.1%	99.4%	99.3%	97.5%

However, since the pins are relocated in this approach, it is obvious that this block center bundling (referred as BC-bundling hereafter) has the following drawbacks:

- The estimation is no longer exact, in the sense that it does not measure the true sum of the wire length of the nets in a bundle because the distribution of the pins is ignored. The magnitude of the error depends on the problem instances. In our experiments, a huge error (41.6% at maximum) is observed for *ami33*.

• This approach cannot sense the change in wire length when a block rotates or flips because the block center does not move. This may mislead to wrong rotation or flipping of abutting blocks. An example is shown in the Fig. 2.

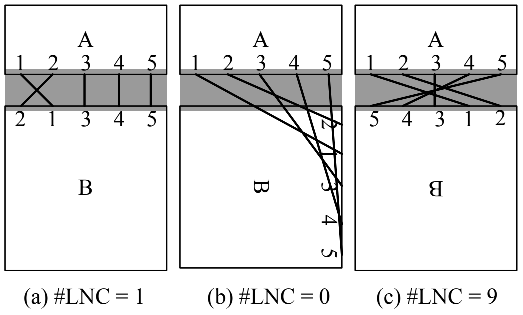


Figure 2: Inexact bundling may lead to bad rotation (b) or flipping (c).

3.2 Bundling by Pin Centroid

We can enhance BC-bundling by a simple modification. Instead of the block center, we use the centroid of the pins of the bundled 2-pin nets as the pin location of the representative net (see Fig. 1). Considering that the pins of parallel nets usually stay very close to each other in a block, such a modification greatly reduces the error. This pin centroid bundling (referred as PC-bundling hereafter) could also prevent the occurrence of bad block rotation because it can often sense the change in wire length when a block is rotated. However, ignoring the detailed pin locations will still lead to bad block flipping and therefore, the number of Local Net Crosses (referred as LNC hereafter), which are the crosses of the nets spanning between adjacent boundaries of abutting blocks (the shaded area in Fig. 2 shows the adjacent boundaries), will eventually increase. Since the LNC number models the routing congestion of the channels between blocks, its increase will result in an increase in the routing space which is definitely not welcome.

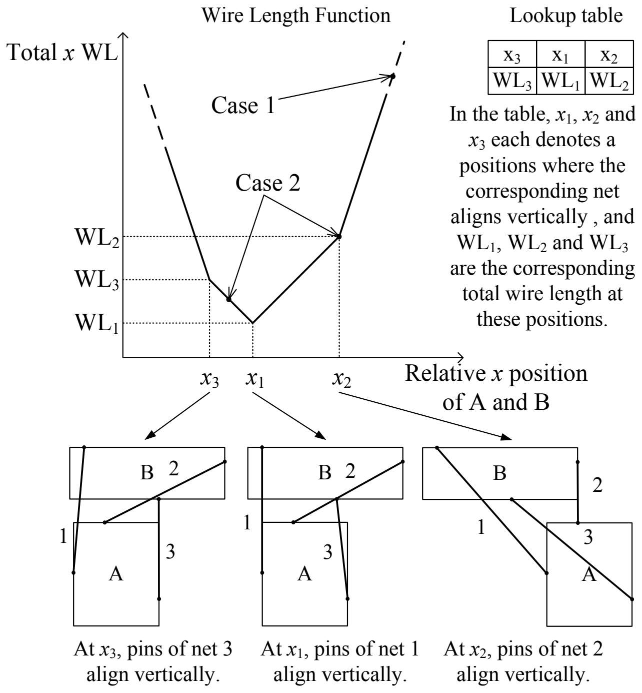
Fig. 2 gives an example of good (sub-figure (a)) and bad (sub-figure (b) and (c)) rotation/flipping of blocks. The wrong rotation of block B in (b) results in an increase in wire length. The wrong flipping of B in (c) results in an increase in the LNC number. BC-bundling could not sense the difference between (a) and (b), (c). Therefore, it may lead to unpreferred placements such as (b) or (c). Interestingly, the LNC number of BC-bundling might be even smaller than the precise estimation because wrong rotation may result in less local nets between adjacent boundaries (there is no net between the adjacent boundaries of A and B in (b) and therefore, the LNC number becomes zero). However, this advantage is obtained by the sacrifice of wire length which is more important. PC-bundling could sense the difference between (a) and (b) but becomes blind when facing (a) and (c). Therefore, it risks a larger LNC number.

