Gate Sizing for Large Cell-Based Designs

Stephan Held

Research Institute for Discrete Mathematics, University of Bonn,
Lennéstr. 2, 53113 Bonn, Germany
Email: held@or.uni-bonn.de

***Abstract*—Today, many chips are designed with predefined discrete cell libraries. In this paper we present a new fast gate sizing algorithm that works natively with discrete cell choices and realistic timing models. The approach iteratively assigns signal slew targets to all source pins of the chip and chooses discrete layouts of minimum size preserving the slew targets. Using slew targets instead of delay budgets, accurate estimates for the input slews are available during the sizing step. Slew targets are updated by an estimate of the local slew gradient.**

To demonstrate the effectiveness, we propose a new heuristic to estimate lower bounds for the worst path delay. On average, we violate these bounds by 6%. A subsequent local search decreases this gap quickly to 2%. This two-stage approach is capable of sizing designs with more than 5.8 million cells within 2.5 hours and thus helping to decrease turn-around times of multi-million cell designs.

I. Introduction

Continuously decreasing feature sizes allow more and more cells on a single chip. On ASIC designs, the number of cells reached 6 million already in 130 nm technology and more than 10 million in 65 nm. Therefore, efficient gate sizing algorithms for large instances are of high interest.

The mathematically best-founded approaches for the gate sizing problem rely on geometric/convex programming and assume continuously sizeable cells or transistors. Delay and slew functions are modeled as posynomials [6]. A geometric programming formulation finding a global optimum was given in [13]. In [3], Lagrangian relaxation and the subgradient method were applied to a simpler class of posynomials, but larger instance sizes. This approach was recently revisited and enhanced in [17]. In the last years, gate sizing has been extended to optimize statistical timing constraints [1], [14], [15], and [11]. Besides numerical and running time challenges on large instances, the apparently optimal geometric programming approaches lack from the need to approximate delays by posynomials and their continuity requirement.

However, many chips, in particular ASIC designs, are designed with predefined discrete cell libraries. Due to the increasing complexity of design rule checking, this approach will rather expand with decreasing feature sizes. Often such libraries are too sparse to allow for fast and simple nearest rounding [10]. In [10] a dynamic programming approach for improved rounding was proposed, but its running time will be rather too high for large designs. In fact, an efficient rounding procedure providing a quality guarantee is not yet known.

The probably fastest approaches for gate sizing in practice are delay budget approaches such as [4], [5], and [8]. These approaches repeatedly (re-)distribute delay budgets to all cells. Then for each cell the minimum size meeting the budgets is determined independently from other cells. Newer budgeting approaches can be found in [16], [12], and [7]. Using simplified delay models and continuously sizable cells, as for the geometric programming formulations, optimality can be proven in some cases, e.g. [16]. For delay budgeting linear/nonlinear programming, or minimum cost flow algorithms are applied.

Due to the mentioned problems of simplified delay models, continuity assumptions, numerical and running time challenges, we propose a different approach that works natively with a discrete library and given delay models. First, we run a fast global gate sizing that falls into the category of delay budgeting heuristics. But, instead of delay budgets we distribute slew (transition time) targets, which is equivalent assuming that delay and slew functions are strictly monotone increasing in the input slew and downstream capacitance. However, slew targets enable a more accurate cell sizing. The cell size preserving a slew target depends basically on the load capacitance and the input slews. During the sizing step we traverse the netlist reverse to the signal direction, so that the downstream RC-networks are known for each sized cell. The input slews are not known, as the preceeding cells are still to be sized. Here, the slew targets at the predecessors provide the capability to compute good estimates for the input slews.

Then, after the fast global sizing, we run a local search on the most critical paths to guide the worst path delay further into a local optimum.

To demonstrate the effectiveness of our approach, we propose a new heuristic to estimate a lower bound for the delay on the most critical path in the design. By comparing with these bounds on several industrial ASIC designs, we show that we exceed this bound by only 6% on average. A subsequent local search reduces this gap quickly to 2%. Furthermore we demonstrate the extreme speed of our algorithms by comparing our gate sizing algorithm with an industrial tool.

The rest of the paper is organized as follows. In Section II we briefly describe the gate sizing problem and introduce our notation. The fast global gate sizing is explained in detail in Section III. A short description of the local search is given in Section IV. In Section V the lower bounds for the worst path delay are explained. Experimental results are given in Section VI. We conclude the paper with some discussion and outlook.

