Boosting: Min-Cut Placement with Improved Signal Delay

Andrew B. Kahng
CSE and ECE Departments
University of CA, San Diego
La Jolla, CA, 92093
abk@cs.ucsd.edu

Igor L. Markov
EECS Department
University of Michigan
Ann Arbor, MI, 48109
imarkov@eecs.umich.edu

Sherief Reda
CSE Department
University of CA, San Diego
La Jolla, CA, 92093
sreda@cs.ucsd.edu

ABSTRACT

In this work we improve top-down min-cut placers in the context of timing closure. Using the concept of *boosting factors*, we adjust net weights according to net spans, so as to reduce the quadratic wirelength. Our method is generic and does not involve any timing analysis during or prior to placement. In essence, we skew the netlength distribution produced by a min-cut placer so as to decrease the number of long nets, with minimal impact on the overall wirelength. Empirically this approach does not significantly affect runtime, but reduces the worst negative slack and total negative slack of industrial benchmarks by up to 70% compared to Capo [5] and a leading industrial placer.

1. INTRODUCTION

As VLSI technology advances, interconnect delays dominate gate delays and become the bottleneck for high-performance designs [10]. Since signal delay grows quadratically with net length [7], long nets are often responsible for critical paths. Force-directed placement addresses this effect by minimizing the sum of squared netlengths. On the other hand, minimization of total netlength to meet routing supply constraints remains the primary objective for most placement strategies. Recent work [5, 22, 1] shows that top-down min-cut placers[1] produce high-quality placements in terms of wirelength, and moreover have excellent scalability and reasonable runtime. While optimal buffer insertion can make net delay linear in netlength [13], it consumes additional power and requires that unused space be available in the right regions. Both limitations may affect the feasibility of optimal buffer insertion. Additionally, buffers themselves introduce non-trivial signal delays.

The fact that many analytical placers [6, 21] minimize the quadratic length objective has been historically viewed as a compromise between mathematical and computational convenience and the end goal of total wirelength minimization. The debate regarding the use of linear versus quadratic wirelength goes back to at least the mid-1980s [19, 16, 8, 11]. However, most comparisons in the literature do not include the evaluation of signal delay, and focus instead on routability and wirelength [16]. A common conclusion is that minimization of linear wirelength is preferable since the objective better models routed wirelength and thus the demand for routing resources. Nevertheless, recent studies [20] confirm that quadratic placers produce placements with better timing than min-cut placers. Hence, a modern placer must minimize linear wirelength while at the same time attempting to reduce the quadratic wirelength measure for better timing. To this end, recent work on timing-driven placement proposes limiting netlengths in several ways [18, 15, 14, 9, 10].

In this paper, we modify modify net weights during min-cut placement to decrease the number of large nets. Observe that some previous approaches tend to do the opposite, e.g., by linearizing the quadratic objective in analytical placement so as to minimize linear wirelength [16]. In contrast, we seek to reduce signal delay. We introduce *boosting factors* that increase the weights of nets which are likely to be long. These adjustments are computed dynamically based on lower bounds of nets during min-cut placement. Through tie-breaking this reduces quadratic wirelength with little impact on linear wirelength. Empirical validation involves global and detailed routing after placement, followed by static timing analysis using Cadence Pearl. Our techniques do not affect runtime min-cut placement, but significantly improve circuit delay in resulting placements.

In our experiments we do not consider optimal buffer insertion that makes signal delay asymptotically linear in terms of net length [13]. Optimal buffer insertion is often infeasible because optimal buffer locations may already be occupied by existing cells. Additional buffers may increase routing congestion and power dissipation. Similar problems arise from logic replication.

The organization of this paper is as follows. Section 2 provides a brief overview of related work in the literature. Section 3 gives basic definitions and motivates our work. Section 4 introduces the concept of boosting, determines which nets should be boosted and discusses the effect of boosting on cut size. Empirical validation is covered in Section 5, and Section 6 summarizes our contributions.

2. PREVIOUS WORK

Several related techniques give important context to this work. Most timing-driven placement methods start by identifying nets that belong to critical paths. Various methods are then used to control the net lengths [12, 17, 18, 15, 6, 14, 9, 10]. Main differences are associated with whether the placement methodology is analytical [17, 18, 6] or top-down [12, 14, 9, 10]. Top-down methodologies use quadratic partitioners [14] or min-cut partitioners [12, 10]. Net length constraints are typically translated into net weights [12, 14] or handled as upper bounds [17, 9].

