Gate Sizing for Cell-Library-Based Designs

Shiyan Hu, *Member, IEEE*, Mahesh Ketkar, and Jiang Hu, *Senior Member, IEEE*

***Abstract*—With increasing time-to-market pressure and shortening semiconductor product cycles, more and more chips are being designed with library-based methodologies. In spite of this shift, the problem of discrete gate sizing has received significantly less attention than its continuous counterpart. On the other hand, cell sizes of many realistic libraries are sparse, for example, geometrically spaced, which makes the nearest rounding approach inapplicable as large timing violations may be introduced. Therefore, it is highly desirable to design an effective algorithm to handle this discrete gate-sizing problem. Such an algorithm is proposed in this paper. The algorithm is a continuous-solution-guided dynamic-programming-like approach. A set of novel techniques, such as locality-sensitive-hashing-based solution pruning, is also proposed to accelerate the algorithm. Our experimental results demonstrate that 1) the nearest rounding approach often leads to large timing violations and 2) compared to the well-known Coudert's approach, the new algorithm saves up to 21% in area cost while still satisfying the timing constraint.**

***Index Terms*—Discretization, dynamic programming (DP), gate sizing, pruning, sparse cell library.**

I. Introduction

INCREASING design complexities along with time-to-market pressures and shortening product cycles have mandated a shift in very large scale integration (VLSI) design from custom crafting to cell-library-based design methodologies. This shift raises an increasing need of salient gate-sizing techniques which are powerful in performing delay–area tradeoff optimizations. A handful set of gate-sizing techniques exist; however, most of them handle the continuous gate-sizing problem which is based on the assumption that gate sizes can be any values within a certain range (see, e.g., [1]–[4]).

On the other hand, a large number of realistic cell libraries are "sparse." For example, when the cell sizes are geometrically spaced instead of uniformly spaced, a significant sparseness is introduced. Refer to [5] for some realistic sparse libraries. Geometrically spaced gate sizes are desired because uniformly spaced gate sizes would result in a large number of gate sizes, and managing this large volume of data is difficult [5]. Furthermore, it is proven in [5] that, under certain conditions, the set of optimal gate sizes must satisfy the geometric progression. When gate implementations are restricted to discrete sizes, as in reality, the problem becomes much more difficult, and very few approaches (see, e.g., [6] and [7]) are known.

In this paper, we propose a novel gate-sizing technique which handles discrete gate sizes. As many efficient solutions exist for the continuous gate-sizing problem, one might think of obtaining a discrete solution through rounding a continuous solution. This is very fast but often results in large timing violations for a sparse cell library. In contrast, the method proposed by Coudert [7], which is based on the multidimensional descent optimization, handles the discrete sizes. However, it has some trial-and-error flavor and thus has room for further improvement. A dynamic-programming-like (DP-like) approach can search solutions more systematically and therefore has the potential to generate high-quality solutions. However, it may suffer from a substantial amount of computation overhead, which imposes a great challenge to our problem.

The key idea of the new algorithm is to integrate the solution quality of DP-like approaches with the short runtime of obtaining a solution to the continuous version of the problem. That is, we narrow down the searching space of DP under the guidance from a best continuous solution. Thus, instead of checking every implementation, our algorithm only investigates a number of discrete implementations around the best continuous solution. This enables us to find solutions with quality close to the best continuous case and, at the same time, obtain a huge speedup in computation. We also develop new techniques to prune inferior solutions and maintain/increase the diversity of intermediate solutions. Focusing on a small number of diversified and representative solutions during the candidate solution propagation can improve the efficiency of the solution search. To this end, an advanced scheme called locality sensitive hashing (LSH) technique [8] is explored to maintain solution diversity via selecting well-spaced solutions in high dimensions.

In summary, the main contributions of this paper are 1) a continuous-solution-guided DP-like approach for discrete gate sizing and 2) an LSH-based solution-pruning technique that allows the obtaining of high-quality solutions by maintaining solution diversity.

Our experimental results demonstrate that 1) the nearest rounding approach often leads to large timing violations for sparse cell libraries and 2) compared to the well-known Coudert's approach [7], the new algorithm saves up to 21% in area cost for the basic library while still satisfying the timing constraint. For a sparser library, the new algorithm provides benefits of up to 24% in area cost, thus showing that the sparser the library, the better are the benefits of the new algorithm.

Manuscript received September 21, 2007; revised November 11, 2008. Current version published May 20, 2009. This paper was recommended by Associate Editor I. Bahar.

S. Hu is with the Department of Electrical and Computer Engineering, Michigan Technological University, Houghton, MI 49931 USA (e-mail: shiyan@mtu.edu).

M. Ketkar is with Strategic CAD Labs, Intel Corporation, Hillsboro, OR 97124 USA (e-mail: mahesh.c.ketkar@intel.com).

J. Hu is with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: jianghu@ece.tamu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2009.2015735

The rest of this paper is organized as follows. Section II presents the problem formulation. Section III presents the high-level description of the approach. Section IV describes the details of the algorithm for discretizing the continuous solution. Section V presents the experimental results. A summary of this paper is given in Section VI.