II. The Gate Sizing Problem

Let $\mathcal{C}$ be the set of *cells*, and P the set of pins on a chip. By $P(c)$ we denote the set of pins of a cell $c \in \mathcal{C}$, and by $P_{in}(c)$ and $P_{out}(c)$ its input pins and output pins. We assume that $\mathcal{C}$ contains one fixed cell to represent primary inputs and outputs of the chip.

The timing constraints we consider are based on static timing analysis [9] with slew propagation. The signal propagation is described by a directed timing graph $G^{\mathcal{T}}$ on the set of pins ($V(G^{\mathcal{T}}) = P$). For easier notation, we assume that a single signal with latest arrival time $at(p)$, slew $slew(p)$, and earliest required arrival time $rat(p)$ is propagated to each pin p in the design. The slack $slack(p) := rat(p) - at(p)$ indicates the criticality of the timing signal in p. If $slack(p) \geq 0$ the paths through p could be delayed by that amount without violating the timing constraints. Otherwise, at least the most critical path through p needs to be accelerated by $-slack(p)$. See [9] for details on static signal propagation.

We point out that our algorithms can easily be used for delay models with arbitrary signal waveforms, maintaining single valued slew targets to guide the waveforms. For simpler notation we assume the propagation of single slew values throughout this paper.

The (discrete) gate or cell sizing problem consists of assigning each individual cell $c \in \mathcal{C}$ to a library cell $B \in \mathcal{B}$ from a discrete cell-library $\mathcal{B}$. By $[c] \subset \mathcal{B}$ we denote the set of logically equivalent library cells to which c may be assigned. In typical formulations the assignment should be chosen such that some objective function, e.g. the total power or area consumption, is minimized while all timing constraints are met, i.e. $\text{slack}(p) \geq 0$ for all $p \in P$.

Besides slacks, slew limits, $\text{slew}(q) \leq \text{slewlim}(q)$ for all input pins $q \in P_{in}(c), c \in \mathcal{C}$, and capacitance limits, $\text{downcap}(p) \leq \text{caplim}(p)$ for all output pins $p \in P_{out}(c), c \in \mathcal{C}$, must be preserved.

As gate sizing is mostly applied when no feasible solution exists, a practical objective is to maximize the worst slack, but to also push less critical negative slacks towards zero. Such a solution reduces the need for other more resource-consuming optimization routines.

III. Fast Global Gate Sizing

The slew targeting algorithm is summarized in Algorithm 1. It iteratively assigns a slew target $\text{slewt}(p)$ to all output pins $p \in P_{out}(c), c \in \mathcal{C}$. In line 1 slew targets are initialized such that the slew limits will just be met at subsequent sinks (accounting for the slew degradations on the wires).

Then, in line 3 cell sizes are chosen such that the slew targets are met. The slew targets are updated based on an estimate of the slew gradient that guides the cell to a locally optimum solution in line 5.

To bound the running time, the algorithm avoids incremental timing updates. Instead, the timing is updated for the complete design by a timing oracle in line 4 once per iteration. This way the algorithm could also take full advantage of a parallel timing engine. The subroutines in lines 3 and 5 can be parallelized as well.

```
1: Initialize slew targets for all cell output pins
2: **repeat**
3:     Assign cells to library cells (Section III-A)
4:     Timing analysis
5:     Refine slew targets (Section III-B)
6: **until** Stopping criterion is met
7: Return best assignment of all iterations
```

Algorithm 1: Fast Gate Sizing

The stopping criterion in line 6 is met when the current cell assignment compared to the last iteration

1) worsens the worst slack, and
2) increases a weighted sum of the absolute worst negative slack (WS), the absolute sum of negative slacks (SNS) divided by the number of endpoints, and the average cell area.

If the criterion is met, the assignment of the previous iteration, which achieves the best present objective value, is recovered. The criterion is never met while the worst slack is improving. A lower worst slack is tolerated if it is accompanied by sufficiently large gains in the average negative slack or average cell area. We now explain the cell assignment and slew target refinement in detail.