Ou and Pedram [14] control the number of cuts in a critical path, since the more cuts a path experiences, the longer the path tends to be. Hence, the method of [14] gives large weights

[1]From now on, we will refer to top-down min-cut placers as min-cut placers. It is not clear whether our proposed boosting technique improves results if min-cut is already combined with quadratic placement, as in [2]. However, unlike the addition of quadratic placement, boosting does not imply increased runtime.

to critical nets and imposes an upper bound on the maximum number of times a path can be cut. In [9], linear programming is used to constrain the bounding boxes of critical nets.

Marek-Sadowska and Lin [12] perform static timing analysis at each level of recursive bisection and then translate slacks to net weights. These weights affect the min-cut partitioner and eventually reduce signal delay. Another recent approach that uses a min-cut partitioner is due to Kahng et al. [10]. Given a block under partition, some of the block's cells are pre-assigned and fixed to the child partitions so as to reduce the negative slack of critical paths. After cell pre-assignment, the hypergraph partitioner is invoked on the remaining cells.

While we later show how our approach attempts to reduce the quadratic measure of placement produced from linear min-cut placers, it is interesting to note that historically attempts are made in the opposite direction, that is, to linearize quadratic programs [16, 8, 15]. This is primarily motivated by wirelength reduction and improvement of routability. In GORDIAN-L [16], edges are given weights to linearize the squared edge-lengths. These weights *decrease* as the edge lengths increase. In our work, by contrast, the net weights *increase* as the net lengths increase. Our motivation is primarily signal delay reduction since we expect good wirelength quality from min-cut placers. We also note that GORDIAN-L requires several iterations within each of the multiple steps of GORDIAN, while boosting has no runtime overhead. Furthermore, GORDIAN-L attempts to optimize wirelength only, whereas we are also attempting to optimize delay, while keeping wirelength low.

3. DEFINITIONS & MOTIVATING EXAMPLE

A circuit netlist is represented by a hypergraph $H(V, E)$, where V is the set of nodes corresponding to the circuit cells such that each node $v \in V$ has weight $w(v)$ reflecting its physical area. We refer to the horizontal location of v by v_x and the vertical location by v_y. Hyperedges $E \subseteq 2^V$ model circuit nets, where each hyperedge $e_i \in E$ is a set of nodes that are connected by a net. Associated with each placed hyperedge e_i is a bounding box $BB(e_i)$ whose corners are the four points (e_i^l, e_i^u), (e_i^l, e_i^b), (e_i^r, e_i^u) and (e_i^r, e_i^b), where the superscripts u, b, l and r refer to upper, bottom, left and right respectively. The length of a given net/hyperedge is the half-perimeter of the bounding box, sometimes denoted with HPWL (half-perimeter wirelength).

We think of a placement region as a collection of *blocks* as shown in Figure 1. Each block corresponds to a fixed rectangle into which some (sub)hypergraph vertices should be placed. Initially, the chip area is comprised of one block. The min-cut placement methodology proceeds by recursively partitioning each block and its associated hypergraph, and assigning the partitioned subhypergraphs to sub-blocks. Cut direction usually alternates between vertical and horizontal, or is determined the block aspect ratio [5]. Hence, the product of the partitioning process is a slicing floorplan as illustrated in Figure 1. In this figure, we illustrate two horizontal and vertical cut levels numbered from 1 to 2. The partitioning process continues until a certain block threshold size, beyond which end-case placers are used to assign actual locations for the hypergraph nodes in their corresponding blocks [4, 5, 22]. For each block $\mathcal{B}_j$, we let $BB(\mathcal{B}_j)$ represent the bounding box of block $\mathcal{B}_j$, where the bounding box of a block $BB(\mathcal{B}_j)$ is the rectangle whose corners are the corners of the block.

Minimizing wirelength has been the most traditional placement objective, yet the total delay is proportional to the sum of the squared net lengths. Consequently, it would be a boost for

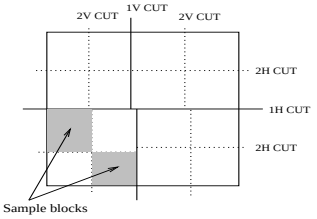

Figure 1: A slicing floorplan produced by a top-down min-cut placer. Two rounds of vertical and horizontal cuts are illustrated. 1H CUT indicates the first horizontal cut level, while 2H CUT indicates the second horizontal cut level. 1V CUT and 2V CUTS are for vertical cuts.

performance to minimize the quadratic wirelength measure, i.e., the sum of the net squared lengths, while trying to minimize the wirelength. Hence, the placer should also minimize $\mathcal{L}_2 = \sum_{i=1}^{|E|} l_i^2$, where l_i is the Half-Perimeter Wirelength (HPWL) of the i^{th} hyperedge e_i, alongside the traditional objective of $\mathcal{L}_1 = \sum_{i=1}^{|E|} l_i$.