There might be other enhancements on this naive bundling. However, as far as they replace multiple nets by one, they will inevitably introduce error that degrades the performance.

4. OUR PRECISE BUNDLING

In this section, we present our error-free bundling approach. In HPWL computation, x span and y span can be calculated separately. Here we only show how to estimate the wire length in the x direction (referred as x wire length hereafter). The y wire length can be estimated similarly.

An interesting fact is that the total x wire length of all the 2-pin nets between a pair of blocks is a piecewise linear convex function with respect to the relative x-position of the two blocks [13, 17]. An example can be found in Fig. 3. Each indifferentiable point (which we call "bend") in this function corresponds to a position where the two pins of a net align vertically.


Wire Length Function
Lookup table
Total x WL
Case 1
Case 2
WL_2
WL_3
WL_1
x_3 x_1 x_2
Relative x position of A and B
B 2
1
3
A
At x_3, pins of net 3 align vertically.
At x_1, pins of net 1 align vertically.
At x_2, pins of net 2 align vertically.

x_3	x_1	x_2
WL_3	WL_1	WL_2

In the table, x_1, x_2 and x_3 each denotes a positions where the corresponding net aligns vertically , and WL_1, WL_2 and WL_3 are the corresponding total wire length at these positions.

Figure 3: The relationship between the x wire length and the relative position of the two blocks is a piecewise linear convex function.

This fact is well exploited for wire length *optimization* of one dimensional single-row cell placement with fixed ordering [3, 13, 17]. However, when it comes to the more complex two dimensional block placement, researchers find it no longer possible to use this function for optimization because the order of the blocks is not fixed. They then turn to the omnipotent SA and totally forget about this function. In the following, we will show that this function is still useful in the annealing context. It can be used for wire length *estimation* instead of optimization.

4.1 Bundling by Lookup Tables

The basic idea of our lookup table based bundling (referred as LT-bundling hereafter) is to record the position x_i of each bend and its corresponding x wire length WL_i into a lookup table. With the table sorted by the position, the x wire length can be looked up promptly by binary search. Fig. 3 shows the data structure of the lookup table. In this part, we only focus on 2-pin nets and in the later part we will show how to bundle 3-pin nets. Nets with more than 3 pins are estimated in the conventional net-by-net style.

Before the annealing process, one lookup table is built for each pair of blocks. At first, empty tables are built for the block pairs. Then, we scan through all the 2-pin nets. For each net, we arrange its two connected blocks in such a way that the two pins of this net align vertically. The relative position of the two blocks and the corresponding total x wire length of all the 2-pin nets between the two blocks are then computed and inserted into the table for this pair of blocks. To keep the table ordered, some technique similar to insertion sort is used.

During the annealing process, the total x wire length can be estimated by the use of these lookup tables. Instead of scanning through the nets, we examine the connected block pairs one by one. For each pair of blocks, say A and B, the absolute rotation/flipping/position are transformed into relative ones. The relative x-position x_{AB} is then looked up in the table and the x wire length WL_{AB} can be computed by linear projection or interpolation as follows (also see Fig. 3):

Case 1: x_{AB} is too large (small) to be covered by the table. In this situation, the slope of its located segment in the function is the same as (the negative of) the number of 2-pin nets connecting the two blocks. Thus WL_{AB} can be projected from the last (first) entry of the table.

Case 2: x_{AB} is located between two entries ($x_i \le x_{AB} \le x_{i+1}$). In this situation, a linear interpolation between WL_i and WL_{i+1} is performed to obtain WL_{AB}.
Since linear projection and interpolation of a linear segment are exact, the resultant x wire length is also exact. Moreover, since the piecewise linearity of this function still holds when the 2-pin nets are weighted, our approach can be easily extended to consider net weights.