A. Assigning Cells to Library Cells

Cells are assigned to the smallest equivalent library cell such that the slew targets at all of their output pins (usually there is only one output pin) are met. This choice depends on the input slews and output loads, respectively the layout and sink pin capacitances of the output nets. These values, in turn, depend on the sizes of other cells.

As the downstream network usually has a bigger impact on the cell timing than the input slew, it is be preferable to know the exact downstream capacitances when sizing a cell. Let the cell graph $G_{\mathcal{C}}$ be defined as the directed graph that contains a vertex for each $c \in \mathcal{C}$ and an edge $e = (c', c) \in E(G_{\mathcal{C}})$ if a pin $p' \in P_{out}(c')$ is connected with a pin $q \in P_{in}(c)$. Cells are processed in order of decreasing longest distance (under unit edge lengths) from a register in the acyclic subgraph, which arises from $G_{\mathcal{C}}$ by omitting edges entering register vertices.

1) Input Slew Estimation: When a cell $c \in \mathcal{C}$ is sized, the successor sizes are already known except for registers as successors. The input slews $\text{slew}(q), q \in P_{in}(c)$, depend heavily on the unknown predecessor sizes. However, the slew targets at the predecessor pins help us to estimate reasonable input slews.

For a pin $q \in P_{in}(c)$ with predecessor $p' \in \delta^-_{G^{\mathcal{T}}}(q)$, as in Fig. 1, we estimate the final slew in p' as the weighted sum

$$\text{est\_slew}(p') := \theta\, \text{slewt}(p') - (1-\theta)\, \text{slew}(p').$$

The weighting factor $\theta \in [0, 1]$ shifts from pure predecessor target ($\theta = 1$) to "real" timing analysis ($\theta = 0$) with each global iteration. In the beginning, the predecessor slews will

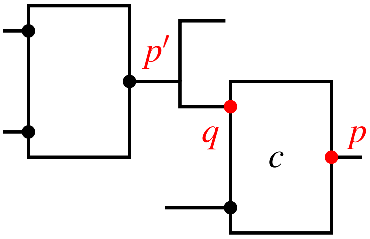

Fig. 1. A cell c and a predecessor.

end up closer to $\text{est\_slew}(p')$ than to $\text{slew}(p')$. Later, when a change of the predecessor size is less likely, the computed slew values dominate the formula.

To obtain an estimate for $\text{slew}(q)$, we need to add the slew degradation on the wire $\text{slew\_degrad}(p', q)$ to $\text{est\_slew}(p')$. When the pin capacitance of q changes, $\text{slew\_degrad}(p', q)$ will change too. This change can be approximated quickly by an RC-delay model for the wire.

2) Sizing: Given the estimated predecessor slews and the layout of the output network, the minimum cell size for c preserving the slew targets (and load capacitance limits) can be computed performing a local timing analysis through c and its downstream wires for all available library cells in $[c]$. Note that a level of cells with equal distance labels can be sized independently and thus in parallel.

In the special case, where the delay and slew propagation through a cell is parameterized by the two parameters load capacitance and input slews, this sizing step can be accelerated via table look-up. The mapping to the minimum cell size can be pre-computed for each input/output pin pair in the cell-library for finite sets of equidistant support points for each of the three parameters:
1) load capacitance, 2) input slew, and 3) slew target.

For more complex delay models, e.g. current source models, such tables would probably become too large and inefficient.

In general, the cell graph is cyclic and the assignment algorithm needs to traverse the cells several times until no more registers are altered. In practice, we observed only a few re-assigned cells even in the second iteration. To speed up the overall algorithm, we perform only a single iteration, leaving it to the next global iteration to remove sub-optimal or illegal assignments.

B. Refining Slew Targets

In Step 5 of Algorithm 1, the slew target $\text{slewt}(p)$ is refined for each output pin p of a cell c based on what we call the *global* and *local criticality* of p. The *global criticality* $slk^+(p)$ is simply the slack

