Min-Max Placement For Large-Scale Timing Optimization

Andrew B. Kahng♯, Stefanus Mantik† and Igor L. Markov‡
♯ UCSD CSE and ECE Depts., La Jolla, CA 92093-0114
† UCLA, Computer Science, Los Angeles, CA 90095-1596
‡ Univ. of Michigan, EECS Department, Ann Arbor, MI 48109-2122
abk@ucsd.edu, stefanus@cs.ucla.edu, imarkov@umich.edu

ABSTRACT

At the 250nm technology node, interconnect delays account for over 40% of worst delays [12]. Transition to 130nm and below increases this figure, and hence the relative importance of timing-driven placement for VLSI. Our work introduces a novel minimization of maximal path delay that improves upon previously known algorithms for timing-driven placement. Our placement algorithms have provable properties and are fast in practice. Empirical validation is based on extending a scalable min-cut placer with proven quality in wirelength- and congestion-driven placement [4]. The CPU overhead of the timing-driven capability is within 50%. We placed industrial circuits and evaluated the resulting layouts with a commercial static timing analyzer.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids – Layout, Place and Route

General Terms

Algorithms, Design

1. INTRODUCTION

Timing-driven layout has recently emerged as a major design bottleneck, highlighting the difficulty of finding a feasible layout of a circuit with prescribed cycle time and logic function. The significance of timing-driven layout increases with that of interconnect delays relative to device delays. While commercial placement engines can evaluate increasingly accurate measures of path timing, simple models often lead to more efficient minimization. To first order, total (average) net length objectives correlate with congestion- and delay-related objectives (since wirelength creates capacitative load and *RC* delay). To bring the topology of timing constraints closer to placement, some works [17, 6, 14] minimize delays along *explicitly* enumerated paths, which becomes impractical when the number of signal paths undergoes combinatorial explosion in large circuits.[1]

Combinatorial explosion is not a problem for static timing analysis methods [16, 1] which can quickly determine whether delays along implicitly defined paths satisfy given timing constraints. The key challenge in timing-driven global placement is to optimize large sets of path delays without explicitly enumerating them. This is typically done by interleaving weighted wirelength-driven placement with timing analysis that annotates individual cells, nets and timing edges with timing information [5]. Such annotations are translated into *edge* or *net weights* [21, 1, 25] for weighted wirelength-driven placement, or into additional constraints for such placement, e.g., per-net delay bounds in "delay budgeting" approaches [15, 23, 28, 20, 11]. Iterations are repeated until they bring no improvement.[2] As noted in [11, 26], "net re-weighting" algorithms are often *ad hoc*[3] and have poor convergence theory, i.e., if delays along critical nets decrease, other nets may become critical.[4] On the other hand, "delay budgeting" may overconstrain the placement problem and prevent good solutions from being found. A unification of budgeting and placement is proposed in [26], but finding scalable algorithms for such a unification remains an open problem.

While many published works focus on timing optimization alone, placement instances arising in the design of state-of-the-art electronics today are often difficult from the wirelength/congestion standpoint alone. Therefore, a robust placement algorithm must have a proven record in wirelength- and congestion-driven context without timing. Motivated by this circumstance, recent works [13, 24] advocate the use of top-down partitioning-driven placement with analytical elements for timing optimization. This provides a generic framework for large-scale placement with near-linear runtime, based on the strong empirical record of min-cut algorithms in wirelength-/congestion-driven placement [4].

The variability of results produced by commercial placers and the attention given to timing-driven placement by industry leaders (e.g., Aart de Geus at ISPD 2000), suggest that the timing-driven placement problem is far from solved. The problem becomes *more difficult* with more objects to place, and *more critical* in the overall design flow, with each successive technology node. Therefore, industrial and academic researchers seek new and more effective approaches.

The contributions of this work are

- a generic continuous path-timing optimization that is the first to avoid *heuristic* budgeting and re-weighting;
- combining path-timing optimization with top-down placement;
- a competitive implementation of a timing-driven placement based on our earlier work on wirelength- and congestion-driven placement[4]; and

[1] E.g., the authors of [5] estimated that explicitly storing all 245K paths in their 5K-cell design requires 163Mb of disk space. An equivalent compact representation took only 1.8Mb in human-readable ASCII format.

[2] Combinations of net re-weighting and delay budgeting have also been proposed (e.g, in [27]).

[3] E.g., at each stage of recursive min-cut in [21], non-critical nets get weights inversely proportional to their slacks, and critical connection get slightly higher weights.

[4] A reasonable mathematical framework for net re-weighting is available via Lagrangian relaxation, but such formulations are vulnerable to combinatorial explosion and imply *linear* convergence of numerical methods versus quadratic convergence of more efficient Newton-based methods.

ISPD'02, April 7-10, 2002, San Diego, California, USA.

- an evaluation flow for large-scale timing-driven placement with a major commercial timing analyzer.

The remainder of this paper is organized as follows. Background is covered in Section 2, including top-down placement, signal delay modeling and static timing analysis. Section 3 covers our new continuous path-delay minimization, which is embedded into a top-down placement framework in Section 4. The empirical validation is given in Section 5, and Section 6 concludes the paper.

