J Comb Optim
DOI 10.1007/s10878-009-9267-0

Approximation scheme for restricted discrete gate sizing targeting delay minimization

Chen Liao · Shiyan Hu

Abstract Discrete gate sizing is a critical optimization in VLSI circuit design. Given a set of available gate sizes, discrete gate sizing problem asks to assign a size to each gate such that the delay of a combinational circuit is minimized while the cost constraint is satisfied. It is one of the most studied problems in VLSI computer-aided design. Despite this, all of the existing techniques are heuristics with no performance guarantee. This limits the understanding of the discrete gate sizing problem in theory.

This paper designs the first fully polynomial time approximation scheme (FPTAS) for the delay driven discrete gate sizing problem. The proposed approximation scheme involves a level based dynamic programming algorithm which handles the specific structures of a discrete gate sizing problem and adopts an efficient oracle query procedure. It can approximate the optimal gate sizing solution within a factor of $(1+\epsilon)$ in $O(n^{1+c}m^{3c}/\epsilon^c)$ time for $0<\epsilon<1$ and in $O(n^{1+c}m^{3c})$ time for $\epsilon \geq 1$, where n is the number of gates, m is the maximum number of gate sizes for any gate, and c is the maximum number of gates per level. The FPTAS needs the assumption that c is a constant and thus it is an approximation algorithm for the restricted discrete gate sizing problem.

Keywords Combinatorial optimization · VLSI design · Delay optimization · Discrete gate sizing · Fully polynomial time approximation scheme

1 Introduction

Discrete gate sizing is a critical optimization due to its effectiveness in obtaining various delay and power trade-off in circuits, which has been widely deployed in

C. Liao (✉) · S. Hu
Department of Electrical and Computer Engineering, Michigan Technological University, Houghton, MI 49931, USA
e-mail: cliao@mtu.edu

S. Hu
e-mail: shiyan@mtu.edu

Published online: 03 September 2009

chip layout design. In the problem, a combinational circuit is given which can be represented by a directed acyclic graph (DAG). Each node corresponds to a gate and each gate is associated with a set of available gate sizes where different gate sizes lead to different delays and powers (or costs). The discrete gate sizing problem asks to assign appropriate size to each gate such that the longest path delay is minimized while the sum of costs over all gates is no greater than a cost bound.

There exist a large multitude of previous works with different objectives for gate sizing. The standard gate sizing techniques for exploring delay and power tradeoff are proposed in Chuang et al. (1995), Chen et al. (1999), Tennakoon and Sechen (2002), Sundararajan et al. (2002), Wang et al. (2007), Fishburn and Dunlop (1985), Sapatnekar et al. (1993), Berkelaar and Jess (1990), Murugavel and Ranganathan (2004). As the extensions to them, gate sizing techniques considering process variations are designed in Mani and Orshansky (2004), Singh et al. (2005), Mahalingam et al. (2006), gate sizing techniques for cross-talk noise reduction are proposed in Sinha and Zhou (2004), Hanchate and Ranganathan (2006), a reliability driven gate sizing technique is proposed in Zhou and Mohanram (2006), and a security aware gate sizing technique is proposed in Bhattacharya and Ranganathan (2008).

Unfortunately, most of the existing techniques such as a Lagrangian relaxation based technique in Chen et al. (1999) and a posynomial programming based approach in Fishburn and Dunlop (1985) can only handle the continuous gate sizing problem which assumes that gate sizes can be any values within certain range (Hu et al. 2007). This assumption is not realistic since it is difficult and not practical to manufacture gates with continuous sizes. In practice, only a small set of gate sizes are available, which imposes a pressing need for the techniques to handle discrete gate sizes. To obtain a discrete gate sizing solution, rounding the sizes of a continuous solution to discrete sizes is fast and intuitive. However, it will result in the significant degradation of circuit delay compared to the obtained continuous gate sizing solution (Hu et al. 2007; Beeftink et al. 1998). This motivates some recent works to design combinatorial algorithms which directly handle discrete gate size, such as a continuous solution guided dynamic programming technique in Hu et al. (2007), a network-flow based approach in Ren and Dutt (2008), a parallelization and randomization based technique in Wu et al. (2008), and a multi-dimensional gradient descent based algorithm in Coudert (1997). These algorithms are effective, however, they are all heuristics without any theoretical guarantee on the quality of their discrete gate sizing solutions. This limits the understanding of the discrete gate sizing problem in theory.

This paper aims to deepen the understanding of the discrete gate sizing problem from the theoretical point of view. Recall that given a minimization problem, an algorithm is said to approximate the optimal solution within a factor α if this algorithm can always produce a solution whose objective function value is at most α times the value of the optimal solution. The problem admits a fully polynomial time approximation scheme (FPTAS) if there is an algorithm which approximates the optimal solution within a factor of $(1+\epsilon)$ for any $\epsilon > 0$ and runs in time polynomial in both of the input size and $1/\epsilon$.

In this paper, the first fully polynomial time approximation scheme is designed for the delay driven discrete gate sizing problem. The algorithm works under the scaling and rounding framework of Hassin (1992), Ergun et al. (2002), Vazirani (2001).