$$slk^+(p) := slack(p) = rat(p) - at(p)$$

at p. The pin p is *globally critical* if $slk^+(p) < 0$.

```
Input: A cell $c$, $\gamma$, max_change, and an iteration number $k$
 1: $\theta_k \leftarrow 1/\log(k + const)$;
 2: $slk^-(c) \leftarrow \min\{slack(p') | p' \in \Gamma^-_{G^{\mathcal{T}}}(P_{in}(c))\}$;
 3: for all $p \in P_{out}(c)$ do
 4:     $slk^+(p) \leftarrow \text{slack}(p)$;
 5:     $lc(p) \leftarrow \max\{slk^+(p) - slk^-(c), 0\}$;
 6:     if $slk^+(p) < 0$ and $lc(p) = 0$ then
 7:         $\Delta\,\text{slewt} \leftarrow -\min\{\theta_k \cdot \gamma \cdot |slk^+(p)|, \text{max\_change}\}$;

 8:     else
 9:         $slk^+(p) \leftarrow \max\{slk^+(p), lc(p)\}$;
10:         $\Delta\,\text{slewt} \leftarrow \min\{\theta_k \cdot \gamma \cdot |slk^+(p)|, \text{max\_change}\}$;
11:     end if
12:     $\text{slewt}(p) \leftarrow \text{slewt}(p) + \Delta\,\text{slewt}$;
13:     Project $\text{slewt}(p)$ into $[\underline{\text{slewt}}([p]), \text{slewlim}(p)]$;
14: end for
```

Algorithm 2: Fast Gate Sizing—Refining Slew Targets

The local criticality should indicate whether the worst of the slacks in p and any direct predecessor pin of c can be improved either by accelerating c, i.e. by decreasing $\text{slewt}(p)$, or by decreasing the input pin capacitances of c, i.e. increasing $\text{slewt}(p)$. First, we define the predecessor criticality of the cell c by

$$slk^-(c) := \min\{slack(p') \mid p' \text{ direct predecessor pin of } c\}.$$

Obviously, $slk^+(p) \geq slk^-(c)$ for all cells but registers, since a path that determines the slack in p must contain one of the predecessor pins. Registers may have smaller output slacks than their predecessors, as inputs and outputs may belong to different data paths. We define the local criticality $lc(p)$ of p by

$$lc(p) := \max\{slk^+(p) - slk^-(c), 0\}.$$

If $lc(p) = 0$ then p is either located on a worst-slack path through a most critical predecessor of c, or p is an output pin of a register whose output path is at least as critical as any path through its predecessors. We call p *locally critical* if $lc(p) = 0$.

Algorithm 2 shows how the slew targets of the cell c are updated in a global iteration $k \in \mathbb{N}$. If p is globally and locally critical, we decrease $\text{slewt}(p)$ by subtracting a number that is proportional to $|slk^+(p)|$, but does not exceed some constant max_change (lines 7 and 12). Otherwise we increase $\text{slewt}(p)$ by adding a number that is proportional to $\max\{lc(p), slk^+(p)\}$ (lines 9, 10, and 12).

The constant $\gamma \in \mathbb{R}$ can be considered as an estimate for the virtual term $\frac{\partial \text{slew}(p)}{\partial slk^+}$. Thus, if $\text{slewt}(p)$ is tightened and assuming linearity, $\gamma \cdot |slk^+|$ represents the required slew change to reach a non-negative slack in p. If $\text{slewt}(p)$ is relaxed, $\gamma \cdot \max\{slk^+, lc\}$ expresses the required slew change, either to align the slack in p with the worst predecessor slack (if $slk^+ \leq lc$), or to decrease the slack to zero (if $slk^+ > lc \geq 0$). As we modify all cells in each iteration, γ should be just a fraction of $\frac{\partial \text{slew}(p)}{\partial slk^+}$. In practice, we simply set γ to a small constant. The multiplier θ_k is a damping factor that is intended to reduce potential oscillation.

In line 13 the slew target is projected into the feasible range. Here, the maximum slew limit $\text{slewlim}(p)$ for the output pin p is induced by the attached sinks:

$$\text{slewlim}(p) := \\ \min\{\text{slewlim}(q) - \text{slew\_degrad}(p, q) \mid q \in \delta^+_{G^{\mathcal{T}}}(p)\},$$

where $\text{slew\_degrad}(p, q)$ is an estimate of the slew degradation as in Section III-A1. The value $\underline{\text{slewt}}([p])$ denotes the lowest possible slew, that is achievable with any equivalent cell, given the current load capacitance and input slew. It prevents unrealistically small slew targets.