4.2 Space & Time Analysis

For each pair of connected blocks, totally 8 lookup tables are built as follows: when one block is fixed, the other block has 8 different relative orientation to it (4 relative rotation multiplied by 2 relative flipping) and for each orientation, two separate tables are built for x and y direction respectively. This leads to 16 tables. However, since vertically flipping a module does not change the x coordinate of the pins, the orientations before and after vertical flipping can share the same x table. Similarly, the orientations before and after horizontal flipping can share the same y table. Therefore, the number of the tables can be cut in half, only 8 tables are needed. For each 2-pin net connecting this pair of blocks, one entry is created in each table. Therefore, for a circuit with b blocks and n 2-pin nets, at most $8(b(b-1)/2)$ lookup tables with totally $8n$ entries are built. A lookup table contains two pointers to the two blocks, one pointer to the head of entry list and one integer storing the length of the entry list. Each entry contains two floats, one for the relative position and the other for the wire length. Therefore, the memory required for the lookup tables is $(4b(b-1) \times 4 + 8n \times 2) \times 4$ bytes assuming pointer, float and integer each consumes 4 bytes. It can easily fit into 71MB memory considering that b is less than 1000 and n is less than $100,000$ in practical cases. In our experiments, less than 50MB memory is used.

The table can be built in $O(n\hat{p})$ time where n is the total number of 2-pin nets and $\hat{p}$ is the maximal number of 2-pin nets between a pair of blocks. This is because for each net, both computing the total wire length between the two blocks it connects and inserting the generated entry into the lookup table require $O(\hat{p})$ only. In our experiments, building the table takes less than 1% of the total runtime.

Since we use binary search, the time to obtain the total wire length between a pair of blocks is $O(\log p)$, where p is the number of 2-pin nets between them. Noticing that conventional estimation method scans through the nets one by one, its time complexity should be $O(p)$. To estimate all the nets, all the connected block pairs should be checked in our approach. Therefore the worst case complexity is $O(b^2 \log \hat{p})$ where b is the number of blocks and $\hat{p}$ is the maximal number of nets between a pair of blocks. However, the runtime for practical circuits is much shorter because their interconnecting structures are far from random. In other words, not every pair of blocks is connected and therefore it is not necessary to examine all of them. For example, only $O(b)$ pairs of blocks are connected in the grid, ring and tree structure. The time complexity is then $O(b \log \hat{p})$ for those structures. Our experiments also confirm such a difference.

4.3 Bundling the 3-pin Nets

For a 3-pin net with pins p_1, p_2 and p_3, we know its x wire length follows:

$$\begin{aligned} HPWL_x &= \max(x_{p_1}, x_{p_2}, x_{p_3}) - \min(x_{p_1}, x_{p_2}, x_{p_3}) \\ &= (|x_{p_1} - x_{p_2}| + |x_{p_2} - x_{p_3}| + |x_{p_3} - x_{p_1}|)/2 \end{aligned}$$

Its y wire length has the same property. In other words, the HPWL of a 3-pin net is equal to half of the sum of the Manhattan distances between every pair of pins in this net. To our best knowledge, there has been no literature on this equation although similar equation is discovered for cut number [4], which is a totally different metric to wire length. By this, we decompose a 3-pin net into three *virtual* 2-pin nets and bundle them together with those actual 2-pin nets. Before we build the lookup table, we scan through all the nets and decompose every 3-pin net into virtual 2-pin nets, each with a weight of 0.5. Then we build the lookup tables taking the net weight into consideration. When we compute the wire length for each entry in the lookup tables, we use the weighted sum of the wire length of all the actual and virtual 2-pin nets between the corresponding two blocks.

When a 3-pin net is decomposed into three virtual 2-pin nets, the number of pins doubles, making the size of the lookup table increase. However, this will cause no serious problem either in the memory consumption or the table lookup time. Since the pin number will be at most twice as before, the size of the lookup table will at most double. As for time complexity, although the number of pins increases, bundling the virtual nets with the actual nets makes the total runtime decrease in most cases. Experiments also show the effectiveness of such decomposition.