The proposed approximation scheme involves a level based dynamic programming algorithm which handles the specific structures in gate sizing problem and adopts an efficient oracle query procedure. It can approximate the optimal gate sizing solution within a factor of $(1+\epsilon)$ in $O(n^{1+c}m^{3c}/\epsilon^c)$ time for $0<\epsilon<1$ and in $O(n^{1+c}m^{3c})$ time for $\epsilon \geq 1$, where n is the number of gates, m is the maximum number of gate sizes for any gate, and the constant c is the maximum number of gates per level. The technique needs the assumption that c is a constant, which is why our algorithm is said to approximate the restricted discrete gate sizing. Due to the fact that the discrete gate sizing problem is strongly NP-hard (Ning 1994), making such an assumption is reasonable. Note that in characterizing the delay of each gate, an appropriate delay model is needed. In this work, the widely used Elmore delay model (Elmore 1948) is adopted. Other delay models, such as those used in Kasamsetty et al. (2000), Ketkar et al. (2000), Roy et al. (2007), Xie and Davoodi (2008) where the delay is additive and the gate delay is only related to the gate and its fan-outs, can also be handled in our discrete gate sizing technique.

The rest of the paper is organized as follows: Sect. 2 presents the notations and the problem formulation of the discrete gate sizing problem. Section 3 proposes our approximation scheme to solve the discrete gate sizing problem and analyzes the time complexity and approximation ratio. A summary of work is given in Sect. 4.

2 Preliminaries

2.1 Notations and definitions

A combinational circuit can be represented by a directed acyclic graph (DAG). Given a DAG $G=(V,E)$ with $n=|V|$ nodes, each node corresponds to a gate. Following the convention of the gate sizing literature, let *the primary input gates*, denoted by *PI*, specify the nodes with zero in-degree, and *the primary output gates*, denoted by *PO*, specify the nodes with zero out-degree. Refer to Fig. 1. Think of pushing a flow into the primary input gates of G and it goes from g_1 to g_3. g_1 is called upstream to g_3, and g_3 is called downstream to g_1. Denote by $fin(g)$ the set of fan-in (input) gates of gate g and by $fout(g)$ the set of fan-out (output) gates of gate g. For example, $fin(g_3)=\{g_1\}$ and $fout(g_1)=\{g_3,g_4,g_7\}$.

For each gate g, denote by $s(g)$ the assigned gate size. It needs to be in a set of available gate sizes for g, denoted by $S(g)$. Denote by m the maximum number of gate sizes for any gate, i.e., $|S(g)| \leq m, \forall g$. In our gate sizing problem, the gate delay of a gate g with a size of $s(g)$, denoted by $d(g)$, is computed according to Elmore delay model (Elmore 1948) which is widely used in VLSI design. Note that other delay models, such as those used in Kasamsetty et al. (2000), Ketkar et al. (2000), Roy et al. (2007), Xie and Davoodi (2008) where the delay is additive and the gate delay is only related to the gate and its fan-outs, can also be handled in our technique. Denote by $C(g,s(g))$ and $R(g,s(g))$ the capacitance and resistance of gate g when it is assigned with gate size $s(g)$, respectively. $d(g)$ is the product of its resistance and the total capacitance of all its fan-out gates, i.e., $d(g)=R(g,s(g))\cdot\sum_{\forall fout(g)} C(fout(g),s(fout(g))$. The arrival time at a gate is defined as the

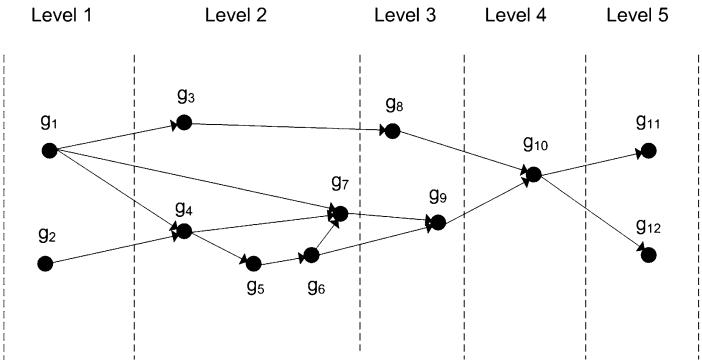

Fig. 1 Illustration of a levelized circuit

maximum current arrival time of all paths at the input of this gate plus its gate delay, i.e., the arrival time at its output. Denote by $t(g)$ the arrival time at gate g. One has, $t(g) = \max\{t(fin(g))\} + d(g)$. The arrival times at the input of all primary input gates are 0, and the arrival times at all the primary input gates are the gate delay of themselves. As in most papers about gate sizing for combinational circuit (Coudert 1997; Hanchate and Ranganathan 2006), we assume that there is a non-sizable gate (flipflop), linking to all the primary output gates. The delay of a circuit refers to the maximum arrival time at any primary output gate of the circuit, which is also called the longest path delay. The discrete gate sizing problem considered in this paper is to minimize the longest path delay through appropriately assigning the gate size at each gate since different gate sizes lead to different delays. It is worth mentioning that another discrete gate sizing formulation is also very important which asks to minimize the total cost of the circuit subject to the delay constraint. It is interesting to design a dedicated algorithm for this problem as well.