1) Enhanced Slew Target Refinement: As described so far, Algorithm 1 yields already good results. But, in some cases it leads to overloaded cells that cannot be enlarged further, or to locally non-critical cells that cannot be downsized sufficiently because of too large successors. In an entire timing optimization flow, gate sizing is alternated with repeater insertion absorbing such situations. However, this can lead to unnecessary repeaters.

In such situations the slew targets of successor cells should be relaxed to enable smaller sizes. Slew targets of locally uncritical cells are relaxed automatically by Algorithm 2. Now, when refining the slew target of a locally critical pin $p \in P_{out}(c), c \in \mathcal{C}$, we consider the largest estimated slew $\text{est\_slew}(p')$ of a most critical predecessor pin

$$p' \in \arg\max\{\text{est\_slew}(r) \mid r \in \Gamma^-_{G^{\mathcal{T}}}(P_{in}(c)), \\ \text{slack}(r) = slk^-(c)\}.$$

If $\text{est\_slew}(p') > \text{slewt}(p)$, we increase the slew target in p by

$$\text{slewt}(p) := \lambda \cdot \text{slewt}(p) + (1 - \lambda) \cdot \text{est\_slew}(p')$$

with $0 < \lambda \leq 1$. The effect of an extraordinary high value of $\text{est\_slew}(p')$ declines exponentially in the number of subsequent cell stages. In our experiments in Section VI we have chosen $\lambda = 0.7$. Note that less critical predecessors $r \in \Gamma^-_{G^{\mathcal{T}}}(P_{in}(c))$ with $\text{slack}(r) > slk^-(c)$ are not considered for relaxing $\text{slewt}(p)$.

To enable the enhanced slew target computation, the cells must be traversed in signal direction, reverse to the sizing step. Again, the slew targets of all cells in a level of equal longest path distance from a register can be updated in parallel.

IV. Local Search Gate Sizing

The local search is applied to further improve the result of the fast global gate sizing. Iteratively, it collects a small set of cells attached to the most critical nets and sizes them one after another to their local optimum based on more accurate slack evaluations. The next iteration starts with collecting cells from scratch.

First, for collecting cells, we traverse all nets by increasing slack at their source pins and select all cells that are attached to the current net and to all nets that have the same slack at their sources. As soon as more than $K \in \mathbb{N}$ cells are collected the traversal of the nets stops. Note that this procedure will collect at least the cells on the most critical paths and their direct successors for any choice of K. It is important to select not only the critical cells but all cells attached to a net, because the pin capacitances of the noncritical cells affect the timing. In our experiments we have chosen K as 0.2% of the total number of cells in the design.

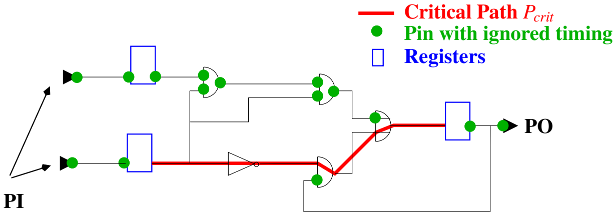

Fig. 2. Lower delay bound computation

Then, for sizing, the collected cells are traversed in the order of decreasing longest path distance from a register as in Section III-A. A cell $c \in \mathcal{C}$ is assigned to a library cell $B \in [c]$ of minimum size such that

$$\min\{0, slk^-(c), slk^+(p) \mid p \in P_{out}(c)\}$$

is maximized. The slacks are computed by an exact analysis within a small neighborhood around c. The neighborhood contains its predecessor cells and all direct successors of c and its predecessor pins.

The algorithm stops when the worst slack could not be improved in the last iteration. For running time reasons, this criterion could be modified to stop when the worst slack improvement falls below some threshold or when some maximum number of iterations is reached.

V. Lower Delay Bounds

To demonstrate the quality of our gate sizing approach, we compare the results of the two gate sizing algorithms with estimated lower bounds for the achievable delay of the most critical path P^{crit} . For this purpose, we size all cells in the design to a minimum size such that capacitance limits are obeyed. We ignore and remove any arrival times and required arrival times on pins that are not located on P^{crit} as shown in Fig. 2. This way the arrival times and required arrival times on P^{crit} are independent from any other input or output.

Now P^{crit} is sized by running the local search algorithm from Section IV w.r.t. to an infinite slack target (instead of '0'), until no more improvement can be found. Note that cells on any side branching from P^{crit} are set to a minimum possible size and therefore have a minimum impact on the load capacitances of the cells on P^{crit}. The following theorem follows from standard geometric programming arguments.