Since the squared length of "long" nets contributes heavily to $\mathcal{L}_2$, it is desirable to restrict *incremental* elongation of long nets since this is costly in terms of the squared distance metric. This restriction can reduce the total delay, as shown below.

Example 1: Assume as in Figure 2 that we have three hyperedges e_1, e_2 and e_3 and that we are currently partitioning the shaded blocks. We have two solutions: solution A and solution B. In both solutions, the total wirelength and cuts are both equal to 8. However, we prefer solution B since its total delay may be smaller. Solution A has 38 units of delay (in terms of squared length), while solution B has 24 units of delay.

The length of each net can be upper-bounded and lower-bounded at any point during top-down placement, based on which blocks contain incident cells ("incident blocks"). For convenience, we treat fixed cells, pins and pads as individual blocks. The half-perimeter (HPWL) of a net cannot exceed the common half-perimeter of all incident blocks. Related lower bounds can be defined separately in the horizontal and vertical dimensions as we now describe for the horizontal case. Indeed, the left-most cell on the net must be placed to the left from the right edge of every incident block; similarly, the right-most cell on the net must be placed to the right from the left edge of every incident block (note that we do not need to know which particular cells will eventually be left-most and right-most when the final placement is produced). To lower-bound the horizontal component of the net's half-perimeter, we need to consider right edges of

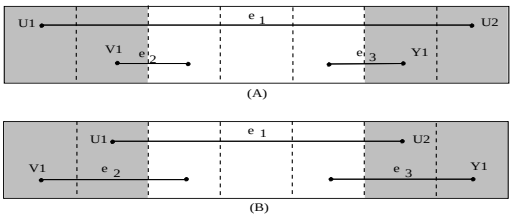

Figure 2: Two placements with equal linear wirelength but different squared wirelength, which is greater in the (A) example.

all incident blocks and find the left-most. Similarly, we find the right-most left edge over all incident blocks, and compute the distance between the two horizontal locations. The net's span cannot be smaller than that. A lower bound for net's length is produced by adding the horizontal and vertical components.

Let us study the progression of lower and upper bounds for nets that are not incident to fixed cells, pins or pads. At the beginning of top-down placement the upper bound for every such net is the half-perimeter of the core area, and the lower bound is zero. At every cut, the upper bound decreases for every uncut net and does not change for every cut net. Similarly, the lower bound increases for every cut net, and does not change for every uncut net. Thus, in general, lower bounds gradually increase and upper bounds gradually decrease until they meet at the end of top-down placement, at which point their values are both equal to the net's actual length.

4. ACHIEVING SMALLER SIGNAL DELAY WITH MIN-CUT PLACERS

In this section we highlight one of the drawbacks of the min-cut placement methodology in the context of timing-driven placement, and then outline a potential solution.

4.1 Boosting Min-Cut Placement

Most netlists allow multiple placements with equal total wirelength, however in timing-driven placement it is important to prevent very long nets because net delay grows quadratically with net length. Traditional min-cut placers may end up producing several long nets so as to shorten medium-sized nets, and often leave such trade-offs up to chance. This suggests the possibility of useful tie-breaking that would not affect runtime or solution quality, but may address additional design objectives. Motivated by this, we propose a way to discourage very long nets by dynamically increasing the weight of each net after it has been cut, so as to prevent further cuts.

Below we assume a placer that partitions each block in approximately equal sub-blocks (this is typically true for sufficiently large blocks in the absence of significant design constraints).

Example 2: Let us examine the first two vertical cuts (and skip horizontal cuts if any), which may be at 50% and 75% of the chip width, as shown in Figure 3. If a hyperedge is cut twice, its length can be as high as the chip width. However, if it is cut only once, its length can be upper bounded by either 50% of the chip width (cut by the second cut) or by 75% of the chip's width (cut by the first cut). In this case, our proposed modification to the standard min-cut framework applies after the first cut. Namely, the weights of cut hyperedges are increased so as to discourage them from being cut again. This way we may achieve a 75% upper bound for the net length of many such hyperedges, while the length of each previously uncut hyperedge is already subject to a 50% upper bound.