2. BACKGROUND

Timing-driven placement draws upon the two more intuitive elements of wirelength-driven placement and timing analysis. Delays in large-scale layouts must be accurately yet quickly computed. Such a tradeoff is provided by static timing analysis (STA) tuned to err on the pessimistic side. STA relies on (i) models of signal delays in individual gates and wires, and (ii) (function-aware) path timing analysis based on gate/wire delays.

2.1 Top-down Placement

Top-down algorithms seek to decompose a given placement instance into smaller instances by subdividing the placement region, assigning modules to subregions, reformulating constraints, and cutting the netlist — such that good solutions to smaller instances (subproblems) combine into good solutions of the original problem. In practice, such a decomposition is accomplished by multilevel min-cut hypergraph partitioning that attempts to minimize the number of nets incident to nodes in multiple partitions. Each hypergraph partitioning instance is induced from a rectangular region, or *block*, in the layout. Conceptually, a block corresponds to (i) a placement region with allowed locations, (ii) a collection of modules to be placed in this region, (iii) all nets incident to the contained modules, and (iv) locations of all modules beyond the given region that are adjacent to some modules in the region (considered as *terminals* with fixed locations). Cells inside the block are represented as hypergraph nodes, and hyperedges are induced by nets incident to cells in the blocks. Node weights represent cell areas. Partitioning solutions must approximately equalize total weight in partitions to prevent more cells being assigned to a block than can be placed inside without overlaps.

The top-down placement process can be viewed as a sequence of passes where each pass refines every existing block into smaller blocks.[5] These smaller blocks will collectively contain all the layout area and cells that the original block did. Some of the cells in a given block may be tightly connected to external cells (*terminals*) located close to the smaller blocks to be created. Ignoring such connections implies a bigger discrepancy between good *min-cut* partitioning solutions and solutions that result in better placements. Yet, external terminals are irrelevant to the classic partitioning formulation as they cannot be freely assigned to partitions due to their fixed status. A compromise is achieved by an extended formulation for "partitioning with fixed terminals", where the terminals are "propagated to" (fixed in) one or more partitions and assigned zero areas [9]. Terminal propagation is typically driven by the relative geometric proximity of terminals to subregions/partitions [3] and is essential to the success of min-cut recursive bisection.

2.2 Static Timing Analysis

A standard-cell circuit has cells $C = \{c_k\}$ and signal nets $N = \{n_l\}$. Nets are connected to cells with *pins*, each of which can be either an IN-pin or an OUT-pin (directionality).[6] The *full timing graph* [16] is built using the pins of the circuit as its vertices $V = \{v_i\}$. Timing edges $E = \{e_{ij}\}$ that connect pins are constructed in two ways. Each signal net is converted into a set of oriented *interconnect* edges that connect each OUT-pin of the net to all IN-pins of the net. Each standard cell or macro is represented by a set of oriented *intracellular* edges determined by the contents of the cell, with the exception that intracellular edges of latches and flip-flops (storage elements) are ignored. We assume acyclic timing graphs and in practice break cycles by removing back-edges discovered during DFS traversals. The delay attributed to a given timing edge is a function of vertex locations, including those of the edge source and sink. In our work on large-scale placement we use a *reduced timing graph* where all pins on every placeable object are clustered into a single vertex so that every vertex can be placed independently. Thus, intracellular arcs are removed; gate delays are computed per driver pin and added to wire delays on the respective outgoing edges.

The primary objective of timing-driven layout, *cycle time*, is modeled by the maximal delay along a directed path between particular source and sink pairs (primary I/Os and I/Os of store elements). The delay t_π along a path $\pi = (e_{i_1 j_1}, e_{i_2,j_2}, \dots)$ is a sum of edge delays ($j_1 = i_2,\ j_2 = i_3,\ \dots$). More generally, every path may come with a timing constraint c_π, which is satisfied if and only if $t_\pi \le c_\pi$, corresponding to "max-delay" *setup* constraints.[7] Those timing constraints c_π (i.e., upper bounds on path delays) are not given explicitly, but rather defined via *actual arrival times* (*AAT*) and *required arrival times* (*RAT*) for every driver-pin and primary output. The timing constraint for a path π is then the difference between *RAT@sink* $\Leftrightarrow$ *AAT@source*. We do not *a priori* restrict how the set of eligible paths is defined; rather, this is relegated to (i) generic static timing analysis based on path tracing [16] described below, plus (ii) extensions to handle false paths, multi-cycle paths, and variabilities (dynamic and statistical) that lead to such phenomena as crosstalk- or supply-induced delay uncertainty. Our own STA implementation has only the generic capability, but we also evaluate placements with an industry STA tool.

Given delays of timing edges (e.g., computed from a placement), static timing analysis (STA) determines (i) whether all timing constraints are satisfied, and (ii) which directed paths violate their constraints. The key to computational efficiency of STA is the notion of *slack*, which allows us to avoid enumerating all paths [16].

Definition 1: The *slack* of a path π is $s_\pi = c_\pi \Leftrightarrow t_\pi$. The *slack* of a timing edge (vertex) is the smallest path slack among the paths containing this edge (vertex).

Lemma-Definition 2: In a given timing graph, the minimal vertex slack, minimal edge slack and minimal path slack are equal. This value is called *circuit slack* and is a convex function of edge delays, which are functions of cell locations.