II. Problem Formulation

Given a combinational circuit with n gate nodes, n_i primary input nodes and n_o primary output nodes, and a gate library L consisting of $|L|$ gate types where each gate type, characterized by the functionality and the number of gate inputs, may have various gate sizes, the *discrete gate-sizing* problem asks to compute a sizing solution with the *minimal total gate area cost* such that the maximum delay between any primary input node and any primary output node is bounded above by a delay constraint α. The problem can be formally defined as

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{n} a_i W_i \\ \text{s.t.} \quad & Delay \leq \alpha \\ & W_i \in L \end{aligned} \tag{1}$$

where W_i denotes the size of gate i and a_i denotes its weighting factor. Weighting factors can be set to unity for gate-size minimization and to the weighted summation of signal probabilities and activity factors for explicit power optimization.

III. Optimization Methodology

Before investigating the discrete gate-sizing problem, it is helpful to go over the closely related continuous gate-sizing problem. In this problem, gate sizes are allowed to be any real value between certain lower and upper bounds, and the resulting problem can be efficiently solved, for example, by using the Lagrangian relaxation technique [2]. Moreover, an optimal solution can be obtained if the underlying delay model is a posynomial function. For example, the solution is proven to be optimal when the Elmore delay model is used [2].

It might be expected that a good discrete solution can be obtained by rounding the gate sizes of the continuous solution to the nearest discrete gate sizes. However, this is not the case for the sparse cell library as the choices of gate implementations are very restrictive (see also [7] for this observation). As indicated from our experimental results, with a sparse cell library, the nearest rounding strategy can make the ISCAS'85 benchmark circuits have timing violations of hundreds of picoseconds in the 90-nm technology (refer to Tables I and II in Section V).

Nearest rounding may introduce a significant amount of timing violations for a sparse cell library. On the other hand, the DP approach can obtain the optimal solution for the discrete gate-sizing problem; however, it is computationally prohibitive as it needs to investigate every gate size at each gate node.

We propose a *continuous-solution-guided dynamic-programming-like algorithm* to integrate the optimality of the DP-like framework with the high efficiency. The searching space of the DP is significantly narrowed down under the guidance from a good continuous solution while solution quality is only slightly degraded in spite of the huge speedup. At each gate node, instead of every discrete gate size, only those close to the continuous solution will be investigated.

There are many continuous gate-sizing techniques which use various delay models such as the Elmore delay model in [2] and the convex delay model in [9]. For simplicity, we adopt the Elmore delay model in this paper. However, our method is independent of delay model, and any delay model is applicable. For example, the convex delay model in [9] can be employed to compute a better continuous-solution guider and a better overall result. The rest of this paper concentrates on our novel algorithm used to discretize the continuous solution.

IV. Discretization Algorithm

Before explaining the algorithm, we will present our circuit model and elaborate on some key terms necessary to describe the algorithm. In our algorithm, a circuit is represented as a directed acyclic graph where each node corresponds to either a logic gate or a primary input or output of the circuit and where each edge corresponds to a pin-to-pin connection between two gates.

In this paper, a *complete solution* refers to the determination of all gate sizes in the circuit. A *partial solution* is a solution where not all gate sizes have been determined. A partial solution becomes a complete solution when all gates are processed. For convenience, when there is no confusion, we also call a partial solution a solution. We denote a discrete solution by γ and the underlying continuous solution by γ^c. The outline of the algorithm is shown in Fig. 1.

The algorithm begins with the primary input nodes, proceeds through a breadth-first traversal of the circuit graph, and processes each gate in turn. As our discretization approach is guided by the continuous solution, at each gate node, only discrete gate sizes close to the continuous solution are investigated. To this end, each gate is measured by its criticality, and more gate sizes will be investigated for more critical gates. The details of this step are presented in Section IV-A.

During breadth-first traversal, once a gate node is *processed*, it is included in a partial solution. Many partial solutions may be formed. It is necessary to perform a solution-pruning technique to reduce the size of the solution set and thus save the runtime. There are two types of pruning. The first type of pruning is called node pruning which is performed during the time a node is processed. The second type of pruning is called solution-set pruning which is performed after a node is processed and when the number of partial solutions is greater than a threshold. The threshold is experimentally determined to achieve balance between solution quality and runtime for each circuit. This solution-set-pruning technique uses the LSH technique and cutline pruning technique to effectively and efficiently reduce the size of the solution set. These two types of pruning are explained in Sections IV-B and C, respectively.

A. Explore Gate Sizes Close to the Continuous Solution

During the breath-first traversal, the gate sizes investigated at each gate are determined by the criticality of the gate. In

```
0. Given: Continuous solution γ^c
1. Algorithm : Discretize_GateSize
2.     For each gate k during the breadth-first traversal of the circuit graph
3.         try several discrete gate sizes around its continuous gate size based on k's criticality
4.             generate partial solutions and perform node pruning
5.         If the size of the solution set is greater than a threshold
6.             perform solution set pruning
7.     Select the best solution at the primary output
```

Fig. 1. Pseudocode for discretization algorithm.

our algorithm, the criticality of a gate is measured by its slack, namely, a gate is more critical if its slack is smaller. Since one does not know the slack of each node before completing the sizing procedure, the slack is estimated by using our guider, i.e., the continuous solution. We first identify the worst slack, denoted by WS, of the circuit in the continuous solution, and then, the relative slack of each gate g, denoted by $RelSlack(g)$, is measured by $Slack(g)/WS$. Thus, the gate is more critical with a smaller relative slack. Note that, when the worst slack is very close to zero, the division may result in a large number, and the aforementioned computation may not be robust. Thus, a small positive constant ϵ is added to the slack of all nodes for the robustness of computation, i.e., $RelSlack(g) = (Slack(g) + \epsilon)/(WS + \epsilon)$.