In our example, we can also track lower bounds for net lengths. Indeed, two cuts imply a 25% lower bound, while any one cut implies only a 0% lower bound. Given how *upper* bounds change, it appears that increasing the weights of cut hyperedges may prevent very long nets. Furthermore, given how *lower* bounds change, it appears that if we do not prevent multiple cuts early in top-down placement, it may be difficult to prevent long nets.

In practical terms, we multiply the weight of the hyperedge e_2 by the factor $\beta = 2$ as shown in Figure 3 (b). This strengthens the connection between the nodes of hyperedge e_2 encouraging the partitioner to move node v to the left shaded sub-block and consequently improving the upper bound on the hyperedge's length by a quarter of the chip width. To formalize the idea of increasing the weight of longer nets, we define *boosting* as follows.

Definition 1: A hyperedge $e_i \in E$ is *boosted* by multiplying e_i's weight by a certain factor, i.e., the *boosting factor* β, and a hyperedge is boosted *only if partitioning the current block can increase the lower bound on the hyperedge span (HPWL).*

We now propose *eligibility* conditions for boosting a hyperedge, i.e., conditions where partitioning can decrease an upper bound on net length. Given a block $\mathcal{B}_j$ and a corresponding hypergraph $H(V, E)$ to be partitioned, a hyperedge $e_i \in E$ is *eligible* for boosting only if it meets all of the following conditions:

1. $BB(e_i) \not\subseteq BB(\mathcal{B}_j)$, i.e., e_i has been cut before.
2. There exists some node v of e_i in $BB(\mathcal{B}_j)$ such that
   If $\mathcal{B}_j$ is cut vertically:
   - either $v_x = e_i^l$ and no point of the line segment $[(e_i^r, e_i^b), (e_i^r, e_i^u)]$ is in $BB(\mathcal{B}_j)$.
   - or $v_x = e_i^r$ and no point of the line segment $[(e_i^l, e_i^b), (e_i^l, e_i^u)]$ is in $BB(\mathcal{B}_j)$.

   If $\mathcal{B}_j$ is cut horizontally:
   - either $v_y = e_i^u$ and no point of the line segment $[(e_i^r, e_i^b), (e_i^l, e_i^b)]$ is in $BB(\mathcal{B}_j)$.
   - or $v_y = e_i^b$ and and no point of the line segment $[(e_i^r, e_i^u), (e_i^l, e_i^u)]$ is in $BB(\mathcal{B}_j)$.

We illustrate these conditions by the following example.

Example 3: Figure 4 illustrates the eligibility for boosting in four sample cases:

- Case i: The hyperedge is eligible for boosting since partitioning the current (shaded) block can extend the lower bound on the hyperedge length.
- Case ii: The two hyperedges are not eligible for boosting since partitioning the current block will not affect on the lower bound of their length. Condition 2 is violated.
- Case iii: Not eligible for boosting since the position of the node contained within the current block does not affect the hyperedge span. Condition 2 is violated.
- Case iv: The hyperedge is eligible since partitioning the current block can increase the lower bound.

While Definition 1 gives eligibility conditions for boosting a hyperedge, this does not mean that we should necessarily boost the hyperedge. We introduce a further condition in order to optimize $\mathcal{L}_2$ (the quadratic wirelength) while keeping $\mathcal{L}_1$ (linear wirelength) small.

If we assume an alternating horizontal-vertical cut sequence and we examine the vertical cut sequences[2] then the first vertical cut bisects the chip width. If the chip width is W then the first vertical cut is at $\frac{W}{2}$, creating 2 blocks. The second-level vertical cuts are at $\frac{W}{4}$ and $\frac{3W}{4}$, creating 4 blocks, and the i^{th} level cuts create 2^i blocks each of width $\frac{W}{2^i}$. This process continues until all blocks reach a certain threshold.

If a hyperedge is cut at a certain cut level then nodes of this hyperedge exist on *both* sides of the cut. Hence, if a hyperedge is first cut at cut level l and and later at cut level $l + 1$

[2]Horizontal cuts are treated similarly.