Negative slack is indicative of violated timing constraints. Therefore, timing-driven layout aims to maximize {minimal slack over all paths}, computed by STA, to improve cycle time. To compute min-slack in linear time, two topological traversals propagate *RAT* and *AAT* from sources and sinks to all vertices. Namely, one traversal computes the *AAT* at a vertex *v* when, for every directed edge *uv* ending at *v*, the *AAT* at *u* and the delay of *uv* are known. We write $AAT_v = \max_{uv}\{delay(uv) + AAT_u\}$. Similarly, RAT_v can be computed when, for every directed edge *vw* beginning at *v*, $delay(vw)$ and RAT_w is known. Then $RAT_v = \min_{vw}\{RAT_w \Leftrightarrow delay(vw)\}$. With *AAT* and *RAT* available at all vertices, slacks at individual vertices are computed as $RAT \Leftrightarrow AAT$, and similarly for edges [16]. If the minimum slack is negative, some paths must violate their constraints and have negative slacks on all of their edges.

[5] When recursive bisection is applied, careful choice of vertical versus horizontal cut direction is important, — one rule of thumb is to keep the aspect ratios of the blocks as close to a given constant (typically 1.0) as possible, for as long as possible.

[6] Bidirectional pins can be captured using pairs of unidirectional pins and constrained timing graph traversals.

[7] As in [28, 11], we leave "min-delay" *hold* constraints to clock-tree tunings and local optimizations, e.g., buffer padding, sizing, snaking, etc.

2.3 Gate and Wire Delay Modeling

When calculating the gate/edge delays on each *continuous placement iteration* (see Section 5.1), we use four different delay models: (i) *linear*, (ii) *quadratic*, (iii) *Elmore MST*, and (iv) *Elmore star*. In the linear model, the delay between a source pin and a sink pin is calculated by the Manhattan distance between them. The quadratic model uses the square of the Manhattan distance. Both the linear model and the quadratic model combine the gate delay and the edge delay into one model which is based on the length. The Elmore MST model uses a lumped-distributed RC gate and wire delay model that is calculated based on the MST for a given net.[8] The Elmore star model uses the same calculation with a star tree (instead of the MST tree) where each sink pin has a direct edge from the source pin. Thus, the load of a given sink pin only affects the interconnect delay of that particular edge.

The Elmore delay calculation uses cell locations and the following parameters (cf. [12]):

- r and c are per-unit resistance and capacitance of interconnect; when routing assignments are unknown, statistical averages from typical placements are used;
- R_i is the resistance of a given driver pin i and C_j is the capacitance of a given sink pin j.

Load-dependent gate delay at output pin i is computed as $R_i(C_{int} + \Sigma_j C_j)$ where the summation is over sinks j and C_{int} is the total interconnect capacitance on the driven net. $C_{int} = cW$, where W is an estimate of the total net length, e.g., the length of a Rectilinear Minimum Spanning Tree (RMST) or the total length of the edges in the star tree.[9] Interconnect delays are computed as rcL^2 where L is the length of a timing edge. The delay calculation in the Elmore MST is more expensive because it includes the construction and the traversal of the tree.

3. MIN-MAX PLACEMENT

Our continuous optimization assumes that some vertices of the timing graph are restricted to fixed locations or rectangles; thus, it can be used in top-down placement. The *minimization* of the path-delay function Φ below, over all paths π, includes optimization of the worst slack as a special case:

$$\Phi = \max_{\pi} \frac{t_\pi}{c_\pi} = \max_{\pi} \frac{\sum_{e\in\pi} d_e}{c_\pi} \tag{1}$$

Here d_e denotes the signal delay along edge e of the timing graph and can also be written as $d_e = d_{ij}(x_i, y_i, x_j, y_j)$, making Φ a function of vertex locations via convex delay models for individual edges.[10] Common edge delay models can be based on linear or quadratic (squared) edge wirelength, or on Elmore delay (see Section 2.3).

Observation 3. A placement satisfies all timing constraints if and only if $\Phi \le 1.0$.

Φ is a *multiplicative* generalization of the common (*additive*) slack objective S, since $\Phi \le 1.0 \Leftrightarrow S \ge 0.0$.

When c_π are identical, $\min\Phi$ is equivalent to the minimization of maximal path delay, and thus to $\max S$ (see Section 2.2). The general $\max S$ problem with arbitrary path delays c_π determined by *AATs* and *RATs* can be reduced to the case of identical c_π by adding a super-source and a super-sink connected respectively by constant-delay edges to all timing sources and sinks. Therefore, the ordinary slack maximization is a special case of $\min\Phi$. Our generic placement algorithm for $\min\Phi$ is a reduction to a simpler objective function.

[8] This is generally selected for simplicity and speed. With a generic STA implementation, more accurate models or black-box delay calculators can be used when affordable in terms of runtime.

[9] In addition, several other interconnect length estimations can be used, such as a weighted half-perimeter wirelength [2], the length of a minimum single-trunk Steiner tree or the length of a heuristic Rectilinear Steiner Minimum Tree (RSMT).

[10] To better model delays, the functions $d_{ij}()$ could also depend on locations of cells (topologically) adjacent to i and j [18].