5. EXPERIMENTAL RESULTS

To demonstrate the effectiveness of our method, we compare our method with the conventional estimation approach and the previous bundling method [5] in the terms of runtime and performance. We build an SA-based fixed-outline block placer with these estimations embedded for comparison. The outline is set to be a square with the area 10% larger than the total area of all the blocks. When the width (height) of the placement exceeds the outline, a cost of the ratio of the exceeding part to the outline width (height) is charged as the punishment. Wire length is evaluated as the ratio of the average HPWL to the half perimeter of the outline. The cost function is shown in the following equations. Sequence-pair [16] is used for packing and Algorithm 1 in [19] is used for decoding the sequence-pair. The placer is implemented in C language and all the experiments are conducted on Sun Blade 1000.

$$\begin{aligned}
Cost_{area} &= \max(0, W_{placement} - W_{outline})/W_{outline} \\
&+ \max(0, H_{placement} - H_{outline})/H_{outline} \\
Cost_{WL} &= \frac{\sum_{i=1}^{n} HPWL_{net_i}/n}{W_{outline} + H_{outline}}, \quad n = \#net \qquad (1) \\
Cost_{total} &= 0.5 \times Cost_{area} + 0.5 \times Cost_{WL}
\end{aligned}$$

The efficiency of the bundling greatly depends on the circuit structure. Intense connections lead to great speedup while sparse connections make the bundling inefficient. We design four benchmarks: *rand, grid, ring* and *tree* to show such a trend. All the four benchmarks have 100 blocks and 10000 2-pin nets but their interconnection structures are different. The connections of *rand* are distributed randomly among the block pairs while the connections of *grid, ring* and *tree* form grid, ring and tree structures respectively. The average numbers of nets between a pair of blocks are 2, 56, 100 and 101 for *rand, grid, ring* and *tree* respectively, indicating that the bundling efficiency of *rand* is low and that of *tree* is high. This is agreed by the experimental result in Table 3, in which conventional estimation (CV) and LT-bundling (LT) are compared. We can see that for *rand*, less than 2 times speedup occurs while for *tree*, an impressive 20 times speedup takes place.

Table 3: Experimental result of our testset.

Runtime	*rand*	*grid*	*ring*	*tree*
CV	1711	1718	1716	1711
LT	1038	107	82	83

We then use the popular MCNC/GSRC [15] benchmark to verify the effectiveness of our proposed approaches. Since the GSRC benchmarks are for floorplanning and lack the information on pin location, we randomly distribute the pins to the peripheries of the blocks. *As can be seen from Table 1, the scale of the inter-block connections (#pin per blk.) in the MCNC/GSRC benchmarks is over 23 (=352/15) times smaller than the IBM18 benchmark. This means that the MCNC/GSRC benchmarks greatly underestimate the interconnection scale of large circuits.* Therefore, we replicate their nets by 3, 10 and 30 times to project the inter-block connections in small, medium and large scale circuits. When replicating a net, we move the pins by a small offset so that the new net does not overlap with the original one.

We embed the conventional estimation, BC-bundling [5] as well as our proposed LT-bundling and PC-bundling into the *same* annealing scheme to estimate the 2-pin and 3-pin nets. Multi-pin nets are estimated in a net-by-net fashion. Four runs with different random seeds are performed for each case and the average runtime is shown by the first sub-column under each estimation method in Table 5. Runtime longer than 12 hours is not reported. The meaning of each column is explained in the footnote of the table. For BC-bundling, we only bundle the 2-pin nets following the description in [5]. In order to show the percentage of runtime spend on wire length estimation in each method, we use *gprof* under Solaris to profile the placer. Since the profiled placer runs very slowly, we use a shorter annealing schedule and perform only one run for each case. However, since we find that this percentage does not change a lot as the schedule and random seed change, we believe that the numbers in the table give a rough image on which procedure dominates the runtime. The second sub-column under each method shows the portion of time spent on estimating the bundled nets (the 2-pin nets for BC, PC-2 and LT-2, and the 2 and 3-pin nets for PC-3 and LT-3, there is no bundled net for CV). The third sub-column shows the percentage of time spent on estimating the unbundled nets in the net-by-net fashion. The time to build the lookup tables is less than 1% of the total runtime and is therefore not reported. The memory consumption of our placer is only 50MB for the largest *n300*30*. The solution quality of these methods is shown in the first three columns in Table 6 (under the label "Same Annealing Scheme"). Since both CV and LT-bundling are precise, they have the same result.