Refer to the example in Fig. 2 for illustrating the arrival time and gate delay. Suppose that in the example, g_1 is assigned with gate size 2, g_2 is assigned with gate size 5, g_3 is assigned with gate size 1, g_4 is assigned with gate size 1, and g_5 is assigned with gate size 7. Recall that $t(g)$ denotes the arrival time at the output of a gate g and $d(g)$ denotes the gate delay of g. $C(g, s(g))$ and $R(g, s(g))$ are the capacitance and resistance of gate g when it is assigned with gate size $s(g)$, respectively. In Fig. 2, g_1 is the primary input gate. The arrival time at g_1, i.e., $t(g_1)$, is equal to the delay of g_1, i.e., $t(g_1) = d(g_1) = R(g_1, 2)(C(g_2, 5) + C(g_3, 1))$. For the gates which have the same fan-out gate, the arrival time at these gates is equal to the maximum arrival time to guarantee the worst-case circuit performance. For example, $t(g_2) = t(g_3) = \max\{t(g_1) + d(g_2), t(g_1) + d(g_3)\} = \max\{t(g_1) + R(g_2, 5)C(g_4, 1), t(g_1) + R(g_3, 1)C(g_4, 1)\}$. $t(g_4)$ is equal to $t(g_2)$ plus the delay of g_4, i.e., $t(g_4) = t(g_2) + d(g_4) = t(g_3) + d(g_4) = t(g_2) + R(g_4, 1)C(g_5, 7)$.

On the other hand, the circuit cost will be also impacted by gate size assignment. Each gate g at size $s(g)$ is associated with a cost $w(s(g))$. The cost of a set of gates is defined as the sum of costs of all gates. The cost constraint W says that the sum of costs of all gates in G needs to be less than or equal to W.

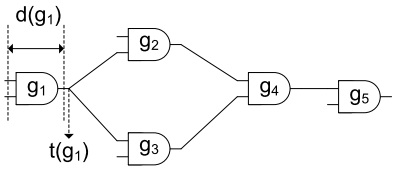

Fig. 2 Illustration of gate delay d and arrival time t

The circuit will be partitioned by levels. Initially, we would let *level*-1 *gates* specify all the primary input gates and let *level*-2 *gates* specify the gates immediately downstream to level 1 gates. However, since a gate g can be reached through different paths from primary input gates, it can belong to different levels according to the above definition. Thus, for each gate, its level is defined as the minimum possible level reachable from any primary input gate. In addition, for a gate g, if there exists a path from g to any level-i gate g', the level of g will be no greater than i, the level of g'. This means that whenever a gate g' is included into a level i, we also include all the gates along any path to g' to level i provided that they are not yet included in any of the previous levels (level $1, 2, \ldots, i-1$). Denote by c the maximum number of gates in any level. Our FPTAS needs the assumption that there are constant gates in each level, i.e., $c = O(1)$. Thus, it is said to approximate the restricted discrete gate sizing.

It is helpful to look at an example to illustrate the concept of level. Refer to Fig. 1 where the levels for gates are shown. After g_4 is classified as a level-2 gate, g_5 would be classified as a level-3 gate. However, g_7 is also classified as a level-2 gate since it connects to g_1 which is a level-1 gate. Backtracking the graph from g_7, g_6 and g_5 can be reached and they are not yet classified. Thus, they are also included in level 2. The backtracking stops when a gate with classified level is encountered. The total backtracking time (summing up backtracking time over all nodes) takes $O(|E|) = O(n^2)$ time which will be bounded by the FPTAS time. In summary, in Fig. 1, $L_1 = \{g_1, g_2\}$, $L_2 = \{g_3, g_4, g_5, g_6, g_7\}$, $L_3 = \{g_8, g_9\}$, $L_4 = \{g_{10}\}$, and $L_5 = \{g_{11}, g_{12}\}$. In addition, $c = 5$ which is due to the level-2 gates.

The following notations will be used in our FPTAS design. Let T^* denote the delay of the optimal gate sizing solution. Let $\overline{T}$ and $\underline{T}$ denote certain upper bound and lower bound on T^*, respectively. Various techniques can be used to obtain $\overline{T}$ and $\underline{T}$. For example, $\overline{T}$ can be obtained from always using the largest capacitance and the largest resistance at any gate (even if they belong to different sizes), ignoring the cost constraint. Similarly, $\underline{T}$ can be obtained from always using the smallest capacitance and the smallest resistance at any gate, ignoring the cost constraint. $\overline{T}$ and $\underline{T}$ can be certainly computed in linear time through evaluating the circuit delay in each case. Note that the lower bound and the upper bound are not tight, however, they make sense due to that the optimal solution, if any, must be in this range and can be searched out. On the other hand, as our FPTAS is independent of $\overline{T}$ and $\underline{T}$, it is not important to design techniques which can generate tighter upper or lower bounds.

Some notations frequently used in this paper are summarized as follows.

- n: the total number of gates.
- m: the maximum number of gate sizes for any gate.
- c: the maximum number of gates in any level.

- T^*: delay of the optimal discrete gate sizing solution.
- $\underline{T}$: a lower bound on T^*.
- $\overline{T}$: an upper bound on T^*.
- $t(g)$: the arrival time at the output of a gate g.
- $s(g)$: the assigned gate size of gate g.
- $d(g)$: the gate delay of g.
- L_i: the set of gates in level i.