3.1 Minimization of Φ by re-weighting

Given *edge weights* $w_{ij} \ge 0$ on the timing graph, we *minimize* the following MAX-based objective function[11]

$$\delta = \max_{ij} w_{ij} d_{ij}(x_i, x_j, y_i, y_j) \tag{2}$$

Define $\delta_{ij} = w_{ij} d_{ij}(x_i, x_j, y_i, y_j)$ so that $\delta = \max_{ij} \delta_{ij}$.

Our placement optimization of Φ starts from an initial solution.[12] Then we compute edge delays and perform Static Timing Analysis. Based on slacks/criticalities and edge delays, we compute w_{ij} as outlined below. After that, the current placement is changed to *minimize* the function given by Equation (2). The values of δ and δ_{ij} after placement at iteration k are denoted by $\delta^{(k)}$, $\delta_{ij}^{(k)}$ and $d_{ij}^{(k)}$ resp. We prove that in this process Φ cannot increase, implying monotonic convergence.

Lemma 4 Given (i) an arbitrary set $w_{ij} \ge 0$ with at least one nonzero, and (ii) any minimum of the respective MAX-based objective, all edge delays cannot be improved simultaneously by another placement. I.e., there is no $\varepsilon > 0$ and new placement for which the delay of every edge e_{ij} is $d'_{ij} \le d_{ij} \Leftrightarrow \varepsilon$.

Proof by contradiction. Suppose we have found $\varepsilon > 0$ and a new placement with ε-smaller edge delays. Then define $C = \max_{ij} d'_{ij}/d_{ij}$ and note that $C \le \max_{ij}(d_{ij} \Leftrightarrow \varepsilon)/d_{ij} < 1$. Since every edge delay d'_{ij} in the new placement will be no longer than Cd_{ij}, the value of the objective function for the new placement will be $C < 1$ times of the value for the original placement. However, this is impossible, since the original placement minimized the objective function. □

Definition 5 Given $k > 1$ and a placement for which the objective function (2) has value $\delta^{(k)}$, we call an iteration of {re-weighting and placement} *successful* iff a placement is found for which $\delta^{(k+1)} \le \delta^{(k)}$. Otherwise we say that the iteration has failed. Finding a true minimal value of the function (2) is not required. An iteration is *trivially successful* if the re-weighted objective function has value $\le \delta^{(k)}$ with respect to the previous placement.

Lemma 6 All timing constraints are satisfied if

$$d_{ij}^{(k+1)} \le d_{ij}^{(k)} / \left[\max_{\pi' \ni e_{ij}} t_{\pi'}^{(k)} / c_{\pi'} \right]$$

Proof $\left[\max_{\pi' \ni e_{ij}} t_{\pi'}^{(k)} / c_{\pi'}\right]$ is the worst ratio between the delay of a path passing through e_{ij} and its constraint. Therefore by reducing every edge delay on path π by the resp. ratio, we will ensure that path delay t_π is within its constraint c_π

$$t_\pi^{(k+1)} = \Sigma_{e_{ij}\in\pi} d_{ij}^{(k+1)} \le c_\pi (\Sigma_{e_{ij}\in\pi} d_{ij}^{(k)}) / t_\pi^{(k)} = c_\pi$$

□

We now determine multiplicative factors for re-weighting such that after a successful iteration all timing constraints are satisfied according to Lemma 6. Namely, for any path π and any edge $e_{ij} \in \pi$ we seek to ensure the left-most inequality in the following chain (the remaining equality and inequality hold *a priori*)

$$d_{ij}^{(k+1)} \le d_{ij}^{(k)} / \left[\max_{\pi' \ni e_{ij}} t_{\pi'}^{(k)} / c_{\pi'} \right] \tag{3}$$

$$= d_{ij}^{(k)} \left[\min_{\pi' \ni e_{ij}} c_{\pi'} / t_{\pi'}^{(k)} \right] \le d_{ij}^{(k)} c_\pi / t_\pi^{(k)} \tag{4}$$

[11] The main difference from more common *total (equiv. average) wirelength* objective is the use of max instead of Σ.

[12] Quadratic placements work well in practice and can be produced very quickly; faster/better approaches are possible.

To ensure inequality (4), we note that $d_{ij}^{(k+1)} \le \delta^{(k+1)}/w_{ij}^{(k+1)}$ by definition of $\delta^{(k+1)}$ and $\delta^{(k+1)} \le \delta^{(k)}$ by definition of a successful iteration. Therefore, our goal will be reached once we have $\delta^{(k)}/w_{ij}^{(k+1)} = d_{ij}^{(k)} / \left[\max_{\pi' \ni e_{ij}} t_{\pi'}^{(k)}/c_{\pi'}\right]$ which can be accomplished by re-weighting:

$$w_{ij}^{(k+1)} = (\delta^{(k)}/d_{ij}^{(k)}) \left[\max_{\pi' \ni e_{ij}} t_{\pi'}^{(k)}/c_{\pi'}\right] \quad (5)$$

The max-terms in this formula are called "criticalities" and can be computed using static timing analysis, which is especially efficient when the main global objective is slack maximization.

Theorem ITC (Immediate Timing Convergence) All timing constraints are satisfied after one *successful* iteration if re-weighting is performed according to Equation (5).