To show the error introduced in inexact estimations (BC, PC-2 and PC-3), we compare their results with those of precise estimation each time we call the estimator in the annealing process. The maximal and average error during a complete annealing process is shown in Table 4. Since adding such function makes the placer very slow, only one run is performed for each case. Here, we only present the error of the original MCNC/GSRC benchmarks. Those of the net-replicated benchmarks are quite similar.

Table 4: The maximal and average error (%) of inexact methods.

Bench	BC		PC-2		PC-3	
mark	max	avg	max	avg	max	avg
ami33	41.6	24.0	5.6	1.8	5.0	1.8
ami49	19.0	6.3	3.3	0.8	5.8	1.4
n100	44.8	33.9	28.9	18.9	32.0	20.6
n200	46.1	34.3	25.0	16.8	27.2	17.9
n300	14.5	10.0	6.4	3.6	21.7	13.4

Several interesting points can be observed from the result:

• BC-bundling has an obvious speedup over the conventional estimation method, especially when the interconnections scale up. However, its huge error (41.6% maximal and 24.0% average for *ami33*) results in a much longer wire length. For *ami33*, we observe a 33% ($= \frac{0.359-0.270}{0.270}$) increase in wire length which is intolerable for block placement.

• PC-2 bundling is as fast as BC-bundling but its error is much smaller for MCNC benchmarks. However, for GSRC benchmarks, the error is as significant as that of BC-bundling. This is because the pins are randomly distributed around the block boundary and lack adjacency. As a result, their centroid tends to be close to the block center. Therefore, PC-bundling's behavior resembles that of BC-bundling in certain degree for GSRC benchmarks.

• The effectiveness of bundling 3-pin nets is not obvious for original MCNC/GSRC benchmarks due to the small scale of interconnections. However, when the interconnections are replicated to simulate larger circuits, we can see an obvious speedup in both PC-bundling and LT-bundling.

• Smaller error in the bundling leads to shorter resultant wire length. The resultant wire length of PC-3 bundling is much shorter than that of BC-bundling and achieves the same level as that of precise estimations (CV and LT) for MCNC benchmarks. However, the number of local net crosses of its result is much larger than that of the exact estimations. For GSRC benchmarks, since PC-3 bundling's behavior is between CV and BC, its resultant wire length is also between CV and BC. Please note that although PC-bundling introduces a fairly large error for GSRC benchmarks, its ability to distinguish good and bad rotation of blocks helps it to achieve a shorter wire length than BC-bundling.

• Our LT-bundling fails to outperform the CV estimation for the original MCNC/GSRC benchmarks. This is because for small interconnection scale, the system overhead overwhelms the time saving of table lookup. However, as the interconnections scale up, the advantage of the LT-bundling becomes obvious. A 10 ($= \frac{21338}{2074}$) times speedup can be observed for *n100*30*. Moreover, since it is error-free, its behavior is the same as CV, outperforming those inexact bundling methods (BC, PC).

• The portion of time spent on wire length estimation is reduced when bundling is used (from 98.9% in CV to 70.4% ($= 52.1 + 18.3$) in LT-3 bundling for *n100*30*).

• As the interconnections increase, wire length estimation for multi-pin nets begins to dominate the runtime (from 14.7% in *ami49* to 81.9% in *ami49*30* for LT-3). Bundling 2/3-pin nets can hardly provide any further speedup.

We also design an experiment to examine the impact of the speedup on the quality improvement when the design time is constrained. We tune the annealing scheme of CV and LT-3 so that they run within the same time as PC-3. The result is concluded in the rightmost two columns of Table 6 (above the label "Same Runtime"). Here we also perform four runs for each case and take the average as the result. Since the runtime for *n300*30* and *n200*30* under CV is not available, the comparison is not made for those two cases. It can be seen that within the same runtime, LT-3 bundling still outperforms the even faster PC-3 bundling for most of the cases, especially when the design instance is large. The speedup of LT-3 over CV could be translated into a great improvement in the wire length in a time-constraint context. A 16.4% ($= \frac{0.182-0.152}{0.182}$) wire length improvement can be observed for *ami49*30*.

