Analytical Optimization Of Signal Delays in VLSI Placement

Andrew B. Kahng$^\sharp$ and Igor L. Markov‡
$\sharp$ UCSD CSE and ECE Depts., La Jolla, CA 92093-0114
$\ddagger$ Univ. of Michigan, EECS Department, Ann Arbor, MI 48109-2122
abk@ucsd.edu, imarkov@umich.edu
IBM contact: Paul G. Villarrubia, `pgvillar@us.ibm.com`

Abstract

In analytical placement, one seeks locations of circuit modules that optimize an objective function, but allows module overlaps. 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.

1 Introduction

The significance of timing-driven layout increases with the dominance of interconnect delays over device delays. While today's commercial placement engines can evaluate increasingly accurate measures of path timing, simple models often lead to more efficient minimization. To first order, the 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 [16, 6, 13] minimize delays along *explicitly enumerated paths*. However, explicit path enumeration becomes impractical when the number of signal paths undergoes combinatorial explosion in large circuits. Indeed, [5] (1994) estimated that explicitely storing all 245K paths in their 5K-cell design requires 80 hours on their hardware and 163Mb of disk space. An equivalent compact representation took only 1.8Mb in human-readable ASCII format and was produced in several hours.

Combinatorial explosion is not a problem for static timing analysis methods [15, 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* [20, 1, 25] for weighted wirelength-driven placement or into additional constraints for such placement, e.g., per-net delay bounds in "delay budgeting" approaches [14, 22, 28, 19, 10]. Iterations are repeated until they bring no improvement. For example, at each stage of recursive min-cut in [20], non-critical nets get weights inversely proportional to their slacks, and critical connection get slightly higher weights. Combinations of net re-weighting and delay budgeting have also been proposed (e.g, in [27]). As noted in [10, 26], "net re-weighting" algorithms are often *ad hoc* and have poor convergence theory, i.e., if delays along critical nets decrease, other nets may become critical. [1] 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 leading-edge electronics today are often difficult even from the wirelength/congestion stand-point. Therefore, a robust placement algorithm must have a proven record in wirelength- and congestion-driven context without timing. Motivated by this circumstance, recent works [12, 23] advocate the use of top-down partitioning-driven placement with analytical elements for timing optimization. This provides a generic framework for large-scale layout with near-linear runtime, based on the strong empirical record of min-cut algorithms in wirelength- and congestion-driven placement [4].

The contributions of this work are

- A generic continuous path-timing optimization, first to avoid *heuristic* budgeting and re-weighting;
- An embedding of continuous path-timing optimization into top-down placement;

The remaining part 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.

[1] 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.

2 Background

Timing-driven placement draws upon the more intuitive wirelength-driven placement and timing analysis. Circuit delay models for large-scale layout must be sufficiently accurate yet quickly computable. Such trade-off 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) path-timing analysis in terms of 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 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.[2] 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 [8]. Terminal propagation is typically driven by the relative geometric proximity of terminals to subregions/partitions [3] and 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).[3] The *full timing graph* [15] 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 and macro are 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 (aka "store 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. Intracellular arcs are removed; gate delays are computed per driver pin and are added to interconnect delays on the respective outgoing edges.

A major 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 $\pi = (e_{i_1 j_1}, e_{i_2, j_2}, \ldots)$ is a sum of edge delays ($j_1 = i_2,\ j_2 = i_3, \ldots$). More generally, every path may come with a timing constraint c_π, which is satisfied if and only if $t_\pi \leq c_\pi$, corresponding to "max-delay" *setup* constraints.[4] 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 $RAT@sink - AAT@source$. We do not *a priori* restrict the set of eligible paths is defined, but rather delegate its treatment to (i) generic static timing analysis based on path tracing [15] described below, and (ii) drop-in use of extensions to handle false paths and multi-cycle paths (and also to model physical phenomena such as cross-talk, inductance and delay uncertainty).

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 that allows to avoid enumerating all paths [15].