Now we show that small placement changes caused by the proposed iteration of re-weighting and placement also minimize Φ. When the current placement is perturbed only slightly, $\delta^{(k)}$ is approximately constant, and so are the values d_{ij}^k. We can now rewrite the MAX-based objective function as

$$\max_{ij} w_{ij} d_{ij} = \max_{e_{ij}} \frac{\delta^{(k)}}{d_{ij}^{(k)}} \left[\max_{\pi' \ni e_{ij}} \frac{t_{\pi'}^{(k)}}{c_{\pi'}}\right] d_{ij} \quad (6)$$

$$\approx \delta^{(k)} \max_{\pi' \ni e_{ij}} \frac{t_{\pi'}^{(k)}}{c_{\pi'}} = \delta^{(k)} \max_{\pi'} \frac{t_{\pi'}^{(k)}}{c_{\pi'}} \quad (7)$$

3.2 Interpretations of re-weighting and comparisons to known results

Define the *timing criticality* of an edge to be the timing criticality of the most critical path passing through the edge, measured by its contribution to Φ, i.e., $\kappa_{ij}^{(k)} = \max_{\pi' \ni e_{ij}} t_{\pi'}^{(k)}/c_{\pi'}$. This can be viewed as the multiplicative version of the traditional negative slack [16]. We also define *relative edge delay* $\rho_{ij}^{(k)} = \delta^{(k)}/\delta_{ij}^{(k)}$. Now Equation (5) can be interpreted as multiplying each weight $w_{ij}^{(k)}$ by a *weight adjustment factor* $\alpha_{ij}^{(k)} = \rho_{ij}^{(k)} \kappa_{ij}^{(k)}$ which can be greater than, less than or equal to 1.0. The main idea here is to force critical edges to shorten (decrease their delays) *by only as much as they need* to cease being critical and allow non-critical edges to elongate (increase their delays) only by *as much as they can* without becoming critical.

Intuitively, the re-weighting can be decomposed into two steps. At the first step every edge weight is multiplied by relative edge delay, which does not change the value of the objective function on the current placement, but makes all edge terms equal ($\rho_{ij}^{(k)} w_{ij}^{(k)} d_{ij}^{(k)} = \delta^{(k)}$ for any i, j). Following that, new edge weights are multiplied by timing criticalities which will increase the objective thanks to timing-critical edges (thus the iteration will never be *trivially successful*). Improving the re-weighted objective improves critical edges and thus improves Φ.

Multiplication by relative edge delays is somewhat counter-intuitive because *shorter edges* on critical paths receive *heavier weights than longer edges* on the same paths. However, the useful effect of multiplication by relative edge delays is that all edge terms attain the maximum and the current placement becomes unimprovable (cf. Lemma 4). Loosely speaking, the work in [28] mentions the κ term, but computes it for vertices rather than edges. However, neither [28], nor [11] have the ρ term, which makes their delay re-budgeting algorithms heuristic.

3.3 Lower-level minimization

The minimization of the MAX-based objective can be performed by linear or non-linear programming [19] depending on specific delay models. In fact, for the linear-wirelength delay objectives, the LP formulation is solvable in linear time using Bellman-Ford [8, Exercise 24.4-5]. We implemented a simpler algorithm that is extremely fast in practice and easy to perform on the original circuit representation. It traverses vertices in an arbitrary order and places them in locally-optimal locations. Such a pass cannot increase the objective, implying monotonic convergence. Given that most vertices are adjacent to very few other vertices (netlist are sparse due to technology and library constraints), every pass has linear runtime. We continue the passes until the objective improves by $\le 0.1\%$. Few passes are required in practice because critical paths have few stages of logic.

3.4 Extensions

Since more accurate delay objectives do not fit into the linear minimization model described above, we extended the overall placement algorithm to accommodate more general delay objectives. The main idea is to perform numerical differentiation and locally minimize linear approximations of given objective functions. This can also be seen as a variant of gradient minimization (in a multi-dimensional space). For this, we first find the smallest height/width over all standard cells. Then we choose an $\varepsilon > 0$ by taking $1/10$ of that number and consider a vector that is applied to the current location of a given cell and has length $\varepsilon > 0$. We then restrict the movements of each cell to a direction that sensitizes the delay function. The tangent line to the graph of the delay function, with respect to that direction, is approximated as the unique straight line passing through the following two points. The coordinates of the first point are the current placement and the delay value, and the coordinates of the second point, are the current placement shifted by the ε-vector and the delay computed for that placement. This construction is applied to each movable object independently; it is straightforward to account for details such as constant pin offsets. The linear approximation can be solved as described earlier. Of course, the MST and SMT length are not smooth and are not convex functions. Currently, we do not know how to achieve global minima for such tree-based objectives through differencing schemes.

4. MIN-MAX PLACEMENT IN A TOP-DOWN PLACEMENT FLOW

Figure 1 extends the basic top-down placement framework described in Section 3. This combined algorithm attempts to improve cycle-time and HPWL simultaneously. It starts by a call to min-max placement that returns cell locations optimizing worst-slack, as well as actual cell slacks in that placement. This information is translated into pre-assignments for the subsequent partitioning runs. Intuitively, the cells with worst slacks should be selected and pre-assigned to the partitions where they were placed by the continuous formulation. Min-max placement optimizes slack, thus slack cannot improve during further top-down placement as region constraints are added. Therefore, the cells with worst slacks can only become more critical in the future and should be pre-assigned to partitions in such a way that the worst slack does not worsen.