6. CONCLUSION

Recent research [5] shows that wire length estimation becomes the bottleneck in the packing based block placers. As the circuits keep sizing up, the increasing number of interconnections will make the problem even worse. Naive bundling by block center could speed up the wire length estimation but the introduced error compromises the performance. In this work, we propose an error-free bundling approach using the piecewise linear wire length function of two connected blocks. Our method achieves a significant speed up for large circuits while keeping the estimation exact. Moreover, we show that 3-pin nets can be bundled together with 2-pin nets by net decomposition. It is also possible to assign different weights to different nets in our method to consider issues such as timing constraint, etc. Due to its precise estimation result and its sensibility to wrong rotation and flipping of blocks, we suggest its usage in the final placement in which the rotation and flipping of blocks greatly affects the difficulty of the later routing.

We also present a pin centroid bundling method which achieves the same speed as previous block center bundling [5] but with a much smaller error. Its resultant wire length is comparable to that of precise estimation although the number of local net crosses of the resultant placement increases. This method is suitable for early stage floorplanning in which the flipping of blocks is less important.

7. ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewer for the idea on cutting the size of the lookup tables by half.

8. REFERENCES

[1] S. N. Adya and I. L. Markov. Fixed-outline floorplanning: Enabling hierarchical design. *IEEE Trans. on VLSI Systems*, 1(11):1120–1135, December 2003.

[2] C. J. Alpert. The ISPD98 circuit benchmark suite. In *Proc. ISPD'98*, pages 80–85. ACM, April 1998.

[3] U. Brenner and J. Vygen. Faster optimal single-row placement with fixed ordering. In *Proc. DATE'00*, pages 117–121. ACM/IEEE, March 2000.

[4] A. E. Caldwell, A. B. Kahng, and I. L. Markov. Optimal partitioners and end-case placers for standard-cell layout. *IEEE Trans. on CAD of Integrated Circuits*, 19(11):1304–1314, November 2000.

[5] H. H. Chan, S. N. Adya, and I. L. Markov. Are floorplan representations important in digital design? In *Proc. ISPD'05*, pages 129–136. ACM, April 2005.

[6] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu. B*-trees: a new representation for non-slicing floorplans. In *Proc. DAC'00*, pages 458–463. ACM/IEEE, June 2000.

[7] T.-C. Chen and Y.-W. Chang. Modern floorplanning based on fast simulated annealing. In *Proc. ISPD'05*, pages 104–112. ACM, April 2005.

[8] T.-C. Chen, Y.-W. Chang, and S.-C. Lin. IMF: Interconnect-driven multilevel floorplanning for large-scale building-module designs. In *Proc. ICCAD'05*, pages 159–164. ACM/IEEE, November 2005.

[9] P.-N. Guo, C.-K. Cheng, and T. Yoshimura. An O-tree representation of non-slicing floorplan. In *Proc. DAC'99*, pages 268–273. ACM/IEEE, June 1999.

[10] X. Hong, G. Huang, Y. Cai, J. Gu, S. Dong, C. K. Cheng, and J. Gu. Corner block list: an effective and efficient topological representation of non-slicing floorplan. In *Proc. ICCAD'00*, pages 8–12. ACM/IEEE, November 2000.

[11] A. B. Kahng. Classical floorplanning harmful? In *Proc. ISPD'00*, pages 207–213. ACM, April 2000.

[12] A. B. Kahng and S. Reda. A tale of two nets: Studies of wirelength progression in physical design. In *Proc. SLIP'06*, pages 17–24. ACM, March 2006.

[13] A. B. Kahng, P. Tucker, and A. Zelikovsky. Optimization of linear placements for wirelength minimization with free sites. In *Proc. ASP-DAC'99*, pages 241–244. ACM/IEEE, Janurary 1999.