Our approach is a criticality-based approach, meaning that we spend more discretization efforts on more critical gates. In this paper, we implement this idea by classifying gates into three groups and spend different amounts of discretization efforts for each group. Note that our approach is not restricted to three groups, and other classification methods can be used. In this paper, we set up two thresholds t_1 and t_2 where $1 < t_1 < t_2$ such that, when $RelSlack(g) < t_1$, T_1 sizes around g's continuous gate size are explored, when $t_1 \leq RelSlack(g) < t_2$, T_2 sizes around g's continuous gate size are explored, and when $RelSlack(g) \geq t_2$, only the gate size closest to g's continuous gate size is explored. Thus, gate criticality, which is inversely proportional to $RelSlack(g)$, decides how many sizes close to the continuous solution are explored. By this, the search space is greatly reduced in our discrete gate-sizing algorithm compared to the standard DP-like approach.

B. Solution Pruning

During solution propagation, even though the search space is judiciously restricted by the use of the continuous guider, at each gate, the algorithm still needs to check and keep several solutions close to the continuous solution. Propagating all solutions is impractical and unnecessary. Many of them are *inferior* to others and should be pruned to save computation cost. There are two types of pruning in our case. The first type of pruning is called node pruning which is performed during the time a node is processed. The second type of pruning is called solution-set pruning which is performed after a node is processed and when the size of the solution set is greater than a threshold.

We now introduce the first type of pruning. Each solution is characterized by a cumulative delay and gate area. That is, each solution γ is characterized by a (D, W) pair, where $D(\gamma)$ refers to the maximum delay from any primary input node to any processed node in γ and $W(\gamma)$ refers to the cumulative gate area for all processed gates in γ. After processing a node, (D, W) will be accordingly updated for each new solution. Note that computing D may require the knowledge of downstream gates (i.e., input capacitance of downstream gates) which are not yet processed. The continuous gate sizes at those downstream gates are then used as an approximation. This makes sense as our whole approach is guided by the continuous solution.

The first type of pruning, called *node pruning*, is formally defined as follows. For two solutions γ_1 and γ_2 generated at *the same* input gate g_i, γ_2 is *node inferior* to γ_1 if and only if $D(\gamma_1) \leq D(\gamma_2)$ and $W(\gamma_1) \leq W(\gamma_2)$. The node-inferior solution will be pruned, and thus, only the solution with either a smaller maximum delay or total area survives.

After carrying out the computation for a while, the size of the solution set Γ becomes very large. To maintain efficiency, Γ has to be shrunken before successive computations. For this purpose, when the size $|\Gamma|$ of the solution set Γ is greater than a threshold (after processing a gate node), the second type of pruning, called *solution-set pruning*, is performed. In this pruning technique, we also consider to maintain the diversity of the solutions.

To diversify solutions, we group similar solutions and then select the representative one from each group for further propagation. The solutions which have not been selected are pruned to save runtime. Suppose that we have computed clusters among solutions and inside each cluster; solutions are similar to each other. In each cluster, the representative solution is the one closest to our continuous guider. The representative solutions will be selected for further propagation while other solutions are pruned.

The proximity of a partial discrete solution to the continuous solution is defined by both delay and area information. We first define the concept of a cutline, as shown in Fig. 2, as a subset of edges in the circuit such that it partitions the circuit graph into two disjoint subgraphs. A cutline is formed when the size of the current solution set is greater than a threshold. Note that all solutions have the same set of processed gates. A cutline is formed such that, for each edge cut by a cutline, the upstream gate node has been processed but the downstream (i.e., its fan-out) gate nodes have not been processed. Denote all those fan-in

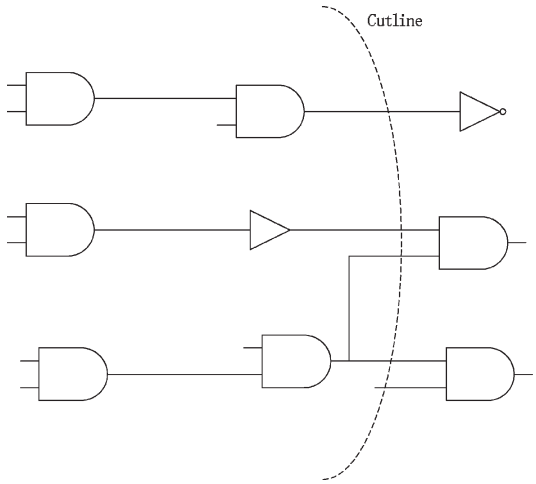

Fig. 2. Cutline.