Selection of cells to be pre-assigned, based on locations and slacks, is performed in two stages. In the first stage, a "goodness" score is computed for each cell as a linear combination of cell slack and a delay equivalent of the cell's distance to the cut-line in its block. Subtracting a weighted delay-equivalent from slack captures the deterioration of slack in case the cell is assigned into a "wrong" partition. The lower (worse) the score, the more important it is to pre-assign the cell into the partition containing its continuous location. Cells are sorted in increasing order of scores, and those with positive scores are considered "good enough" to not be pre-assigned before a partitioner is called.

We try not to pre-assign too many cells before calling a partitioner; otherwise, half-perimeter wirelength and congestion of resulting placement can increase. To this end, we introduce two parameters that further limit the number of cells selected to be assigned at any given level. Both limits are in terms of movable cells; one applies to the whole layout, and the other to individual blocks.

Once all movable cells in the layout are sorted by their scores, those

Top-Down Timing-Driven Placement Flow

Variables:
$B, B1, B2$ — placement blocks (aka "bins")
$A1$ and $A2$ — stacks of placement blocks
P — cell locations (placement)
N — the circuit netlist
α, % — trades off timing- and wirelength
Initialization: single block in $A1$ representing the original placement instance
$A2$ — empty; P — arbitrary
α — from 0% (WL-driven) to 50% (t-driven)

Output: P — global placement

Algorithm:

```
while ( A1 not empty )
{ - find continuous locations P that minimize
    signal delay in N subject to block constraints
  - find critical paths
  - mark cells that lie on critical paths
  - while ( A1 not empty )
    { pop B from A1
      if ( B small enough ) { process end-case; continue }
      prepare to partition B into B1 and B2
        (terminal propagation, etc.)
      assign marked cells in B to B1 and B2
        according to P and fix them
      call hypergraph partitioner
      finalize B1 and B2; push B1 and B2 onto A2
    }
  - copy A2 to A1; clear A2
}
```

Figure 1: Pseudocode of proposed timing-driven placement framework.

cells are traversed in the order of increasing goodness and marked (as pre- assigned), with their areas accumulated.[13] This traversal goes on until the total area of marked cells reaches a global area limit (% of the total area of all movable cells). Later, when a block is about to be partitioned, its cells marked for pre-assignment are traversed again in order of increasing scores and pre-assigned to partitions based on their continuous locations. This traversal continues until the total area of pre-assigned cells reaches the area limit for the block.

We also reuse the continuous cell locations for more accurate terminal propagation. In many cases, this improves both half-perimeter wirelength and cycle time.

5. EMPIRICAL VALIDATION

Our implementation CapoT is based on the Capo placer [4]. The general architecture is similar to the "slack-graph" technique [5], with its separation of concerns between delay calculation, STA and placement. We additionally separate continuous min-max placement from top-down placement.

Name	# Cells	# Nets	Clk Period
DesignA	6390	8033	12.3 ns
DesignB	20449	21230	11.1 ns
DesignC	40349	42487	16.6 ns
DesignD	58987	59922	68.0 ns

Table 1: Test case parameters.

5.1 Our Placer Implementation

In the timing-driven regime, CapoT calls our min-max placer TDplace, which interfaces with our Static Timing Analyzer (STA). (Recall from above that STA is a generic path-tracer.) Thus, TDplace influences the construction of partitioning instances in CapoT (see Section 4). TDplace and STA are instantiated at the beginning of the top-down placement and construct a timing graph from the netlist, such that vertices correspond to movable objects. Timing edges are created from every source in a given hyperedge to every sink on that same hyperedge. All information necessary to compute gate and edge delays as functions of placement is made available. Fixed-delay edges (e.g., between fixed cells/pads) and storage elements are marked in the timing graph. The directed graph is traversed by a depth-first search, and back-edges that cause purely combinational cycles are removed. STA performs classical static analysis with two topological traversals and slack computations, as described in Subsection 2.2. It also computes *criticalities* from slacks, assuming that all AATs are the same and all RATs are the same (we also implemented the general case, but have not used it in this work).

After STA is constructed, TDplace is instantiated, using the locations of fixed vertices and optional initial locations of movable vertices. There is an optional array of bounding boxes, one per vertex, that constrain the possible locations of respective vertices. The first continuous placement is performed subject to every movable object being inside the layout bounding box, with initial location at the geometric center. Then, CapoT reads placement solutions and vertex slacks, as well as various status information such as the worst slack. The cell locations and slacks reported by TDplace are used by CapoT to pre-assign hypergraph nodes before multi-level partitioning, as explained below. After every round of min-cut partitioning, the array of bounding boxes in TDplace is changed by CapoT according to the partitioning results. Subsequently TDplace is called to perform a continuous-variable placement subject to new bounding-box constraints. The cell locations and slacks are used at the next round of partitioning, and this top-down placement process continues until reaching end-cases (see Figure 1).

While performing continuous placement, TDplace *iterates* gate and edge delay calculations along the lines of Subsection 2.3, calls to STA, and min-max-weighted-delay placement until a convergence criterion is met. The min-max-weighted-delay placement is performed by a nested iteration. This iteration attempts to improve a previously existing solution by linear passes in which every vertex is placed optimally. Due to the proven monotonic convergence, termination criteria are fairly straightforward — each iteration is stopped when its objective function changes by less than 0.1%.