[14] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. In *Proc. DAC'99*, pages 343–348. ACM/IEEE, June 1999.

[15] MCNC/GSRC Floorplan Benchmarks. http://www.cse.ucsc.edu/research/

surf/GSRC/progress.html.

[16] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. VLSI module placement based on rectangle-packing by the sequence pair. *IEEE Trans. on CAD*, 15(12):1518–1524, December 1996.

[17] M. Ohmura, S. Wakabayashi, J. Miyao, and N. Yoshida. Improvement of one dimensional module placement in VLSI layout design. *Electronics and Communications in Japan*, 74(7):67–76, July 1991.

[18] D. Sylvester and K. Keutzer. Getting to the bottom of deep submicron. In *Proc. ICCAD'98*, pages 203–211. ACM, November 1998.

[19] X. Tang, R. Tian, and D. F. Wong. Fast evaluation of sequence pair in block placement by longest common subsequence computation. In *Proc. DATE'00*, pages 106–111. ACM/IEEE, March 2000.

Table 5: Time comparison of the estimation methods under the same annealing scheme.

Bench	CV			BC			PC-2			PC-3			LT-2			LT-3		
mark	TT	BN	UN	TT	BN	UN	TT	BN	UN	TT	BN	UN	TT	BN	UN	TT	BN	UN
ami33	13	-	77.8	10	12.3	57.4	10	12.3	57.4	10	16.2	52.4	13	19.4	48.6	14	24.6	42.7
*ami33*3*	56	-	92.2	31	6.2	80.9	31	6.5	80.4	27	8.7	77.1	35	10.2	74.9	33	14.1	69.6
*ami33*10*	190	-	97.6	113	2.3	93.6	113	2.2	93.7	102	3.1	92.3	118	3.8	91.0	107	5.4	88.7
*ami33*30*	563	-	99.1	327	0.7	97.8	326	0.8	97.7	290	1.1	97.3	332	1.7	96.4	298	2.4	95.2
ami49	104	-	81.7	47	20.9	43.5	48	20.1	42.9	41	33.9	21.7	71	28.6	33.9	76	42.8	14.7
*ami49*3*	349	-	93.5	119	11.8	70.6	119	11.4	70.5	64	25.2	45.1	148	17.7	61.7	105	34.4	34.2
*ami49*10*	1124	-	97.9	373	4.4	89.4	374	4.4	89.0	179	12.5	73.9	403	7.3	84.3	225	19.4	63.4
*ami49*30*	3393	-	99.2	1240	2.7	95.1	1242	2.7	94.9	561	7.0	87.8	1332	4.0	93.0	661	11.0	81.9
n100	759	-	78.0	452	39.7	21.5	453	40.1	21.3	467	58.1	1.5	805	47.4	14.6	952	60.5	0.9
*n100*3*	2071	-	91.6	708	28.3	45.3	707	28.3	45.3	484	57.4	4.1	1112	37.8	33.2	1040	60.0	2.4
*n100*10*	6618	-	97.1	1521	13.8	73.4	1519	13.8	73.3	548	52.2	12.9	1987	23.0	59.4	1198	56.7	6.9
*n100*30*	21338	-	98.9	5488	9.5	85.8	5467	9.6	85.8	1001	43.9	31.7	7096	15.6	77.0	2074	52.1	18.3
n200	2168	-	72.4	1441	35.3	20.1	1446	35.1	20.1	1484	53.2	1.5	2376	43.3	14.2	2802	56.5	1.0
*n200*3*	5591	-	88.3	2091	25.4	42.7	2088	25.6	42.6	1544	51.3	4.5	3151	34.7	32.0	3034	55.8	2.8
*n200*10*	18103	-	95.9	6261	16.4	69.1	6275	16.4	69.2	2075	46.8	15.5	8659	23.7	59.3	3929	52.4	10.1
*n200*30*	>12h	-	-	20531	8.8	85.4	20412	8.7	85.5	3621	40.4	33.4	25938	14.3	78.0	7063	49.4	20.9
n300	6610	-	63.5	5488	23.1	30.6	5484	23.3	30.6	5448	46.4	3.8	7682	30.2	24.8	9833	51.8	2.5
*n300*3*	16201	-	83.3	9228	14.5	56.7	9222	14.4	56.7	5957	43.5	10.5	11698	20.8	49.2	10881	50.3	7.0
*n300*10*	52024	-	94.1	27930	8.5	79.1	28288	8.3	79.2	10489	37.4	29.4	33826	12.4	74.4	17328	45.0	21.8
*n300*30*	>12h	-	-	>12h	-	-	>12h	-	-	25118	27.7	53.6	>12h	-	-	41796	39.0	41.0