gate nodes by $\{g_i\} = \{g_1, g_2, \ldots\}$. Each solution γ is assigned a proximity value $f(\gamma)$ which is defined as

$$f(\gamma) = \sum_{g_i} |D(\gamma(g_i)) - D(\gamma^c(g_i))| \cdot \sum_{g_i} |W(\gamma(g_i)) - W(\gamma^c(g_i))| \quad (2)$$

where $|D(\gamma(g_i)) - D(\gamma^c(g_i))|$ measures the delay difference between γ and the continuous solution and $|W(\gamma(g_i)) - W(\gamma^c(g_i))|$ measures the area difference. Clearly, the f value of a solution measures its proximity to the continuous solution. Suppose that there are many nodes along the cutline. It is difficult and inefficient to compare all solutions by comparing each node along the cutline. The proximity defined as aforementioned assigns a single value for each solution which makes the comparison much easier and efficient. Clearly, a solution with a small proximity value is preferred. In another words, γ_2 is inferior to γ_1 if and only if $f(\gamma_1) \leq f(\gamma_2)$. The representative solution from each cluster is the one which has the smallest proximity value.

We are now to describe the solution grouping technique. Grouping solutions can be realized through mapping each solution to a high-dimensional vector followed by clustering geometrically close vectors. The mapping is computed as follows.

Assuming that each gate node in the circuit is indexed and each gate implementation in the gate library is also indexed, each gate index corresponds to a distinct dimension, and the coordinate along that dimension is equal to the index of the assigned gate implementation. In this way, a solution is mapped to a d-dimensional vector if d nodes have been processed. We are now to compute clusters among the resulting vectors. For a large circuit, one has to cluster vectors in a very high dimension. Although the high-dimensional clustering problem has been intensively studied in decades, it remains as one of the hardest problems in the database research. In this paper, based on a highly effective and efficient nearest neighbor query technique called LSH [8], a new technique for solution clustering and representative-solution selection is introduced.

C. Solution Clustering by LSH

In cluster computation and nearest neighbor query, the "curse of dimensionality," which roughly says that the computational complexity of the aforementioned two operations increases exponentially with dimension, has remained as a notorious problem in database research for a long time. To effectively attack this problem, a new approach, called LSH, is introduced in [8]. With provable bound on approximations, LSH can efficiently approximate similarity search by the hashing technique. The basic idea is to hash the vectors such that the geometrically close (respectively, far apart) vectors are hashed to the same (respectively, different) bins with large probability. LSH enables us to answer a nearest neighbor query in $O(dm^{1/(1+\epsilon)})$ time over an m-point d-dimensional database for any $\epsilon > 0$. In addition, an approximate nearest neighbor query can be answered in sublinear time *excluding the preprocessing time* [8], where given a point set P, the problem asks to return a point $p \in P$ such that the distance of p to the query point q is at most $1 + \epsilon$ times the distance from the nearest point in P to q. It is shown in [8] that LSH is much more efficient compared to many other methods. LSH can be easily used for clustering since each bin in a hash table can be treated as a cluster (note that geometrically close vectors are hashed to the same bin with a large probability).

Suppose that d nodes have been processed in each solution in the solution set Γ. Each solution γ is first mapped to a d-dimensional point p, where a dimension corresponds to a node. Suppose that there are m solutions in the solution set. After mapping, there are m d-dimensional points, which form the point set P. We then embed these points into the Hamming space H with dimension $d' = Md$, where M is the number of available sizes for any cell type in the library. This embedding allows us to perform random sampling on the embedded bit string. For embedding, taking each point $p \in P$, we transform it into a binary vector $v(p) = \langle \Upsilon_M(x_1), \ldots, \Upsilon_M(x_d) \rangle$ where $\Upsilon_M(x)$ denotes the unary representation of x (i.e., x ones followed by $M - x$ zeroes).

It is helpful to illustrate the aforementioned concept by a simple example. Suppose that, in a solution set Γ, there are three solutions γ_1, γ_2, and γ_3, and three nodes have been processed. Suppose that, in solution γ_1, the processed nodes are assigned with sizes of 1, 2, and 5, respectively, which are the indices of the sizes for each processed gate. The solution is then mapped to a point $p_1 = (1, 2, 5)$. Further suppose that $M = 5$; then, $|v(p_1)|$ has a length of $d' = 15$, and $v(p_1) = 10000 : 11000 : 11111$ since, e.g., $2 = 11000$ in unary representation (i.e., two 1s followed by three 0s). Similarly, suppose that, in γ_2, the assigned gate sizes are 1, 1, and 5, and in γ_3, the assigned gate sizes are 2, 1, and 3. γ_2 will be mapped to a point $p_2 = (1, 1, 5)$, and $v(p_2) = 10000 : 10000 : 11111$. γ_3 will be mapped to a point $p_3 = (2, 1, 3)$, and $v(p_3) = 11000 : 10000 : 11100$. Refer to Fig. 3(a) for the result.