We validate our placer by checking the "sanity" of its behavior with respect to the internal optimization objective. For each industry test case (see Table 1) we drive the placer with four edge delay models (linear, quadratic, Elmore MST and Elmore star), and both the standard delay objective and the extended delay objective described above (denoted by *-td* and *-td1* respectively). Thus, for every test case eight separate placements are obtained (and for each delay model as well, eight separate placements are obtained). All placement results are evaluated by our STA timing evaluator, with respect to all four delay models. According to each delay model, we rank the placements from one to eight in order of decreasing slack. We then find the rank achieved when the placer was actually driven by this specific delay model.[14] Table 2 reports these ranks, averaged over all four test cases. The ranks for the combination of *-td* and *-td1* can range between 1.5 and 7.5; the ranks for either objective alone can range between 1 and 4. The results indicate that our algorithm is better able to optimize with respect to the linear delay model than with respect to Elmore-based delay models. This may partly explain why timing optimization can return worse results than timing-oblivious wirelength minimization: our algorithm targets simpler delay models and does not take into account the subtleties of non-linear and topology-sensitive delay models.

5.2 Experimental Results

We report half-perimeter WL (HPWL) and timing slack results for global timing-driven placement based on the linear, the quadratic and

[13] This $O(N \log(N))$ sorting-based computation can be sped up by a linear-time weighted-median computation, but its share in total runtime is already negligible, and therefore we simply call `sort()` from the Standard Template Library.

[14] If the placer "behaves perfectly" then the, e.g., quadratic-driven placements would have ranks 1 and 2 among all eight placements evaluated by the quadratic delay model. On the other hand, if these ranks are around 4.5 (or higher), then driving the placer with the given delay model achieves no better (or, worse) results than driving it with a random other delay model.

Option	Linear	Quadratic	Elmore MST	Elmore Star
-td and -td1	2.375	2.875	4.5	3.375
-td	1.75	1.25	2.5	2.5
-td1	1	1.75	2.75	2.25

Table 2: Evaluation of placer's behavior on its internal objective.

the Elmore MST delay models. Both the linear model and the quadratic model are technology-blind (results are independent of interconnect parasitics). The MST model is used with technology parameters (calibration and correlation of our internal STA, library and tech file modeling, timing constraints definition, etc. required significant effort but details are beyond space limitations). We applied the two options for timing-driven placement described earlier in the text (-td and -td1). Table 3 compares CapoT with an industrial placer in a flow with black-box industry STA timing analysis. Non-timing-driven results are marked with *ntd*. We used the industry placer in both timing-driven (industry-td) and non-timing-driven (industry-ntd) configurations. CapoT was called by a "meta-placer" that applied branch-and-bound placement improvement in sliding windows and removed cell occasional cell overlaps by a naive greedy heuristic. The meta-placer also greedily optimized cell orientations within the constraints given by the input LEF/DEF files. Note that we do not have a detailed timing-driven placement capability, while the industry placer does.

Placer/Config	HPWL	slack	HPWL	slack
	Design A		Design B	
industry-ntd	6.74e5	0.44	2.73e6	0.49
industry-td	6.58e5	0.06	2.59e6	0.55
mp-ntd	8.89e5	-0.01	2.49e6	0.66
mp-td-lin	9.40e5	0.08	2.73e6	0.28
mp-td-mst	8.93e5	-0.14	2.78e6	0.42
mp-td-quad	9.19e5	-0.09	2.69e6	0.48
mp-td1-lin	9.07e5	0.08	2.68e6	0.62
mp-td1-mst	9.16e5	-0.38	2.81e6	0.64
mp-td1-quad	8.94e5	0.15	2.80e6	0.62
	DesignC		DesignD	
industry-ntd	3.56e6	-5.70	2.39e7	-3.47
industry-td	3.41e6	-5.66	2.20e7	-4.98
mp-ntd	3.20e6	-5.62	1.97e7	0.65
mp-td-lin	3.41e6	-5.64	2.14e7	-3.36
mp-td-mst	3.30e6	-5.74	2.17e7	-14.57
mp-td-quad	3.34e6	-5.69	2.21e7	-4.43
mp-td1-lin	3.40e6	-5.88	2.12e7	1.30
mp-td1-mst	3.34e6	-5.75	2.09e7	-0.55
mp-td1-quad	3.34e6	-5.75	2.08e7	-1.37

Table 3: Timing-driven placement results.

The experimental results (test case C is essentially non-probative) show that our implementation performs competitively. Currently, the linear star delay model outperforms Elmore type models. Also, in several cases, WL-driven versions of both our implementation and the industry tool outperformed td versions. We stress that the evaluations were performed by an industrial static timing analyzer, independent of the placer implementations. Thus, our conclusions may have dependencies on instance size, technology scaling, library characterization (since our delay modeling and calculation is abstracted from standard .lib models), and mis-correlations between the industry STA and our STA capability (we drive our placer with internal STA calculations). We conclude that formulating a more precise delay objective and minimizing it by a reasonable algorithm does not necessarily yield better results.

6. CONCLUSIONS

We have proposed a new global timing-driven placement algorithm and evaluated it on a set of recent industrial circuit benchmarks. Circuit delay was evaluated by a commercial static timing analyzer (i) after placement by our placer, and (ii) after placement by a commercial placer. The main contribution of this paper is to global timing-driven placement; without a detailed placer, we were able to demonstrate competitive results on several industrial benchmarks. The proposed algorithms are flexible and can be adapted to many placement frameworks, especially those based on quadratic placement.