2.2 Problem formulation

Discrete Gate Sizing Problem Given a DAG $G = (V, E)$, and a set $S(g)$ of available gate sizes for gate g, to compute a gate size assignment at each gate in G from the available gate sizes such that the arrival time at any primary output gate (longest path delay) is minimized while the cost constraint W is satisfied.

Instead of enumerating all paths to obtain the longest path delay, the discrete gate sizing problem is usually formulated to a mathematical programming problem as follows (Hu et al. 2007).

$$\begin{aligned}
\min\ & T \\
\text{s.t.}\ & \textstyle\sum_{\forall g_i} w(s(g_i)) \le W, \\
& t(g_i) \le T, \quad \forall g_i \in PO, \\
& t(g_i) + d(g_j) \le t(g_j), \quad \forall g_i \in fin(g_j), \\
& d(g_i) = R(g_i, s(g_i)) \textstyle\sum_{\forall g_j \in fout(g_i)} C(g_j, s(g_j)), \\
& d(g_i) \le t(g_i), \quad \forall g_i \in PI, \\
& s(g_i) \in S(g_i), \quad \forall g_i.
\end{aligned} \tag{1}$$

It is clear that the delay of a gate depends on the gate size assignment on its immediate downstream (fan-out) gates and the arrival time at a gate depends on the gate assignments of all the gates along any path from primary input gates to the fan-out gates of this gate.

3 The discrete gate sizing algorithm

3.1 Overview of the algorithm

Our FPTAS is motivated from Hu et al. (2009a, 2009b) and it works under the framework of the scaling and rounding based FPTAS design (Hassin 1992; Ergun et al. 2002; Vazirani 2001). In contrast to discrete gate sizing problem, (Hu et al. 2009a, 2009b) consider different problems, namely, the minimum cost delay driven buffering problem and layer assignment problem. Although the new technique shares some common features and flow with Hu et al. (2009a, 2009b), in appearance, there are underlying difference between their technique and our technique. In particular, Hu et al. (2009a, 2009b), can only handle tree topology while our discrete gate sizing technique needs to handle DAG which is a larger class of graphs. Due to this, a level based dynamic programming technique is proposed in this work.

Under the framework of Hassin (1992), Ergun et al. (2002), Vazirani (2001)), the algorithm first makes a guess on T^*. Denote the guessed value by T. Subsequently, check whether the guessed value T is a good guess, namely, whether it is sufficiently close to (at most ϵ away from) the optimal cost T^*. If T is a good guess, the corresponding discrete gate sizing solution will be returned as an approximate solution which is at most ϵ away from the optimal solution. Otherwise, the other guess is made. This process is repeated until a good guess is obtained.

There are two major algorithmic design issues with the above flow. First, since the optimal solution is not known, how can we tell whether a solution is sufficiently close to the optimal solution? Second, how can we effectively make the new guess to reduce the total number guesses if the current guess is not good? Certainly, one should utilize the information from the previous guesses in generating a possibly good new guess.

For the first issue, a procedure called *oracle* is used to check whether a solution is good. That is, the oracle can approximately decide whether $T^* \geq T$ for any positive number T efficiently. Once we have the oracle in hand, the second issue can be handled by efficient search using oracle. For example, one could perform a binary search between the upper and lower bounds of T^* using the oracle. However, this kind of technique cannot be used in designing the FPTAS since the number of iterations for binary search depends on the initial bounds. If the range between them is unbounded, the FPTAS will run in unbounded time. Thus, an efficient bound independent oracle based search technique proposed in Ergun et al. (2002) is used in this paper.

3.2 Oracle construction

3.2.1 Level based dynamic programming

We begin with describing how to construct the oracle. The key part of the oracle is a level based dynamic programming algorithm which can efficiently tell either $(1+\epsilon)T \geq T^*$ or $T < T^*$ for any $T > 0$. The efficiency is achieved by effectively pruning redundant solutions to make the number of solutions polynomially bounded during dynamic programming. By the proposed level based dynamic programming, the following lemma can be reached.

Lemma 1 *Given any $T, \epsilon > 0$, the level based dynamic programming algorithm can compute a solution with the delay at most $(1+\epsilon)T$, or report that there is no solution which can have delay no greater than T, in $O(nm^{3c} \cdot (n/\epsilon)^c))$ time, where n is the number of gates, m is the maximum number of gate sizes for any gate and the constant c is the maximum number of gates per level.*

Proof Let V_i denote the set of all the gates up to level i in G, and V_{i+1} denote the set of all the gates up to level $i+1$ in G. A gate sizing solution $\gamma(V_i)$ refers to a gate size assignment on gates in V_{i+1} (but not V_i). We call such a solution a level-i solution. Given a level-i solution $\gamma(V_i)$, to model the impact to the downstream gates, it is characterized by

$$(V_i, t(g_1), s(g_1), t(g_2), s(g_2), \ldots, t(g_k), s(g_k), s(g_{k+1}), s(g_{k+2}), \ldots, s(g_l), w(V_{i+1})), \tag{2}$$

where $L_i = \{g_1, g_2, \ldots, g_k\}$, and $L_{i+1} = \{g_{k+1}, g_{k+2}, \ldots, g_l\}$. This means that for any gate $g \in L_i$, the solution is characterized by the arrival time $t(g)$ at g when it is assigned with the size of $s(g)$ and its fan-out gates $fout(g)$ are assigned with the sizes of $s(fout(g))$. Thus, in a solution after processing a level, *the gate size assignments for all the gates in its next level are also determined*. Further, the solution characterization includes the cumulative cost so far, which is also called the cost of the solution, denoted by $w(V_{i+1})$. It is computed as the sum of costs for all the gates in V_{i+1}, i.e., $w(V_{i+1}) = \sum w(s(g)), \forall g \in V_{i+1}$. It can be easily seen that a solution is uniquely characterized as in (2). Further, if there are multiple solutions having the same characterization on $V_i, t(g_1), s(g_1), t(g_2), s(g_2), \ldots, t(g_k), s(g_k), s(g_{k+1}), \ldots, s(g_l)$, only one of them (with smallest cumulative cost $w(V_{i+1})$) needs to be maintained in the dynamic programming and all others are called *redundant* which can be pruned for acceleration. Note that these solutions are gate sizing solutions on V_i, and the cumulative cost is $w(V_{i+1})$, which is the total cost of the set of all gates up to level $i + 1$.

The level based dynamic programming begins with the primary input gates, i.e., L_1. For each gate g in L_1, for any size, the arrival time at the input of g is always 0 since it is a primary input gate, and the arrival time at a primary input gate is equal to its gate delay. For any solution of V_1, the gate size assignment for both L_1 and L_2 gates are determined. The cumulative cost of the solution is $w(V_2)$ (but not $w(V_1)$) which can be easily computed by summing up the costs of all gates in L_1 and L_2. Since there are at most m gate sizes for any gate and at most c gates per level, there are at most $O(m^{2c})$ possibilities on gate size assignment for the gates in the 2 levels. Thus, there are at most $O(m^{2c})$ solutions for V_1.

The level based dynamic programming then proceeds to the second level gates, i.e., L_2. For a solution $\gamma_1(V_1)$ which already includes the gate size assignment on V_2, we grow it to incorporate the gate size assignment on L_3. For this, all the possible sizes of L_3 are enumerated and a solution is generated per combination. Since there are at most m gate sizes for any gate and there are at most c gates per level, there will be at most m^c new solutions generated from a solution $\gamma_1(V_1)$. This is also the case for other solutions $\gamma_2(V_1), \gamma_3(V_1), \ldots$. Totally, there would be $O(m^{2c} \cdot m^c) = O(m^{3c})$ level-2 solutions.

If this process is continued, the number of solutions would be exponential in terms of the number of levels which is $O(n)$. On the other hand, when we proceed to compute the gate sizing solution on V_2, there are at most $O(m^c)$ possibilities on gate size assignment for the gates in L_3. If we are able to polynomially bound the number of possibilities on arrival times for all gates in L_2, the number of solutions can be polynomially bounded. This is due to the following fact. For two solutions with the same gate size assignment of L_2 and L_3, and the same arrival time at every gate in L_2, we only need to pick the solution with smaller cumulative cost (in case of tie, arbitrarily pick one) to propagate in dynamic programming since both solutions have the same impact to the downstream gates except the cumulative cost. This motivates us to use the classic rounding technique to bound the number of possibilities on the arrival time for all gates in L_2.

Given a solution, for each gate $g_i \in L_2$, round down its arrival time $t(g_i)$ to be the nearest multiple of $T\epsilon/n$, i.e., $t(g_i) = \lfloor t(g_i)/(T\epsilon/n) \rfloor \cdot T\epsilon/n$. Recall that T is

the guessed circuit delay and we are only interested in whether there is a solution with circuit delay (approximately) upper bounded by T. Thus, if there is any solution with the arrival time at any gate larger than T, the solution will be pruned. Due to this, at any gate, there can be at most n/ϵ distinct arrival times after rounding. Thus, after processing the level-2 gates and the rounding procedure, there can be at most $O(m^{2c} \cdot (n/\epsilon)^c)$ solutions since there are at most m^{2c} distinct gate sizes for L_2 and L_3, and $(n/\epsilon)^c$ distinct arrival times for L_2.

In this fashion, the level based dynamic programming proceeds level by level until the last level is handled. In general, one can see that there are at most $O(m^{2c})$ distinct gate sizes and $O((n/\epsilon)^c)$ distinct arrival times for any level. At each level, at most $O(m^{2c} \cdot (n/\epsilon)^c \cdot m^c)$ solutions will be generated where only $O(m^{2c} \cdot (n/\epsilon)^c)$ of them are not redundant according to rounding. Note that updating cumulative cost can be easily performed in $O(c)$ time for each solution. The pruning can be implemented using a multi-dimensional array where each entry links to a solution with different gate sizes and rounded arrival times for a level. Locating an entry takes $O(c)$ time. When a new level-i solution is generated, find and compare its cost to the level-i solution with the same gate sizes and arrival times at all level-i gates. If there is no such solution, link the solution to the entry. Otherwise, compare the cost to the cost of the existing solution. If its value is larger, prune the new solution. Otherwise, replace the solution with the new solution. Thus, the time complexity can be bounded as $O(c \cdot m^{2c} \cdot (n/\epsilon)^c \cdot m^c)$ per level and $O(nm^{3c} \cdot (n/\epsilon)^c)$ for all levels, assuming that $O(c) = O(1)$. Note that if T is too small, it is possible that no solution can be generated at any primary output gate (e.g., at a level, all the solutions have delay greater than T and are pruned).

Recall that the delay of a circuit refers to the maximum arrival time at the output of any primary output gate. Due to the rounding procedure, the obtained circuit delay is not accurate. First, it is a lower bound on the actual circuit delay since only down-rounding is performed. Thus, if the dynamic programming technique cannot find a solution which has delay no greater than T, there is no solution which can have the delay no greater than T with unrounded delay at each gate. Second, since the rounding error in delay for each gate is bounded by $T\epsilon/n$ which is our rounding factor, together with the fact that there are at most n gates along any path between a primary input gate and a primary output gate, the rounding error for the whole circuit is bounded by $T\epsilon$. This means that if a solution has the circuit delay of T, its actual circuit delay without rounding is bounded by $(1+\epsilon)T$. □

It is helpful to see an example to illustrate the pruning process. Refer to Fig. 3 where two solutions are presented. In Fig. 3, $[a, b]$ to the right of a gate g means that its rounded arrival time is a and its gate size is b. The two solutions are level-3 solutions and their gate size assignments at every level-3 gate are the same. Suppose that after rounding, both solutions have the same arrival time at every level-3 gate. Since the second solution has larger cumulative cost 14, it will be pruned.

3.2.2 Oracle construction

The oracle will decide whether either $(1+\epsilon)T \geq T^*$ or $T < T^*$ for any $T > 0$. For this, the oracle calls the level based dynamic programming with the input T and ϵ.

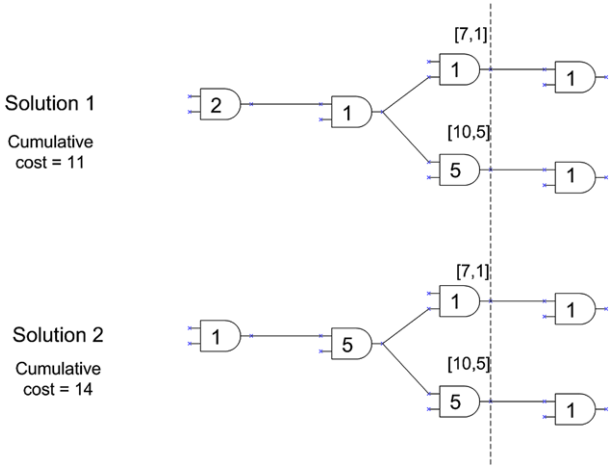

Fig. 3 An example of pruning where solution 2 is redundant with respect to solution 1

If there is a solution returned, it means that a solution with the circuit delay at most $(1+\epsilon)T$ has been obtained. Thus, $T^* \leq (1+\epsilon)T$. Otherwise, it means that there is no solution which can have the circuit delay T. In the first case, the oracle will return TRUE. In the second case, the oracle will return FALSE.

3.3 The FPTAS

FPTAS works as searching for the best delay T within the lower bound $\underline{T}$ and the upper bound $\overline{T}$ in an iterative manner. None of binary search and logarithmic scale binary search within these bounds can terminate in time independent of the bounds (Ergun et al. 2002; Hu et al. 2009a). If the range between them is unbounded, the time complexity of FPTAS will be unbounded. Thus, a technique proposed in Ergun et al. (2002), which is also used in Hu et al. (2009a, 2009b), is adopted to tackle this difficulty. By the proposed FPTAS, we can reach the following theorem.

Theorem 2 *The discrete gate sizing problem can be approximated within a factor of* $(1+\epsilon)$ *in* $O(n^{1+c}m^{3c}/\epsilon^c)$ *time for any* $0 < \epsilon < 1$ *and in* $O(n^{1+c}m^{3c})$ *time for any* $\epsilon \geq 1$, *where* n *is the number of gates*, m *is the maximum number of gate sizes for any gate*, *and the constant* c *is the maximum number of gates per level*.

Proof The idea of the searching technique is that instead of sticking to ϵ during optimization, adapting ϵ can significantly improve the runtime. This is due to the fact that the time complexity of the dynamic programming algorithm is inversely proportional to ϵ. If one sets ϵ as a geometrically decreasing sequence leading to ϵ, the total runtime will be bounded by the last run, independent of the number of iterations and the initial bounds.

Ergun et al. (2002) shows that this is possible. The oracle is called iteratively. In i-th iteration, set $\epsilon_i = \sqrt{\overline{T_i}/\underline{T_i}} - 1$ and $T_i = \sqrt{\overline{T_i}\underline{T_i}/(1+\epsilon_i)}$. Use T_i and ϵ_i as the input to the oracle. Depending on the binary result of the oracle, either $\overline{T_{i+1}}$ will be updated to $(1+\epsilon)T_i$ (when the oracle returns TRUE) or $\underline{T_{i+1}}$ will be updated to T_i