Theorem 1: Assuming posynomial delay functions and continuously sizable cells, The local search computes the minimum possible delay for the path P^{crit}.
By complete enumeration on several thousand paths from different designs and libraries, and with lengths bounded by 10, we verified empirically that this holds for the discrete problem in practice as well. Note that the quality of this bound depends on the state of the design. With ideal zero wires it will

Chip	**Fast Gate Sizing**			**Local Search Sizing**		
	Delay	Bound	÷	Delay	Bound	÷
Fazil	5.85	5.14	1.14	4.30	4.22	1.02
Franz	4.25	4.05	1.05	4.69	4.63	1.01
Lucius	1.20	1.15	1.04	1.60	1.55	1.03
Felix	2.21	1.95	1.13	2.15	1.93	1.11
Lucius45	0.69	0.61	1.13	0.63	0.62	1.02
Minyi	2.96	2.92	1.02	2.96	2.96	1.00
Maxim	2.49	2.37	1.05	2.47	2.41	1.02
Tara	1.54	1.43	1.07	1.48	1.46	1.01
Karsten	9.00	8.64	1.04	8.66	8.64	1.00
Fidelio	5.93	5.77	1.03	5.77	5.77	1.00
Arijan	3.25	3.20	1.01	3.20	3.20	1.00
David	4.96	4.73	1.05	4.86	4.73	1.03
Valentin	0.15	0.15	1.00	0.15	0.15	1.00
Trips	6.00	5.37	1.12	5.64	5.40	1.04
Avg.			**1.06**			**1.02**

TABLE I
DEVIATIONS FROM DELAY BOUNDS

(delays in nanoseconds)

be rather weak. In the next Section we will see that our gate sizing algorithms come close to this bound on placed designs with estimated wires.

VI. Experimental Results

We ran comprehensive experiments on 14 industrial ASIC designs, provided by our industrial partners at IBM. We thank the University of Texas at Austin for granting us access to the "Trips" data [2], our largest instance. The instances arise from four different technologies, namely 45, 65, 90, and 130 nm. The first three columns in Table II list the instances by technology and number of cells $|\mathcal{C}|$ in thousands. The instances range from small RLMs to full chips with almost 6 million cells. All instances are placed, and estimated global routes are used for wire extraction. The designs are buffered, so that results are not distorted by unavoidable capacitance and slew violations. But, positive slacks are not achievable in most cases. We ran all experiments w.r.t. a slack target of 0.25 nanoseconds, which was not achievable for any design.

Table I shows the deviations from the lower delay bounds on the critical paths according to Section V. For each chip the critical path after fast gate sizing and the delay of the most critical path after the local search are given, as well as their delay bounds. The columns labeled by "÷" show the ratio of the actual delays to their lower bounds. Note that the most critical paths and therefore the delay bounds can be different after fast gate sizing and after local search sizing. The average deviations after fast gate sizing and refine gate sizing are 6% and 2%, with a maximum final deviation of 11% on Felix. On many designs we can even close the gap. Here, the critical paths have only a few branchings or the delays are dominated by wires.

During the fast gate sizing the slacks usually improve rapidly during the first 3–4 iterations. Then, the worst slack sometimes starts oscillating slightly. Fig. 3 shows exemplary how the worst slack (WS), the sum of negative slacks (SNS), and the area evolve during the course of the fast gate sizing on the design David. The area consumption increases until iteration 7 and decreases slightly afterwards.

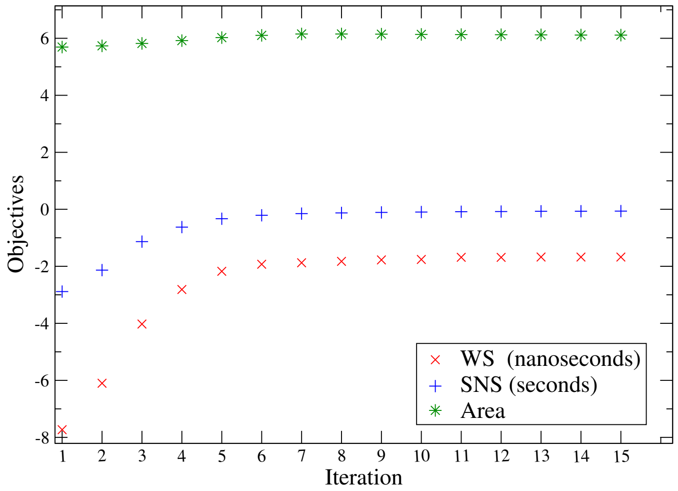

Fig. 3. Objectives during the course of the fast gate sizing on David (the area numbers are scaled linearly to fit into the figure)

Table II shows a comparison with a contemporary industrial gate sizing tool using the same underlying timing engine. The industrial tool is currently used for the design of high performance ASICs and processor blocks. Its approach is to start with a minimum area solution preserving capacitance and slew limits. Then, it incrementally improves the timing of the critical paths in an area efficient way, selecting a subset of the critical cells with a large ratio of their slack gain to their area increase (similar to [6] but selecting multiple cells per iteration). It is run until no further improvement is found.

Our combined algorithm achieves 0.09 nanoseconds (ns) better worst slacks (WS) and 61% lower sums of negative slacks (SNS in milliseconds) on average. The significantly better slack distributions are accompanied by an area increase of 7%. The better SNS enables efficient downsizing of branching cells from the critical paths and thus enables better worst slacks. Note that both algorithms were driven towards a slack target of 0.25 ns. Thus, the SNS can be negative even if the WS is not.

The running times in columns 7 and 14 were obtained in sequential runs on an Intel Xeon server with 3.0 GHz. On average our algorithm is faster by a factor of 2.17. But, the total running time over all designs is smaller by a factor of 5. For large instances the running time of the industrial tool grows up to 14 hours (on Valentin), while the running time of our algorithm grows to at most 2.5 hours for nearly 5.9 million cells (on Trips). About 1/3 of our running time is consumed by the local search. The maximum number of iterations for the fast gate sizing was restricted to 10. Within the fast gate sizing about 50% of the running time is consumed by the timing analysis in step 4 (in Algorithm 1). The running time of the local search refinement consists mostly of timing analysis.

			Industrial Tool				**New Combined Fast Gate Sizing and Local Search**							
Chip	**Tech.**	$\lvert\mathcal{C}\rvert$	**WS**	**SNS**	**Area**	**Time**	**WS**		**SNS**		**Area**		**Time**	
	nm	in K	ns	ms		h:m	ns	Δ	ms	÷		÷	h:m	÷
Fazil	130	64	−1.27	−31	574	0:03	−1.20	0.07	−19	0.60	612	1.06	0:01	0.54
Franz	90	69	0.09	−1	740	0:03	0.16	0.07	−1	0.45	747	1.01	0:02	0.55
Lucius	65	71	−0.48	−24	268	0:02	−0.34	0.14	−13	0.52	301	1.12	0:01	0.67
Felix	130	76	−0.64	−34	689	0:04	−0.56	0.08	−22	0.64	782	1.13	0:01	0.41
Lucius45	45	118	−1.04	−62	888	0:03	−0.94	0.10	−47	0.75	940	1.06	0:02	0.66
Minyi	65	245	−0.51	−11	4906	0:04	−0.48	0.02	−1	0.08	4880	0.99	0:03	0.69
Maxim	65	428	−1.41	−403	5466	0:42	−1.33	0.08	−247	0.61	6347	1.16	0:09	0.21
Tara	65	778	−0.51	−53	5808	0:26	−0.51	0.01	−12	0.23	5967	1.03	0:07	0.30
Karsten	130	3056	−3.76	−186	41106	1:03	−3.52	0.24	−65	0.35	41363	1.01	0:30	0.48
Fidelio	65	3694	−3.86	−3369	57370	1:26	−3.85	0.01	−1421	0.42	60641	1.06	0:46	0.54
Arijan	90	3754	−1.11	−635	63522	10:39	−0.84	0.28	−47	0.07	65710	1.03	1:04	0.10
David	90	4334	−1.44	−1311	60800	1:43	−1.51	−0.07	−209	0.16	62457	1.03	1:31	0.88
Valentin	90	5378	−1.62	−3880	79362	14:49	−1.48	0.14	−1581	0.41	92464	1.17	1:57	0.13
Trips	130	5879	−2.15	−4563	69739	10:45	−2.08	0.07	−823	0.18	76407	1.10	2:35	0.24
Avg.								**0.09**		**0.39**		**1.07**		**0.46**

TABLE II
COMPARISON WITH AN INDUSTRIAL TOOL

VII. CONCLUSIONS & OUTLOOK

We have presented a new combined algorithm for the discrete gate sizing problem without specific assumptions on the underlying delay model. We demonstrated that gate sizing for multi-million cells can be solved within a few hours, while the worst path delays almost reach their lower bound. Parallelization has not yet been exploited and promises further speed-up.