Our empirical results justify the pursuit of simple global objectives for timing-driven placement and show that minimizing a simple objective well may be more useful than minimizing a very accurate objective poorly. We believe that the utility of our approaches should increase with future scaling of VLSI technologies.

7. REFERENCES

[1] M. Burstein, "Timing Influenced Layout Design", *Proc. Design Automation Conf.*, 1985, pp. 124-130.

[2] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov and A. Zelikovsky, "On Wirelength Estimations for Row-Based Placement", *IEEE Trans. on CAD*, 18(9), Sept. 1999, pp. 1265-1278.

[3] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Optimal Partitioning and End-Case Placement for Standard-Cell Layout", *IEEE Trans. on CAD*, 19(11), Nov. 2000, pp. 1304-1313.

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

[5] C.-C. Chang, J. Lee, M. Stabenfeldt and R.-S. Tsay, "A Practical All-Path Timing-Driven Place and Route Design System", *Proc. Asia-Pacific Conf. on Circuits and Systems*, 1994, pp. 560-563.

[6] A. H. Chao, E. M. Nequist and T. D. Vuong, "Direct Solutions of Performance Constraints During Placement", *Proc. Custom Integrated Circuits Conf.*, 1990, pp. 27.2.1-27.2.4.

[7] Y.-C. Chou and Y.-L. Lin, "A Performance-Driven Standard-Cell Placer Based on a Modified Force-Directed Algorithm", *Proc. Intl. Symp. on Physical Design*, 2001, pp. 24-29.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, 2nd ed., MIT Press, 2001.

[9] A. E. Dunlop and B. W. Kernighan, "A Procedure for Placement of Standard Cell VLSI Circuits", *IEEE Trans. on CAD* 4(1), 1985, pp. 92-98.

[10] H. Eisenmann and F. Johaness, "Generic Global Placement and Floorplanning", *Proc. Design Automation Conf.*, 1998, pp. 269-274.

[11] J. Frankle, "Iterative and Adaptive Slack Allocation for Performance-Driven Layout and FPGA Routing", *Proc. Design Automation Conf.*, 1992, pp. 539-542.

[12] P. Gopalakrishnan, A. Obasioglu, L. Pileggi and S. Raje, "Overcoming Wireload Model Uncertainty During Physical Design", *Proc. Intl. Symp. on Physical Design*, 2001, pp. 182-189.

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

[14] T. Hamada, C. K. Cheng and P. M. Chau, "Prime: A Timing-Driven Placement Tool Using a Piecewise Linear Resistive Network Approach", *Proc. Design Automation Conf.*, 1991, pp. 531-536.

[15] P. S. Hauge, R. Nair and E. J. Yoffa, "Circuit Placement for Predictable Performance", *Proc. Intl. Conf. on Computer-Aided Design*, 1987, pp. 88-91.

[16] R. B. Hitchcock, Sr., G. L. Smith and D. D. Cheng, "Timing Analysis of Computer Hardware", *IBM J. Res. Develop.* 26(1), 1982, pp. 100-108.

[17] M. A. B. Jackson, A. Srinivasan and E. S. Kuh, "A Fast Algorithm for Performance-Driven Placement", *Proc. Intl. Conf. on Computer-Aided Design*, pp. 328-331.

[18] T. Koide, M. Ono, S. Wakabayashi and Y. Nishimaru, "Par-POPINS: A Timing-Driven Parallel Placement Method with the Elmore Delay Model for Row Based VLSIs". *Proc. Asia and South Pacific Design Automation Conf.*, 1997, pp. 133-140.

[19] D. G. Luenberger, *Linear and Nonlinear Programming*, 2nd ed., Addison Wesley, 1984.

[20] W. K. Luk, "A Fast Physical Constraint Generator for Timing Driven Layout", *Proc. Design Automation Conf.*, 1991, pp. 626-631.

[21] M. Marek-Sadowska and S. P. Lin, "Timing-Driven Layout of Cell-Based ICs", *VLSI Systems Design*, May 1986, pp.63-73.

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

[23] R. Nair, C. L. Berman, P. S. Hauge and E. J. Yoffa, "Generation of Performance Constraints for Layout", *IEEE Trans. on CAD* 8(8), Aug. 1989, pp. 860-874.

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

[25] B. M. Riess and G. G. Ettelt, "Speed: Fast and Efficient Timing Driven Placement", *Proc. Intl. Symp. on Circuits And Systems*, 1995, pp. 377-380.

[26] M. Sarrafzadeh, D. Knol and G. Tellez, "Unification of Budgeting and Placement", *Proc. Design Automation Conf.*, 1997, pp. 758-761.

[27] R. S. Tsay and J. Koehl, "An Analytical Net Weighting Approach for Performance Optimization in Circuit Placement", *Proc. Design Automation Conf.*, 1991, pp. 620-625.

[28] H. Youssef, R.-B. Lin and E. Shragowitz, "Bounds on Net Delays for VLSI Circuits", *IEEE Trans. on Circuits and Systems*, 39(11), Nov. 1992, pp. 815-824.