(when the oracle returns FALSE). This process is iterated until $\overline{T_i}/\underline{T_i} < 2$. During iterations, the ratio $\overline{T_i}/\underline{T_i}$ will be progressively reduced as $\overline{T_{i+1}}/\underline{T_{i+1}} = (\overline{T_i}/\underline{T_i})^{3/4}$ (Ergun et al. 2002). Note that the total runtime for dynamic programming is in the form of $O(\sum_i nm^{3c} \cdot (n/\epsilon_i)^c)) = O(nm^{3c}n^c \cdot \sum_i (1/\epsilon_i)^c))$. It is shown in Ergun et al. (2002) that $1/\epsilon_i < (2+\sqrt{2})\sqrt{\underline{T_i}/\overline{T_i}}$. As a result, the runtime bound becomes $O(nm^{3c}n^c \cdot \sum_i (\sqrt{\underline{T_i}/\overline{T_i}})^c)$. Since $c \geq 1$, $O(nm^{3c}n^c \cdot \sum_i (\sqrt{\underline{T_i}/\overline{T_i}})^c) = O(nm^{3c}n^c \cdot (\sum_i \sqrt{\underline{T_i}/\overline{T_i}})^c)$. Together with the fact that $\sum_i \sqrt{\underline{T_i}/\overline{T_i}} = O(1)$ when setting T_i and ϵ_i as above Ergun et al. (2002), the total time will be bounded by $O(nm^{3c}n^c \cdot O(1)^c)$.

At some point, $\overline{T_i}/\underline{T_i} < 2$. The above iterative procedure terminates. Similar to Hassin (1992), Hu et al. (2009a, 2009b), the level based dynamic programming is applied using the following setting. For each gate, its arrival time will be down rounded

Algorithm 1 Level based dynamic programming

```
DP(T_r, T, ε)
// there are l levels in the combinational circuit
i ← 1
while i ≤ l do
  for each level-(i − 1) solution do
    generate level-i solutions by enumerating all gate size
    assignments of level-i and level-(i + 1) gates
  end for
  for each level-i solution γ do
    if the cost of γ is > W then
      remove γ
    else
      for each level-i gate g do
        if the arrival time at g is > T then
          remove γ
        else
          round it down to the nearest multiple of T_r ε/n
        end if
      end for
    end if
    if cost of γ is larger than cost of the level-i solution
    with the same arrival time at every level-i gate then
      remove γ
    else
      replace the solution with γ
    end if
  end for
  i ← i + 1
end while
return the best delay solution or no feasible solution
```

Algorithm 2 The oracle

ORACLE(T, ϵ)
if DP(T, T, ϵ) returns a solution **then**
  return TRUE
else
  return FALSE
end if

Algorithm 3 The fully polynomial time approximation scheme for discrete gate sizing problem

FPTAS$(\overline{T}, \underline{T}, \epsilon)$
while $\overline{T}/\underline{T} > 2$ **do**
  $\epsilon' \leftarrow \sqrt{\overline{T}/\underline{T}} - 1$
  $T \leftarrow \sqrt{\overline{T}\underline{T}/(1+\epsilon')}$
  if ORACLE$(T, \epsilon') =$ TRUE **then**
    $\overline{T} \leftarrow T(1+\epsilon')$
  else
    $\underline{T} \leftarrow T$
  end if
end while
return DP$(\underline{T}, \overline{T}, \epsilon)$

to the nearest multiple of $\underline{T}\epsilon/n$ where ϵ is the target approximation ratio ϵ. If its arrival time is greater than $\overline{T_i}$, it will be pruned. Thus, at any gate, there will be at most $2n/\epsilon$ distinct arrival times. In any level, there are at most $(2n/\epsilon)^c$ possibilities of arrival times. After the whole circuit is processed, pick the smallest delay T_s with the cost no greater than the cost constraint W to be our solution. Since arrival time is only down-rounded, $T_s \leq T^*$. Rounding the arrival time of this gate sizing solution back, the delay is at most $(1+\epsilon)T_s \leq (1+\epsilon)T^*$. This single run of dynamic programming takes $O(nm^{3c} \cdot (2n/\epsilon)^c))$ time since the only difference from Lemma 1 is that we can have $O((2n/\epsilon)^c)$ possible arrival time combinations at any level instead of $O((n/\epsilon)^c)$ possibilities. Together with the runtime for the iterative oracle calls, the total runtime is bounded by $O(nm^{3c}n^c \cdot O(1)^c + nm^{3c} \cdot (2n/\epsilon)^c)$. This gives an FPTAS when c is a constant. □

The pseudo-code for the algorithm is shown in Algorithms 1, 2 and 3.

4 Conclusion

Discrete gate sizing is a critical optimization due to its effectiveness in obtaining various delay and power trade-off in a combinational circuit. However, all of the existing techniques are heuristics without any theoretical guarantee on the quality of

their gate sizing solutions. This paper designs the first FPTAS for the discrete gate sizing problem. Our algorithm can obtain an approximation within a factor of $(1+\epsilon)$ in $O(n^{1+c}m^{3c}/\epsilon^c)$ time for any $0 < \epsilon < 1$ and in $O(n^{1+c}m^{3c})$ time for any $\epsilon \geq 1$, where n is the number of gates, the constant c is the maximum number of gates per level, and m is the maximum number of gate sizes for any gate. The FPTAS needs the assumption that c is a constant. An interesting future work would be to design an FPTAS with a relaxed assumption.

Acknowledgements The authors are grateful to the reviewers for their very insightful comments which help improve the draft.