For convenience, $v(p)$ for a point p is treated as a bit string. It is clear that $v(p)$ can be very long for the large circuit. Given a set of solutions $\{\gamma_1, \gamma_2, \ldots\}$ and their corresponding set of $\{v(p_1), v(p_2), \ldots\}$, many bits may be the same. In a solution set, each solution is characterized only by its difference

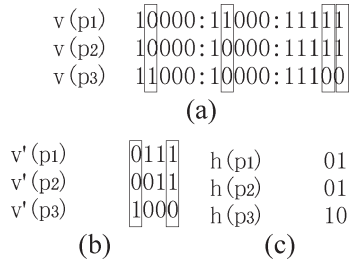

Fig. 3. Illustration of concepts in LSH. (a) Original three bit strings from $v(p)$. (b) Bit strings $v'(p)$ after removing redundancy. (c) Bit strings $h(p)$ after LSH.

from other solutions. Thus, we can remove the redundancy before clustering them for high efficiency. For this purpose, $\{v(p_1), v(p_2), \ldots\}$ are reduced to $\{v'(p_1), v'(p_2), \ldots\}$ by keeping only the difference among solutions. Denote the length of $v'(p)$ by $\mathtt{d}$. In Fig. 3(a), one sees that, in $\{v(p_1), v(p_2), v(p_3)\}$, only four bits are different, and all the other bits are the same. Thus, we can shrink the length of the bit strings to four. Refer to Fig. 3(b) for the result.

LSH then performs a further dimension-reduction mapping to the bit strings through random sampling for clustering. For dimension-reduction mapping, we randomly choose k elements from $\{1, 2, \ldots, \mathtt{d}\}$, where each element has equal probability to be chosen, and form an index subset $I = \{i_1, i_2, \ldots, i_k\}$. We then map each point p into $h(p) = \langle v'(p)[i_1], v'(p)[i_2], \ldots, v'(p)[i_k]\rangle$. For Fig. 3(b), if we choose $k = 2$ elements from $\{1, 2, 3, 4\}$ as 1, 4, then $I = \{1, 4\}$ and $h(p_1) = 01$ since $v'(p_1)[1] = 0$ and $v'(p_1)[4] = 1$. Similarly, $h(p_2) = 01$ and $h(p_3) = 10$. Refer to Fig. 3(c) for the result.

In literature, function $h(\cdot)$ is called the *locality-sensitive hash function* [8]. It is shown in [8] that the probability for two embedded points to have the same hash value (i.e., hashed into the same bucket) is "proportional" to their similarity. Based on this intuitive fact, we are able to build a hash table to support an efficient similarity search and clustering among a set of points. In Fig. 3(c), p_1 and p_2 are hashed to the same bucket, and p_3 is hashed to a different bucket. Each bucket corresponds to a cluster; thus, we have two clusters in Fig. 3(c). Given n gate nodes in the circuit, mapping a solution to Hamming space takes $O(n)$ time, and dimension reduction takes $O(n)$ time. Thus, for m solutions, the total time complexity for clustering is $O(mn)$.

V. Experimental Results

The new discrete gate-sizing algorithm, denoted by NEW, is implemented in C++ and tested on an X86 computer. Our test cases are ISCAS'85 benchmark circuits with a dense 90-nm gate library where each type of gate has ten sizes. They are $1\times, 2\times, 3\times, 4\times, 6\times, 8\times, 12\times, 16\times, 24\times,$ and $32\times$ of the minimum size with respect to each cell type. We use the Elmore delay model in continuous optimization phase, and the library gates are characterized by using the Elmore metric as well. In the experiments, for the most critical gates, we use $T_1 = 4$ discrete gate sizes around the continuous gate size (if available). For the less critical gates, we use $T_2 = 2$ (if available). For the least critical gates, we use the closest gate size. Different values can be assigned based on the usage model.

To judge the efficacy of our discretization algorithm, we compare its results with the solutions obtained by simply rounding each size in the continuous solution to the nearest discrete size. In addition, Coudert's approach [7], which is a well-known discrete gate-sizing technique, is also implemented for comparison. In this paper, we use the total capacitance of gates as our area cost function. Comparison results are summarized in Table I. We make the following observations.

1) Nearest rounding always introduces large timing violations. Although the total gate area of nearest rounding is similar to that of a continuous solution, its timing is much worse compared to the continuous solution.
2) NEW archives much better timing compared to nearest rounding, which makes sense as NEW benefits much from the knowledge of criticality and other global information which is lacking in the case of nearest rounding.
3) Compared to [7], 1%–21% of area-cost reductions are obtained by NEW.
4) The runtime of NEW including computing the continuous solution is, on average, about 50% higher than that of [7]. This is already very good considering that a DP-style approach is performed. The efficiency comes from our continuous-solution-guided scheme and pruning techniques.

As our approach is proposed for cell-library-based designs, we also inspect how its effectiveness scales with the discreteness in the library. For this, we select six geometrically spaced sizes $(1\times, 2\times, 4\times, 8\times, 16\times,$ and $32\times)$ for each cell type to form a sparser cell library. Gate sizing using nearest rounding and NEW are performed with this new cell library, and the results are summarized in Table II. We make the following observations.

Nearest rounding often introduces larger timing violations for our geometrically spaced cell library compared to the original cell library. For a detailed comparison on the solutions by nearest rounding and NEW, the histogram for the gate sizes for the whole circuit and the critical path of two circuits are shown in Figs. 4 and 5. Nearest rounding blindly tries to remain close to the continuous solution, while that may not be the best, given the discreteness in the library. NEW, since it covers much wider optimization space and is aware of gate criticality, can judiciously upsize gates. For example, it can be seen from Fig. 4 that NEW moves a number of gates from size $1\times$ to $2\times$ for the whole design, but on the critical path, it moves significantly more percentage of gates to upper sizes, thus showing the benefit over blind rounding to the nearest size. From Table II, one can also see that NEW and [7] are able to obtain the solutions satisfying the timing constraints, and the solutions are generally worse than the solutions computed with the original cell library. Compared to [7], 4%–24% area-cost reductions are obtained by NEW. Clearly, our algorithm becomes increasingly useful with a sparser cell library.