Definition 1: The *slack* of a path π is $s_\pi = c_\pi - t_\pi$. 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 functions of edge delays, which are functions of cell locations.

Negative slack is indicative of violated timing con-

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

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

[4]As [28, 10], we leave "min-delay" *hold* constraints to clock-tree tunings and local optimizations, e.g., buffering, sizing, snaking, etc.

straints. 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 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 $RAT_v = \min_{vw}\{RAT_w - delay(vw)\}$. With AAT and RAT available at all vertices, slacks at individual vertices are computed as $RAT - AAT$, and similarly for edges [15]. If min-slack is negative, some paths must violate their constraints and have negative slacks on all of their edges.

2.3 Gate and Wire Delay Modeling

Our slope-agnostic "lumped RC" gate and wire delay models in terms of cell locations are simple and fast.[5] They are based on the following parameters (cf. [11]):

- r and c are per-unit resistance and capacitance of interconnect; when routing assignments are unknown, statistical averages from typical placements are used;
- for each cell, R_i is the resistance of driver-pin i and C_j is the capacitance of 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., a weighted half-perimeter wirelength [2], the length of a Rectilinear Minimum Spanning Tree (RMST), the length of a minimum single-trunk Steiner tree or the length of a Rectilinear Steiner Minimum Tree (RSMT). Interconnect delays are computed as rcL^2 where L is the Manhattan length of a timing edge. Alternatively, we use Elmore delay model which entails a Steiner tree computation and is therefore more expensive.

3 Continuous Min-Max Placement

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

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

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.[6] Common edge delay models can be based on linear/quadratic edge wirelength or Elmore delay.

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

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

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 by constant-delay edges to all timing sources and sinks resp. 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.

3.1 Generic minimization of Φ by re-weighting

Given *edge weights* $w_{ij} \geq 0$ on the timing graph, we *minimize* the following MAX-based objective function[7]

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

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.[8] 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 Formula 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, which establishes monotonic convergence.

Lemma 4 Given (i) an arbitrary set $w_{ij} \geq 0$ with at least one non-zero, 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 $d'_{ij} \leq d_{ij} - \varepsilon$.

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 $C \leq \max_{ij}(d_{ij} - \varepsilon)/d_{ij} < 1$. 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. $\square$

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

[5]With a generic STA implementation, more accurate models or black-box delay calculators can be used when affordable in terms of runtime.

[6]The functions $d_{ij}()$ could also depend on locations of cells neighboring with i and j to better model delays [17].

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

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

found for which $\delta^{(k+1)} \leq \delta^{(k)}$. 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 $\leq \delta^{(k)}$ with respect to the previous placement.

Lemma 6 All timing constraints are satisfied if

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

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

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

□

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*)

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

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

In order to ensure Inequality (4), we note that $d_{ij}^{(k+1)} \leq \delta^{(k+1)} / w_{ij}^{(k+1)}$ by definition of $\delta^{(k+1)}$ and $\delta^{(k+1)} \leq \delta^{(k)}$ 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.

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

The max-terms in this formula are called "criticalities" and can be computed using static timing analysis, which is especially efficient in the case 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 by little, $\delta^{(k)}$ is approximately constant, and so are the values d_{ij}^{k}. We can now rewrite the MAX-based objective function as

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

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

3.2 Interpretations of re-weighting and comparisons to known results

In the proposed iteration, let us define the *timing criticality* of an edge as timing criticality of the most critical path passing through the edge, measured by its contribution to Φ, i.e., $\kappa_{ij}^{(k)} = \max_{\pi' \ni e_{ij}} t_{\pi'}^{(k)} / c_{\pi'}$. It can be viewed as the multiplicative version of the traditional negative slack [15]. 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 *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 *by only as much as they need* to cease being critical and allow non-critical edges to elongate 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*). Improvement of the re-weighted objective will address critical edges and thus improve Φ.

Multiplication by relative edge delays is somewhat counter-intuitive because it gives *shorter edges* on critical paths *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).After being multiplied by edge criticalities, the new weights (i) encourage decreasing delays of critical edges by only as much as they need to become non-critical, and (ii) allow increasing delays of non-critical edges by only as much as they can without becoming critical.

Loosely speaking, the work in [28] mentions the κ term, but computes it for vertices rather than edges. However, neither [28], nor [10] have the ρ term, which seems to be what keeps their delay re-budgetings heuristic.

3.3 Lower-level minimization

We note that minimization of the MAX-based objective can be performed by linear or non-linear programming [18] depending on specific delay models. In fact, for the linear-wirelength delay objectives, the LP formulation is solvable in linear time (by a result of Megiddo ca 1992). We implemented a simple and extremely fast algorithm in which vertices are traversed in an arbitrary order and placed in locally-optimal locations. Such a pass cannot increase the objective, which implies monotonic convergence. Given that most vertices are adjacent to very few other vertices (sparcity), every pass has linear runtime. Passes are repeated until the objective function improves by less than, e.g., 0.1%. Very few passes are required in practice because of the relatively few stages of combinational logic on critical paths.

Non-linear delay models can be linearized [26].

4 Embedding Min-max Placement Into a Top-Down Placement Flow

Below we sketch a combined algorithm that attempts to simultaneously minimize cycle-time and half-perimeter wirelength/congestion. It starts by a call to continuous min-max placement that returns cell locations optimizing worst-slack, as well as actual cell slacks in that placement. That information is translated into pre-assignments for the subsequent partitioning runs. Intuitively, the cells with worst slacks should be selected and pre-assigned in the partitions where the continuous formulation placed them. Since the continuous min-max placement optimizes slack, the slack cannot improve during further top-down placement when the same continuous min-max placement will be used only with more constraints. 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 do 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 cells is assigned into a "wrong" partition. The lower (i.e., the worse) the score, the more important it is to pre-assign the cell into the partition containing its continuous location. Cells are sorted in the increasing order of goodness scores and those with positive goodness are considered "good enough" not be pre-assigned before calling a partitioner.

It is very important not to pre-assign too many cells before calling a partitioner, otherwise, half-perimeter wirelength and congestion of resulting placement can increase. Therefore our combined is controlled by two parameters that further limit the number of cell selected to be assigned at any give level. Both limits are in terms of % total area of movable cells. One applies to the whole layout, the other to individual blocks. Once all movable cells in the layout are sorted by their scores, those cells are traversed in the order of increasing goodness and marked as for being pre-assigned, and their areas accumulated.[9] This traversal continues until the total area of marked cells reaches the global area limit (% of the total area of all movable cells).

Subsequently, when a particular block is about to be partitioned, those of its cells that were marked for pre-assignment are traversed again in the order of increasing goodness scores, pre-assigned to partitions based on their continuous location, and their areas accumulated. This traversal continues until the accumulated total reaches the area limit for the block.

In addition to pre-assigning cells based on the continuous locations and cell slacks, we re-use those cell locations for more accurate terminal propagation. In many cases, this improves both half-perimeter wirelength and cycle time.

[9]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 it suffices to call `sort()` from the Standard Template Library.

5 Implementation details

Our implementation CapoT is based on the Capo placer reported in [4] and runs just like Capo unless the command-line parameter `-td` is specified. The general architecture somewhat reminds the "slack-graph" approach from [5], with its separation of concerns between delay calculation, static timing analysis and placement optimization. As explained in Section 4, we additionally separate continuous min-max placement from top-down placement.

5.1 Our Placer Implementation

In the timing-driven regime, CapoT makes calls to our continuous min-max placer TDplace, which interfaces with our Static Timing Analyzer (STA). The results returned by TDplace affect the construction of partitioning instances in CapoT as described in 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 store elements (latches and flip-flops) are marked in the timing graph. The directed graph is traversed by a Depth-First Search, and back-edges that cause purely-combinational (i.e., not containing store elements) cycles are removed. STA maintains an array of edges in topological order and edges that create purely-combinational cycles are not added to that array. STA also maintains *AATs@sources* and *RATs@sinks*. Store elements are considered as sources and sinks simultaneously. The delays of timing edges in STA are produced by a delay calculator using the latest available placement information. STA performs classical static analysis with two topological traversals and slack computations, as described in Subsection 2.2. It also computes *criticalities* from slacks, in the assumption 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). From criticalities, STA computes edge weights for use in min-max-weighted delay placement algorithms. We charge both interconnect delays and load-dependent gate delays to edges of the timing graph because gate delays are computed for each driver pin, thus the possibility to account for non-zero vertex delays in our timing analysis is not used.

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 the initial location in its geometric center. After that, 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.

While performing continuous placement, TDplace iterates gate/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. Thanks to the proven monotonic convergence of both iterations, convergence criteria are fairly straightforward — each iteration is stopped when its objective function changes by less than 0.1%. In practice this ensures sufficient accuracy for large-scale VLSI placement, and keeps the overall CPU budgets low.

5.2 Experimental Results

We evaluated the proposed algorithms in a simplified framework with linear delay model. Our circuits benchmarks are described in Table 1.

Design Name	Number of Cells	Number of Nets	Production Year
D1	6390	8033	1998
D2	20449	21230	1998
D3	40349	42487	1999
D4	58987	59922	1999

Table 1: Test cases used in the experiments.

The “no timing” configuration reported in Table 2 is the default configuration of Capo/CapoT, as reported in [3], followed by greedy overlap removal, greedy improvement of cell orientations and sliding-window improvement using branch-and-bound [4]. Greedy overlap removal and improvement of cell orientations took negligible time (there were very few or no cells overlaps). The sliding-window algorithm took up to 10% of the total time and typically improved the wirelength by several percent. We did not use the `-fast` configuration described in [3], but point out that it is clearly possible to speed-up all runs by a factor of 2 or more by using it and also skipping the post-processing steps described above. This would entail the increase of wirelength by at most several percent.

Given that Capo and CapoT return different placement solution every time they are launched, we collected results over ten independent starts in each configuration. While runtimes were fairly stable, we observed a significant variance of final wirelength and an even greater variance of final longest-path delay. In order to reach statistically significant conclusions, we averaged each type of readings in Table 2 over ten independent random starts of CapoT.

The results clearly indicate that, the two proposed schemes successfully improve circuit delay compared to the “no timing” configuration. The increase in wirelength is modest. While the timing configurations tend to require up to 60% more time, this appears within the expectations of our industrial colleagues.

On the largest design, the continuous(analytical) minimization of path delay lasted for 62 iterations, roughly 0.4 sec per iteration. We empirically observed a linear convergence rate. Clearly, finding optimization algorithms with faster convergence rate is an open direction for future research.

6 Conclusions

Our work proposed a new global timing-driven placement algorithm and evaluated it on a set of recent circuit benchmarks. The main contribution of this paper is to global timing-driven placement, and even without a detailed placer, we were able to demonstrate superior results on several industrial benchmarks. The proposed algorithms are rather flexible and can be adapted to various delay formulations and types of placers. In particular, they should be applicable with non-partitioning driven placers popular among some EDA developers.

As the feature sizes decrease, interconnect delays begin to dominate gate delays. Therefore, the improvements provided by our algorithms should increase with every technology step. Moreover, because of their generic (mathematical) nature, their applicability is not restricted to silicon circuits.

References

[1] M. Burstein, “Timing Influenced Layout Design”, *DAC ‘85*, pp. 124-130.

[2] A. E. Caldwell, A. B. Kahng, S. Mantik, I. Markov, A. Zelikovsky, “On Wirelength Estimations for Row-based Placement”, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol.18, (no.9), IEEE, Sept. 1999

[3] A. E. Caldwell, A. B. Kahng and I. Markov, “Optimal Partitioning and End-case Placement For Standard-Cell Layout”, to appear in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, November, 2000.

[4] A. E. Caldwell, A. B. Kahng and I. Markov, “Can Recursive Bisection Alone Produce Routable Placements?”, *DAC 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 Circuits and Systems”, *Proc. Asia-Pacific Conf. on Computer-Aided Design ‘94,* 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. ‘90*, 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. Int. Symp. Physical Design 2001*, pp. 24-29.

[8] A. E. Dunlop and B. W. Kernighan, “A Procedure for Placement of Standard Cell VLSI Circuits”, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 4(1) (1985), pp. 92-98

[9] H. Eisenmann and F. Johaness, “Generic Global Placement and Floorplanning”, *Proc. Design Automation Conf. ‘98*, pp. 269-274.

[10] J. Frankle, “Iterative and Adaptive Slack Allocation for Performance-driven Layout and FPGA routing”, *Proc. Design Automation Conf. ‘92*, pp. 539-542.

Design	No timing			Scheme 1			Scheme 2		
Name	HPWL	Delay	CPU	HPWL	Delay	CPU	HPWL	Delay	CPU
D1	7.52e8	2.10e6	**102.3**	7.59e8	**1.85e6**	124.5	**7.50e8**	1.93e6	131.9
D2	**2.44e8**	6.85e5	**356.8**	2.63e8	**6.31e5**	501.6	2.68e8	**6.31e5**	542.7
D3	**3.01e8**	6.32e5	**722.8**	3.14e8	6.22e5	951.2	3.15e8	**5.53e5**	1053
D4	2.11e9	3.08e6	**3246**	2.07e9	2.68e6	3980	2.06e9	**2.56e6**	4193

Table 2: Path-delay minimization results: **averages** of ten independent starts. Bests follow the same trend.

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

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

[13] 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.

[14] P. S. Hauge, R. Nair and E. J. Yoffa, "Circuit Placement For Predictable Performance", *Proc. Intl. Conf. Computer-Aided Design '87*, pp. 88-91.

[15] R. B. Hitchcock, Sr., G. L. Smith and D. D. Cheng, "Timing Analysis Of Computer Hardware", *IBM J. Res. Develop.*, Vol. 26, No. 1, pp. 100-108, 1982.

[16] M. A. B. Jackson, A. Srinivasan and E. S. Kuh, "A Fast Algorithm For Performance-driven Placement", *Proc. Intl. Cong. Computer-Aided Design*, pp. 328-331.

[17] T. Koide, M. Ono, S. Wakabayashi, Y. Nishimaru, "ParPOPINS: a timing-driven parallel placement method with the Elmore delay model for row based VLSIs". *Proc. Asia and South Pacific Design Automation Conf. '97*, pp.133-40.

[18] D. G. Luenberger, "Linear and Nonlinear Programming", 2nd Edition, *Addison Wesley*, 1984.

[19] W. K. Luk, "A Fast Physical Constraint Generator for Timing Driven Layout", *Proc. Design Automation Conf. '91*, pp. 626-631.

[20] M. Marek-Sadowska and S. P. Lin, "Timing-Driven layout Of Cell-based ICs", *VLSI Systems Design*, pp.63-73, May 1986.

[21] M. Marek-Sadowska and S. P. Lin, "Timing driven placement". *Proc. Intl. Conf. on Computer-Aided Design '89*, pp. 94-7.

[22] R. Nair, C. L. Berman, P. S. Hauge and E. J. Yoffa, "Generation of Performance Constraints for Layout", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 8, pp. 860-874, August 1989.

[23] 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.

[24] L. Pileggi, "Timing Metrics For Physical Design of Deep Submicron Technologies", *Proc. Int. Symp. Physical Design, 1998* pp. 28-33.

[25] B. M. Riess and G. G. Ettelt, "Speed: Fast and Efficient Timing Driven Placement", *Proc. ISCAS '95*, pp. 377-380.

[26] M. Sarrafzadeh, D. Knoll and G. Tellez, "Unification of Budgeting and Placement", In *Proc. Design Automation Conf. '97*, 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 S. Shragowitz, "Bounds On Net Delays", *IEEE Trans. on Circuits and Systems*, vol. 39, no. 11, November 1992, pp. 815-824.