References

Beeftink F, Kudva P, Kung D, Stok L (1998) Gate size selection for standard cell libraries. In: Proceedings of IEEE/ACM international conference on computer-aided design, pp 545–550

Berkelaar M, Jess J (1990) Gate sizing in mos digital circuits with linear programming. In: Proceedings of the European conference on design automation, pp 217–221

Bhattacharya K, Ranganathan N (2008) A linear programming formulation for security-aware gate sizing. In: Proceedings of ACM great lakes symposium on VLSI, pp 273–278

Chen C-P, Chu C, Wong D (1999) Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation. IEEE Trans Comput-Aided Des Integr Circuits Syst 18(7):101–1025

Chuang W, Sapatnekar SS, Hajj IN (1995) Timing and area optimization for standard-cell VLSI circuit design. IEEE Trans Comput-Aided Des Integr Circuits Syst 14(3):308–320

Coudert O (1997) Gate sizing for constrained delay/power/area. IEEE Trans Very Large Scale Integr (VLSI) Syst 5(4):465–472

Elmore WC (1948) The transient analysis of damped linear networks with particular regard to wideband amplifiers. J Appl Phys 19(1):321–336

Ergun F, Sinha R, Zhang L (2002) An improved FPTAS for restricted shortest path. Inf Process Lett 83(5):287–291

Fishburn J, Dunlop A (1985) Tilos: a posynomial programming approach to transistor sizing. In: Proceedings of IEEE/ACM international conference on computer-aided design, pp 326–328

Hanchate N, Ranganathan N (2006) Simultaneous interconnect delay and crosstalk noise optimization through gate sizing using game theory. IEEE Trans Comput 55(8):1011–1023

Hassin R (1992) Approximation schemes for the restricted shortest path problem. Math Oper Res 17(1):36–42

Hu S, Ketkar M, Hu J (2007) Gate sizing for cell library-based designs. In: Proceedings of ACM/IEEE design automation conference, pp 847–852

Hu S, Li Z, Alpert C (2009a) A fully polynomial time approximation scheme for timing driven minimum cost buffer insertion. In: Proceedings of ACM/IEEE design automation conference (DAC)

Hu S, Li Z, Alpert C (2009b) A faster approximation scheme for timing driven minimum cost layer assignment. In: Proceedings of ACM international symposium on physical design (ISPD)

Mahalingam V, Ranganathan N, Harlow J III (2006) A novel approach for variation aware power minimization during gate sizing. In: Proceedings of international symposium on low power electronics and design, pp 174–179

Mani M, Orshansky M (2004) A new statistical optimization algorithm for gate sizing. In: Proceedings of international conference on computer design, pp 272–277

Murugavel A, Ranganathan N (2004) Gate sizing and buffer insertion using economic models for power optimization. In: Proceedings of IEEE international conference on VLSI design, pp 195–200

Ning W (1994) Strongly NP-hard discrete gate-sizing problems. IEEE Trans Comput-Aided Des Integr Circuits Syst 13(8):1045–1051

Ren H, Dutt S (2008) A network-flow based cell sizing algorithm. In: Proceedings of international workshop on logic synthesis, pp 7–14

Sapatnekar S, Rao V, Vaidya P (1993) An exact solution to the transistor sizing problem for CMOS circuits using convex optimization. IEEE Trans Comput-Aided Des Integr Circuits Syst 12(11):1621–1634

Singh J, Nookala V, Luo Z, Sapatnekar S (2005) Robust gate sizing by geometric programming. In: Proceedings of ACM/IEEE design automation conference, pp 315–320

Sinha D, Zhou H (2004) Gate sizing for crosstalk reduction under timing constraints by Lagrangian relaxation. In: Proceedings of IEEE/ACM international conference on computer-aided design, pp 14–19

Sundararajan V, Sapatnekar SS, Parhi KK (2002) Fast and exact transistor sizing based on iterative relaxation. IEEE Trans Comput-Aided Des Integr Circuits Systems 21(5):568–581

Tennakoon H, Sechen C (2002) Gate sizing using Lagrangian relaxation combined with a fast gradient-based pre-processing step. In: Proceedings of IEEE/ACM international conference on computer-aided design, pp 395–402

Vazirani VV (2001) Approximation algorithms. Springer, Berlin

Wang J, Das D, Zhou H (2007) Gate sizing by Lagrangian relaxation revisited. In: Proceedings of IEEE/ACM international conference on computer-aided design, pp 111–118

Wu T-H, Xie L, Davoodi A (2008) A parallel and randomized algorithm for large-scale discrete dual-Vt assignment and continuous gate sizing. In: Proceedings of international symposium on low power electronics and design, pp 45–50

Zhou Q, Mohanram K (2006) Gate sizing to radiation harden combinational logic. IEEE Trans Comput-Aided Desi Integr Circuits Syst 25(1):155–166

Kasamsetty K, Ketkar M, Sapatnekar SS (2000) A new class of convex functions for delay modeling and their application to the transistor sizing problem. IEEE Trans Comput-Aided Des Integr Circuits Syst 19(7):779–788

Ketkar M, Kasamsetty K, Sapatnekar SS (2000) Convex delay models for transistor sizing. In: Proceedings of the ACM/IEEE design automation conference, pp 655–660

Roy S, Chen W, Chen CC, Hu Y (2007) Numerically convex forms and their application in gate-sizing. IEEE Trans Comput-Aided Des Integr Circuits Syst (TCAD) 26(9):1637–1647

Xie L, Davoodi A (2008) Fast and accurate statistical static timing analysis with skewed process parameter variation. In: Proceedings of ACM/IEEE international symposium on quality electronic design (ISQED'08), pp 712–717