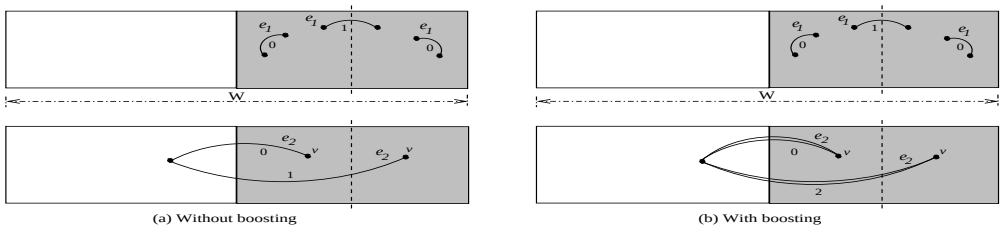

Figure 3: An illustration of boosting. The hyperedge e_2 was cut at the first vertical cut. The dashed vertical line represents the new second vertical cut. With respect to the new cut, each hyperedge can be cut or uncut as illustrated by the various positions of each hyperedge; we label each possibility by its contribution (0/1) to the cut value as well. We notice that without boosting the partitioner does not differentiate between the two hyperedges. However, if e_2 is cut then its length is upper-bounded by three quarters of the chip width. To encourage the partitioner to cut e_1 over e_2, we multiply the weight of e_2 by a factor of 2 as shown in picture (b).

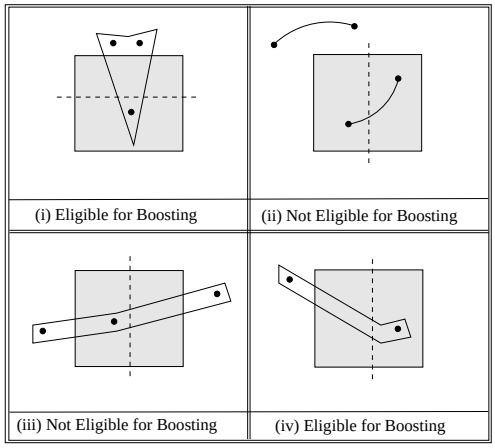

Figure 4: Illustrative cases for the eligibility of hyperedge boosting. The shaded block represents a block under partitioning by the vertical dashed line. A hyperedge is eligible for boosting if partitioning the block can increase the lower bound on its length.

then this increases the lower bound on its horizontal length by $W/2^{l+1}$. Suppose that a hyperedge was cut at levels l through $l+\tau$, where $\tau \geq 1$, then the lower bound on the hyperedge's length is $\sum_{i=l+1}^{l+\tau} W/2^i = \frac{W\cdot(2^\tau-1)}{2^{l+\tau}}$, where the contribution of level $l+\tau$ to the lower bound is $\frac{W}{2^{l+\tau}}$. This means that the contribution of new cuts is exponentially decreasing as we descent in the top-down hierarchy. Hence, there is little point in boosting after $\tau \geq 4$ since the total contributions of new cut levels (if the hyperedge ever gets cut again) will never exceed 7% of the hyperedge's horizontal length at $\tau = 4$. In practice, we never boost a hyperedge past level 8 (to account for 4 horizontal cuts and 4 vertical cuts) in min-cut placement since block sizes at that point are less than 0.39% of the chip size, and boosting will effectively hurt the cutsize without any prospect of significant reduction of the $\mathcal{L}_2$ objective.

4.2 Effect of Boosting on Cut Values

In the previous subsection, we determined the eligibility conditions for boosting a hyperedge. However, a crucial parameter is the value of the boosting factor β. A high value for β is expected to encourage a min-cut partitioner to move nodes so as to reduce the hyperedge's span. In the following, we analyze the relation between β and the cut size, which directly affects $\mathcal{L}_1$ [3].

Suppose that we have two nodes v and u as shown in Figure 5, where the rectangle represents the current block under partition. We say that a hyperedge is *external* if some of its nodes lie outside the current block under partition, and a hyperedge is *internal* if all of its nodes lie within the block under partition. Node v is connected to the sets E_1 and E_1 of external hyperedges eligible for boosting, where $|E_2| > |E_1|$, and to the sets of internal hyperedges sets I_1 and I_2. Node u is connected to only the internal hyperedges sets N_1 and N_2, where $|N_1| = |I_1|$ and $|N_2| = |I_2|$. Thus, the gain of moving node v from partition 1 to partition 2 is $\delta_v = |I_2| - |I_1| + |E_2| - |E_1|$, while the gain of moving node u is $\delta_u = |N_2| - |N_1|$. With boosting, the gain of node v becomes $\delta_v^\beta = |I_2| - |I_1| + \beta(|E_2| - |E_1|)$, while the gain of node u remains the same since it is not connected to any hyperedges eligible for boosting. We now distinguish three important cases for a min-cut partitioner operating with boosting:

- Case 1: $\delta_u = \delta_v$ but $\delta_v^\beta > \delta_u$. In this case the partitioner ends up moving node v rather than u. Hence, boosting does not worsen the cut size but at the same time the span of the long external hyperedges is reduced.
- Case 2: $\delta_u > \delta_v$ but $\delta_v^\beta < \delta_u$. Here the partitioner ends up moving node u rather than v as if it is not operating under boosting. Hence, boosting does not worsen the cut size but there is no benefit from boosting either.
- Case 3: $\delta_u > \delta_v$ and $\delta_v^\beta > \delta_u$. In this case the partitioner moves node v rather than u, effectively reducing the spans of the external hyperedges but worsening the cut size by $(|E_2| - |E_1|)(\beta - 1)$.