As a byproduct, the proposed algorithm enables us to compute a local delay–area tradeoff curve around the continuous

TABLE I
COMPARISONS USING A LIBRARY WITH TEN SIZES PER GATE TYPE. TIMING CONSTRAINTS AND SLACK ARE IN *ps*. CPU IN SECONDS REFERS TO RUNTIME. AREA REFERS TO AREA COST

Circuit	Timing Constraint	Continuous Solution			Nearest Rounding		Approach in [7]			NEW			Area Reduction
		Slack	Area	CPU	Slack	Area	Slack	Area	CPU	Slack	Area	CPU	NEW v.s. [7]
C432	800	0	4.1	4.1	-42	4.1	12	5.1	10.7	6	4.4	13.7	14%
C499	900	0	7.6	5.9	-94	7.4	12	8.7	28.9	11	8.3	47.8	5%
C880	700	0	5.9	4.2	-115	5.8	4	8.2	21.2	5	6.5	41.7	21%
C1355	1200	0	7.7	7.2	-93	7.5	15	10.1	32.1	26	9.2	37.2	9%
C1908	1200	0	19.7	12.6	-94	19.7	5	24.6	67.5	20	21.7	81.0	12%
C2670	1200	0	20.9	27.8	-102	20.9	37	28.7	121.5	26	23.9	202.1	17%
C3540	1700	0	33.3	21.5	-172	33.2	32	44.4	179.2	41	38.8	284.5	13%
C5315	1500	0	40.1	37.8	-139	40.0	5	55.3	301.8	1	45.7	634.7	17%
C6288	2500	0	33.1	37.6	-191	33.0	19	41.3	403.2	40	38.2	736.0	8%
C7552	2100	0	58.1	67.7	-130	58.1	31	59.7	497.5	1	59.1	534.7	1%

TABLE II
COMPARISONS USING A SPARSER LIBRARY WITH SIX SIZES PER GATE TYPE. TIMING CONSTRAINTS AND SLACK ARE IN *ps*. CPU IN SECONDS REFERS TO RUNTIME. AREA REFERS TO AREA COST

Circuit	Timing Constraint	Continuous Solution			Nearest Rounding		Approach in [7]			NEW			Area Reduction
		Slack	Area	CPU	Slack	Area	Slack	Area	CPU	Slack	Area	CPU	NEW v.s. [7]
C432	800	0	4.1	4.1	-101	4.0	43	5.4	10.4	4	4.7	12.4	13%
C499	900	0	7.6	5.9	-94	7.4	20	9.1	27.0	12	8.4	40.7	8%
C880	700	0	5.9	4.2	-115	5.8	5	9.0	18.7	1	6.8	33.5	24%
C1355	1200	0	7.7	7.2	-93	7.5	39	11.3	31.5	6	9.1	35.7	19%
C1908	1200	0	19.7	12.6	-147	19.5	1	25.7	60.4	9	22.4	87.1	13%
C2670	1200	0	20.9	27.8	-117	20.8	57	29.0	115.7	83	24.2	188.0	17%
C3540	1700	0	33.3	21.5	-247	33.1	40	44.9	165.8	11	39.4	247.7	12%
C5315	1500	0	40.1	37.8	-155	39.8	7	55.8	329.8	15	46.4	647.8	17%
C6288	2500	0	33.1	37.6	-312	32.7	72	44.2	417.7	45	40.7	775.1	8%
C7552	2100	0	58.1	67.7	-153	58.1	25	61.5	471.9	3	59.0	537.1	4%