[a]CV means ConVentional net-by-net estimation, BC means Block Center bundling, PC means Pin Centroid bundling and LT means our proposed Lookup Table bundling. 2 means only 2-pin nets are bundled and 3 means 2 and 3- pin nets are bundled.
[b]TT means the Total runTime (in sec.), BN means the percentage of time spent on estimating the Bundled Nets and UN denotes the percentage of time spent on estimating the Unbundled Nets.

Table 6: Performance comparison of the estimation methods.

	Same Annealing Scheme														
Bench	CV & LT			BC			PC-3			CV			LT-3		
mark	FR	WL	LNC	FR	WL	LNC	FR	WL	LNC	FR	WL	LNC	FR	WL	LNC
ami33	75%	0.270	27	100%	0.359	0	75%	0.263	46	0%	0.269	3	100%	0.284	47
*ami33*3*	100%	0.275	260	75%	0.331	5	100%	0.268	234	50%	0.274	7	50%	0.258	223
*ami33*10*	50%	0.275	2537	100%	0.340	178	100%	0.264	7793	100%	0.270	3507	50%	0.261	633
*ami33*30*	50%	0.270	2591	75%	0.332	27	75%	0.272	48720	50%	0.273	28328	50%	0.268	1190
ami49	50%	0.161	102	100%	0.182	39	75%	0.160	96	75%	0.169	59	100%	0.172	57
*ami49*3*	100%	0.161	539	75%	0.177	428	50%	0.158	722	25%	0.175	742	50%	0.161	704
*ami49*10*	50%	0.160	11030	75%	0.184	7169	25%	0.163	11169	50%	0.178	7496	50%	0.151	14537
*ami49*30*	75%	0.154	53126	100%	0.185	50812	100%	0.161	101758	0%	0.182	46897	50%	0.152	65327
n100	100%	0.368	2	100%	0.414	0	100%	0.375	5	100%	0.369	1	100%	0.375	3
*n100*3*	100%	0.368	60	100%	0.435	41	100%	0.380	24	100%	0.387	20	100%	0.379	35
*n100*10*	100%	0.371	311	100%	0.429	217	100%	0.380	471	100%	0.401	175	100%	0.378	263
*n100*30*	100%	0.377	1378	100%	0.424	985	100%	0.389	762	25%	0.417	403	100%	0.381	898
n200	100%	0.360	1	100%	0.408	1	100%	0.368	3	100%	0.361	1	100%	0.363	1
*n200*3*	100%	0.354	32	100%	0.415	16	100%	0.366	25	100%	0.372	29	100%	0.362	50
*n200*10*	100%	0.359	322	100%	0.413	303	100%	0.369	336	0%	0.388	158	100%	0.368	259
*n200*30*	100%	0.364	1391	100%	0.415	4148	100%	0.376	788	-	-	-	100%	0.372	956
n300	100%	0.291	3	100%	0.312	1	100%	0.297	2	100%	0.293	2	100%	0.296	3
*n300*3*	100%	0.294	74	100%	0.312	35	100%	0.296	72	100%	0.302	70	100%	0.296	59
*n300*10*	100%	0.291	478	100%	0.310	381	100%	0.298	382	100%	0.311	358	100%	0.296	410
*n300*30*	100%	0.296	1001	-	-	-	100%	0.304	666	-	-	-	100%	0.300	1346
							Same Runtime								

[c]FR (Fit-in Rate) means the success rate of fitting the placement into the outline in the four runs. WL is the Wire Length cost in the form of Eq. (1). LNC means the number of Local Net Crosses.