The proposed estimates of lower bounds are generally useful to judge and improve the quality of discrete gate sizing algorithms and implementations. In fact, they helped us to eliminate several flaws from our tools.

REFERENCES

[1] Agarwal, A., Chopra, K., Blaauw, D., and Zolotov, V.: Circuit optimization using statistical static timing analysis. Proc. DAC (2005), 321–324.

[2] Burger, D., Keckler, S.W., McKinley, K.S., Dahlin, M., John, L.K., Lin, C. Moore, C.R., Burrill, J., Mcdonald, R.G., Yoder,W., and the TRIPS team: Scaling to the end of silicon with edge architectures. Computer 37 (7), 2004, 44–55.

[3] Chen, C.-P., Chu, C.C.N., and Wong, D.F.: Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation. IEEE Trans. on Computer-Aided Design 18 (7), 1999, 1014–1025.

[4] Chen, H.Y., and Kang, S.M.: iCOACH: a circuit optimization aid for CMOS high-performance circuits. Integration, the VLSI Journal 10, 1991, 185–212.

[5] Dai, Z.-J., and Asada, K.: MOSIZ: a two-step transistor sizing algorithm based on optimal timing assignment method for multi-stage complex gates. IEEE Custom Integrated Circuits Conference (1989), 17.3.1–17.3.4.

[6] Fishburn, J. and Dunlop, A.: TILOS: A posynomial programming approach to transistor sizing. Proc. ICCAD (1985). Digest of Technical Papers, 326–328.

[7] Ghiasi, S., Bozorgzadeh, E., Huang, P.-K., Jafari, R., and Sarrafzadeh, M.: A Unified Theory of Timing Budget Management. IEEE Trans. on Computer-Aided Design 25 (11), 2006, 2364–2375.

[8] Heusler, L.S., and Fichtner, W.: Transistor sizing for large combinational digital CMOS circuits. Integration, the VLSI Journal 10, 1991, 155–168.

[9] Hitchcock, R.B., Smith, G.L., and Cheng, D.D.: Timing Analysis of Computer Hardware. IBM Journal of Research and Development 26 (1), 1982, 100–105.

[10] Hu, S., Ketkar, M., and Hu J.: Gate Sizing For Cell Library-Based Designs. Proc. DAC (2007), 847–852.

[11] Khandelwal, V., and Srivastava, A.: Variability-Driven Formulation for Simultaneous Gate Sizing and Postsilicon Tunability Allocation. IEEE Trans. on Computer-Aided Design 27, 2008, 610–620.

[12] Kursun, E., Ghiasi, S., and Sarrafzadeh, M. : Transistor Level Budgeting for Power Optimization. Proc. of the 5th International Symp. on Quality Electronic Design (2004), 116–121.

[13] Sapatnekar, S.S., Rao, V.B., Vaidya, P.M., and Kang, S.-M.: An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization. IEEE Trans. on Computer-Aided Design 12 (11), 1993, 1621–1634.

[14] Singh, J., Nookala, V., Luo, Z.-Q., and Sapatnekar, S.S.: Robust gate sizing by geometric programming. Proc. DAC (2005), 315–320.

[15] Sinha, D., Shenoy, N.V., and Zhou, H.: Statistical Timing Yield Optimization by Gate Sizing. IEEE Transactions on VLSI Systems 14 (10), 2006, 1140–1146.

[16] Sundararajan, V., Sapatnekar, S.S., and Parhi, K.K.: Fast and exact transistor sizing based on iterative relaxation. IEEE Trans. on Computer-Aided Design 21 (5), 2002, 568 – 581.

[17] Wang, J., Das, D., and Zhou, H.: Gate sizing by Lagrangian relaxation revisited. Proc. ICCAD (2007), 111–118.