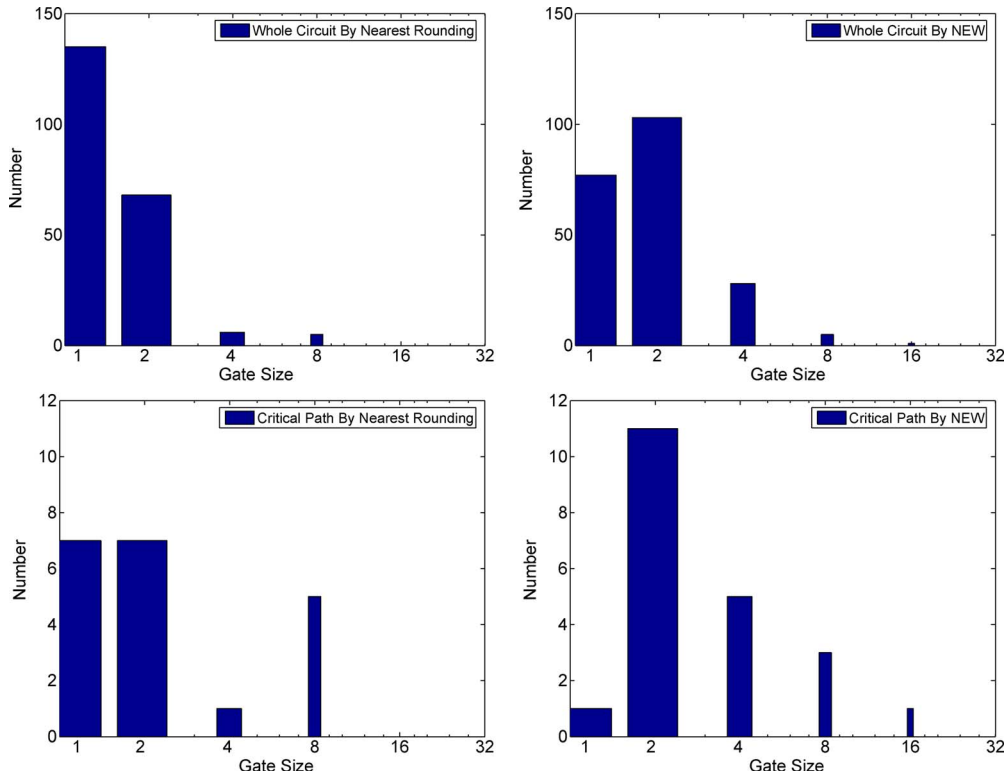

Fig. 4. Gate-size histogram for the whole circuit and the critical path of C432 benchmark circuit.

solution. Refer to Fig. 6 for the two resulting curves. For each plot in Fig. 6, the delay–area tradeoff curve computed by NEW is shown, and the result by Coudert's approach [7] is also shown for reference. Note that a single point is shown for Coudert's approach since it only returns a single solution, namely, its smallest area solution satisfying the timing constraint. The

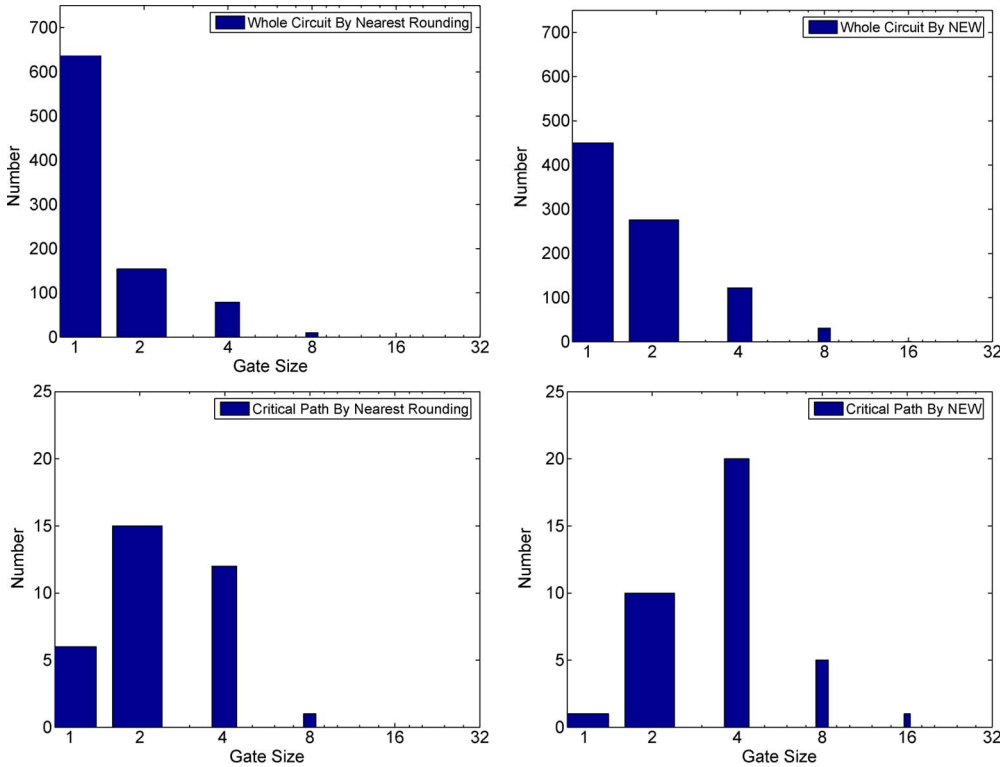

Fig. 5. Gate-size histogram for the whole circuit and the critical path of C1908 benchmark circuit.

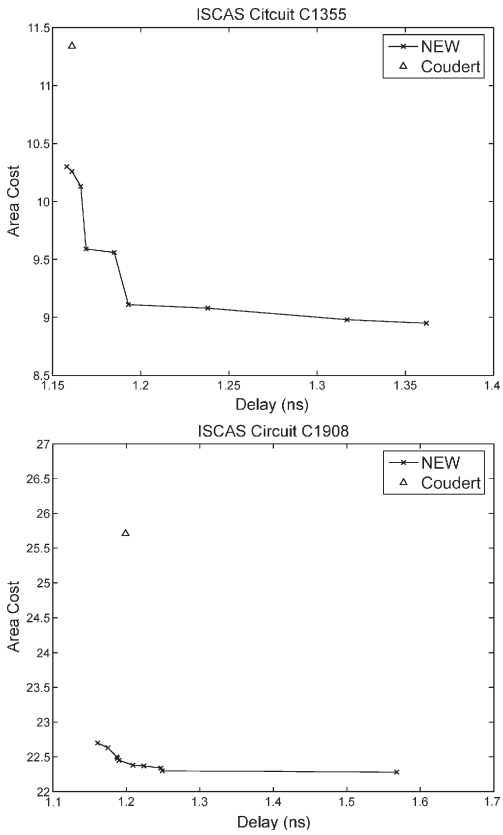

Fig. 6. Delay–cost tradeoff curves for optimizing two ISCAS benchmark circuits. The results of NEW and Coudert's approach [7] are shown.

obtained local tradeoff curve can help users get a better timing constraint for the circuit. With a new timing constraint, users can use NEW to generate better solutions.