Under any scheme that boosts net weights only for small values of τ (the maximum amount of levels a hyperedge can be boosted),

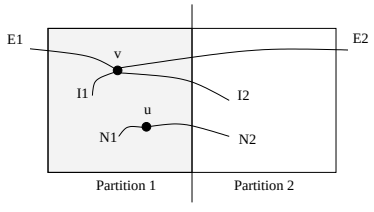

Figure 5: The effects of boosting on cut size. Node v is connected to the sets of hyperedges I_1, I_2, E_1, and E_2 (these are not individual hyperedges). Node u is connected to the two sets of hyperedge N_1 and N_2.

the likelihood of Case 3 is small. To see this, we observe that with $\beta = 2$, the difference between moving node v and u is $|E_2| - |E_1|$. However, during the first few levels in min-cut placement, the number of nodes within each block is essentially large while the min-cut values are practically low. Thus, the common case is that each node is connected to few boosted hyperedges, leading to a graceful reduction in the cut size.

5. EXPERIMENTAL VALIDATION

In this section we empirically assess how boosting affects circuit delay in terms of the worst negative slack and the total negative slack (TNS). We also report the sum of squared net lengths as well as wirelength distributions before and after boosting. Our implementation is based on Capo [5] (version 8.7) and is compared to (i) the original Capo, (ii) Cadence's QPlace place (version 5.2). After placement, we perform global and detailed routing using Cadence's WRoute (version 5.4) and evaluate circuit timing by means of static timing analysis using Cadence's Pearl (version 5.1). The four industrial benchmarks used for experiments are described in Table 1.

Since Capo is randomized, Table 2 reports the best results out of three independent runs. The alternate design flows in this table are as follows. In the flow **indust NTD**, the industrial placer is used to place the benchmark in a non-timing driven mode and then WRoute is used to route the benchmark in a timing-driven mode. In the flow **indust TD**, the industrial placer is used to place the benchmark in a timing-driven mode and then WRoute is used to route the benchmark in a timing-driven mode. In the flow **CAPO regular**, Capo is used to place the benchmark and then WRoute performs timing-driven routing on Capo's output placement. In the flow **CAPO BOOST**, we use a modified version of Capo and route the resulting placements using WRoute in timing-driven mode. In Table 2, we report a number of metrics: $\mathcal{L}_1$ is the Half-Perimeter Wirelength, $\mathcal{L}_2$ is the sum of squared net lengths, **SLACK PRE** is the worst negative slack as computed by pre-routing timing analysis, **time** is the placement time in seconds, **Wirelength** is the actual wirelength as reported by WRoute, **SLACK** is the worst negative slack as reported by the analysis using Pearl, **TNS** is the Total Negative Slack of all nets that have negative slack.

The data in Table 2 suggest that boosting improves circuit timing. For example, in Design A negative slack improves by about 53% over the industrial placer in timing-driven mode and 57% over Capo, and furthermore boosting reduces the TNS by about 76% over the industrial placer and 78% over Capo. Also, boosting improves the worst negative slack of Design B by 25% over the industrial placer and 21% over Capo. For Design C, none of the placers is able to exploit any advantage in their timing-driven mode. For Design D, boosting reduces the TNS by about 13% over Capo and the industrial placer in timing-driven mode. We have experienced a small number of routing violations with some test cases. These can be handled by logic transformations or manually.

We also examine net length distribution of placements produced by regular Capo and boosted Capo to further support our claims that boosting reduces the number of long nets. For designs A and B we normalize the half-perimeter of each net with respect to the half perimeter of the core area and then produce a histogram of net lengths. Each normalized net length is assigned to one of 10 equal bins, and results are shown in Table 3. These data confirm that in general boosting reduces the number of long nets, but increase the number of small nets. All in all, boosting alters the distribution of net lengths and tends to reduce the total quadratic length and circuit delay.

6. CONCLUSIONS

Our work improves improves min-cut placers in terms of signal delay. The novel technique we propose (boosting) is based on dynamically changing net weights during top-down placement. We first observe that the length of each net is subject to a series of increasing lower bounds that depend on when the net is cut. Therefore we increase the weights of nets with the highest lower bounds, so as to discourage cutting them in the future. This limits the further increase of lower bounds.

While the changes in weights do not significantly affect the resulting half-perimeter wirelength, they can be informally seen to decrease squared wirelength, and therefore generically improve circuit delay. To validate this empirically, we implement boosting within the well-known min-cut placer Capo, perform detailed routing, and evaluate timing with Cadence Pearl. As a result, we improve circuit timing in several industrial benchmarks, compared to both Capo and a leading industrial placer. It is surprising that such an improvement in timing can be achieved, across several real-world circuits, without giving timing constraints to the placer and with a slight runtime penalty. We believe that more can be done by accounting for timing constraints during placement and by performing timing analysis during placement, but this may imply longer runtimes and would require a more complex software infrastructure.

Compared to other means of improving circuit delay, such as buffer insertion, gate sizing and logic duplication, our technique requires no additional cells/area and is unlikely to increase overall power dissipation. Quantifying these comparisons is the subject of our future work.

7. REFERENCES

[1] S. N. Adya et al., "Benchmarking for Large-Scale Placement and Beyond", *Proc. ACM/IEEE Intl. Symp. Physical Design*, 2003, pp. 95-103.

[2] C. Alpert, G Nam and P. Villarubia, "Free Space Management for Cut-Based Placement", *Proc. IEEE Intl. Conf. Computer-Aided Design*, 2002, pp. 746-751.

[3] M. A. Breuer, "Min-Cut Placement", *Design Automation and Fault Tolerant Computing*, 1977, pp. 343-962, vol. 1(4).

[4] A. Caldwell, A. B. Kahng and I. Markov, "End-Case Placers for Standard-Cell Layout", *Proc. ACM/IEEE Intl. Symp. Physical Design*, 1999, pp. 90-96.

[5] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Can Recursive Bisection Alone Produce Routable Placements?", *Proc. ACM/IEEE Design Automation Conf.*, 2000, pp. 477-482.

[6] H. Eisenmann and F. M. Johannes, "Generic Global Placement and Floorplanning", *Proc. ACM/IEEE Design Automation Conf.*, 1998, pp. 269-274.

[7] W. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers", *J. of Applied Physics* 19 (1948), pp. 55-63.

[8] L. Hagen and A. B. Kahng, "Improving the Quadratic Objective Function in Module Placement", *Proc. IEEE Intl. ASIC Conf.*, 1992, pp. 171-174.

[9] B. Halpin, C. Y. Roger Chen and N. Sehgal, "Timing Driven Placement Using Physical Net Constraints", *Proc. ACM/IEEE Design Automation Conf.*, 2001, pp. 780-783.

[10] A. B. Kahng, S. Mantik and I. L. Markov, "Min-Max Placement for Large-Scale Timing Optimization", *Proc.*

benchmark	cells	nets	core region	whitespace	metal layers	clock period (ns)
A	21103	21230	2142.20 x 1969.40	13.5%	4	9.459
B	33917	39153	23157.00 x 12732.30	49.8%	4	30.962
C	9585	10398	8705.40 x 86968.90	29.7%	5	37.515
D	40347	42487	2483.00 x 2456.00	42.5%	4	27.166

Table 1: Benchmark characteristics.

benchmark	**flow**		$\mathcal{L}_1$	$\mathcal{L}_2$	**SLACK**	**Wire**	**SLACK**	**TNS**
	tool	mode	**(HPWL)**	(x10^7)	**PRE (ns)**	**length**	**(ns)**	**(ns)**
A	**indust**	NTD	2826832	135	-0.796	3491503	-0.660	28.107
		TD	2892279	131	-0.413	3570024	-0.368	12.022
	CAPO	regular	2702130	125	-0.547	3335483	-0.607	23.36
		boost	2758502	103	**-0.376**	3260184	**-0.342**	**5.107**
B	**indust**	NTD	7392711	514	-28.344	9080259	-31.793	68253.5
		TD	7352731	469	-28.520	9100552	-31.750	56618.6
	CAPO	regular	6654505	361	-22.548	8375939	-29.595	**44951.8**
		boost	6978647	315	**-18.12**	8711281	**-25.757**	49627.6
C	**indust**	NTD	579958	125	**-0.302**	839408	-1.882	1870.3
		TD	574717	127	-1.119	841585	-1.878	**1750.8**
	CAPO	regular	590833	118	-0.323	835962	**-1.875**	1810.6
		boost	629473	110	-0.819	872089	-1.878	1799.5
D	**indust**	NTD	3514553	1180	-5.226	4251419	-5.226	3016.4
		TD	3524751	1130	**-5.253**	4257518	**-5.264**	3270.7
	CAPO	regular	3165932	954	-5.322	4076119	-5.315	3178.3
		boost	3282640	826	-5.383	4182982	-5.363	**3083.6**

Table 2: Benchmark Results (average of 4 seeds). NTD is the industrial placer in a non-timing driven mode. TD is the industrial placer in timing-driven mode. In CAPO flows, regular is for unmodified Capo and boost is for Capo in boosted mode. $\mathcal{L}_1$ is Half-Perimeter WireLength. $\mathcal{L}_2$ is the sum of SQUARE of net HPWLs. SLACK PRE is the negative slack calculated before actual routing. SLACK is the negative slack calculated after routing. TNS is the total negative slack. Numerical values in bold indicate the best result of each respective benchmark.

ACM/IEEE Intl. Symp. Physical Design, 2002, pp. 143-148.

[11] J.-M. Li et al., "New Spectral Linear Placement and Clustering Approach", *Proc. ACM/IEEE Design Automation Conf.*, 1996, pp. 88-93.

[12] M. Marek-Sadowska and S.-P. Lin, "Timing Driven Placement", *Proc. IEEE Intl. Conf. Computer-Aided Design*, 1989, pp. 94-97.

[13] R. Otten and R. Brayton, "Planning For Performance", *Proc. ACM/IEEE Design Automation Conf.*, 1998, pp. 122-127.

[14] S. L. Ou and M. Pedram, "Timing-Driven Placement Based on Partitioning with Dynamic Cut-Net Control", *Proc. ACM/IEEE Design Automation Conf.*, 2000, pp. 472-476.

[15] B. Riess and G. Ettelt, "SPEED: Fast and Efficient Timing Driven Placement", *Proc. IEEE Intl. Symp. Circuits and Systems*, 1995, pp. 377-380.

[16] G. Sigl, K. Doll and F. M. Johannes, "Analytical Placement: A Linear or a Quadratic Objective Function?", *Proc. ACM/IEEE Design Automation Conf.*, 1991, pp. 427-431.

[17] A. Srinivasan, K. Chaudhary and E. S. Kuh, "RITUAL: A Performance Driven Placement Algorithm for Small Cell ICs", *Proc. IEEE Intl. Conf. Computer-Aided Design*, 1991, pp. 48-51.

[18] R. Tsay and J. Koehl, "An Analytical Net Weighting Approach for Performance Optimization in Circuit Placement", *Proc. ACM/IEEE Design Autom. Conf.*, 1991, pp. 620-625.

[19] R. S. Tsay, E. S. Kuh and C. P. Hsu, "PROUD: A Sea-of-Gates Placement Algorithm", *IEEE Design & Test of Computers* (1988), pp. 44-56.

bin	Design A			Design B		
	regular	boost	change	regular	boost	change
1	19661	19578	-0.4%	35070	35002	-0.2%
2	889	1002	12.7%	2765	3077	11.2%
3	340	381	12.0%	844	710	-15.8%
4	195	150	-23.1%	262	186	-29.0%
5	60	40	-33.3%	130	141	8.4%
6	41	34	-17.1%	58	31	-46.6%
7	11	13	18%	18	1	-94.4%
8	3	2	-33.3%	2	2	0%
9	0	0	0.0%	3	2	-33.33%
10	0	0	0.0%	0	0	0%

Table 3: Wirelength distribution histogram for designs A and B for the two modes of Capo. We normalize the HPWL of individual nets with respect to the chip's half perimeter and assign each normalized net length to one of 10 equal bins.

[20] P. Villarrubia, "Important Considerations for Modern VLSI Chips", *Proc. ACM/IEEE Intl. Symp. Physical Design*, 2003, pp. 1-6.

[21] J. Vygen, "Algorithms for Large-Scale Flat Placement", *Proc. ACM/IEEE Design Autom. Conf.*, 1997, pp. 746-751.

[22] M. Wang, X. Yang and M. Sarrafzadeh, "DRAGON2000: Standard-Cell Placement Tool for Large Industry Circuits", *Proc. IEEE Intl. Conf. Comp.-Aided Des.*, 2001, pp. 260-263.