VI. Conclusion

This paper proposed a new gate-sizing approach which handles the discrete gate library for sparse cell libraries. The new algorithm is based on the idea of continuous-solution-guided DP and uses pruning techniques for speedup. Our experimental results demonstrated that up to 21% area-cost reduction can be obtained compared to the well-known Coudert's approach [7]. Furthermore, by our approach, a set of tradeoffs instead of a single solution are obtained, which can help users get a better timing constraint for the circuit and provide significant freedom to meet design specifications.

Acknowledgment

The authors would like to thank the anonymous reviewers for their helpful comments.

References

[1] J. Fishburn and A. Dunlop, "Tilos: A polynomial programming approach to transistor sizing," in *Proc. Int. Conf. Comput.-Aided Des.*, 1985, pp. 326–328.

[2] C.-P. Chen, C. Chu, and D. Wong, "Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 7, pp. 1014–1025, Jul. 1999.

[3] S. Boyd, S. Kim, D. Patil, and M. Horowitz, "Digital circuit optimization via geometric programming," *Oper. Res.*, vol. 53, no. 6, pp. 899–932, Nov./Dec. 2005.

[4] N. Hanchate and N. Ranganathan, "Simultaneous interconnect delay and crosstalk noise optimization through gate sizing using game theory," *IEEE Trans. Comput.*, vol. 55, no. 8, pp. 1011–1023, Aug. 2006.

[5] F. Beeftink, P. Kudva, D. Kung, and L. Stok, "Gate-size selection for standard cell libraries," in *Proc. Int. Conf. Comput.-Aided Des.*, 1998, pp. 545–550.

[6] W. Chuang, S. Sapatnekar, and I. Hajj, "Delay and area optimization for discrete gate sizes under double-sided timing constraints," in *Proc. IEEE Custom Integr. Circuits Conf.*, 1993, pp. 9.4.1–9.4.4.

[7] O. Coudert, "Gate sizing for constrained delay/power/area optimization," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 5, no. 4, pp. 465–472, Dec. 1997.
[8] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc. ACM Int. Conf. Very Large Data Bases*, 1999, pp. 518–529.
[9] K. Kasamasetty, M. Ketkar, and S. S. Sapatnekar, "A new class of convex functions for delay modeling and its application to the transistor sizing problem," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 7, pp. 779–788, Jul. 2000.

Shiyan Hu (M'08) received the Ph.D. degree in computer engineering from Texas A&M University, College Station, in 2008.

He was with the IBM Austin Research Laboratory, Austin, TX, from May 2007 to November 2007. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Michigan Technological University, Houghton. His research interests are primarily on very large scale integration computer-aided design including interconnect optimization, gate sizing, placement, variation-aware optimization, and design for manufacturability.

Dr. Hu has served as a Technical Program Committee member for conferences including the IEEE International Symposium on Quality Electronic Design, IEEE SOC Conference, IEEE International Conference on Microelectronics, IEEE Asia Pacific Conference on Circuits and Systems, and IEEE International Midwest Symposium on Circuits and Systems.

Mahesh Ketkar received the Ph.D. degree in electrical engineering from the University of Minnesota, Minneapolis, in 2002.

Since then, he has been a Researcher with Strategic CAD Labs, Intel Corporation, Hillsboro, OR, and has worked in multiple areas including circuit optimization, power delivery, and design-silicon interaction. His prior work experience includes a year spent with Siemens, India, and summer internships with Intel in 1999 and 2000.

Dr. Ketkar was the recipient of the IBM Fellowship in 2000 and 2001 and a Best Paper nomination at Design Automation Conference 2007.

Jiang Hu (M'01–SM'07) received the B.S. degree in optical engineering from Zhejiang University, Hangzhou, China, in 1990 and the M.S. degree in physics and the Ph.D. degree in electrical engineering from the University of Minnesota, Minneapolis, in 1997 and 2001, respectively.

He was with IBM Microelectronics from January 2001 to June 2002. Currently, he is an Associate Professor with the Department of Electrical and Computer Engineering, Texas A&M University, College Station. His research interest is on computer-aided design for very large scale integration circuits, particularly on interconnect optimization, clock network synthesis, variation tolerance technology, and design for manufacturability.

Dr. Hu was the recipient of the Best Paper Award at the Association for Computing Machinery/IEEE Design Automation Conference (DAC) in 2001 and an IBM Invention Achievement Award in 2003. He has served as a Technical Program Committee member for DAC, International Conference on Computer-Aided Design, International Symposium on Physical Design, International Symposium on Quality Electronic Design, International Conference on Computer Design, Design Automation and Test in Europe, and International Symposium on Circuits and Systems. Currently, he is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN.