Obstacle-avoiding and Slew-constrained Buffered Clock Tree Synthesis for Skew Optimization [*]

Feifei Niu[⋆], Qiang Zhou[⋆], Hailong Yao[⋆], Yici Cai[⋆], Jianlei Yang[⋆], C. N. Sze[†]

[⋆]Dept. of Computer Science and Technology, Tsinghua University, Beijing, China
feifeiniu@gmail.com, {zhouqiang, hailongyao, caiyc}@mail.tsinghua.edu.cn, jerryyangs@gmail.com

[†]IBM Austin Research Laboratory, 11501 Burnet Road, Austin, Texas 78758
csze@us.ibm.com

ABSTRACT

Buffered clock tree synthesis (CTS) is increasingly critical as VLSI technology continually scales down. Many researches have been done on this topic due to its key role in CTS, but current approaches either lack the obstacle-avoiding functionality or lead to large clock latency and/or skew. This paper presents a new obstacle-avoiding CTS approach with separate clock tree construction and buffer insertion stages based on an integral view to explore the global optimization space. Aiming at skew optimization under constraints of slew and obstacles, our CTS approach features the clock tree construction stage with the obstacle-aware topology generation algorithm called OBB, balanced insertion of candidate buffer positions, and a fast heuristic buffer insertion algorithm. Experimental results show the effectiveness of our CTS approach with significantly improved skew and latency than [6] by 46% and 63% on average, and 15.3% reduction in skew than [5]. Our OBB heuristic obtains 36% improvement in skew than the classic balanced bipartition algorithm (BB) in [9].

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids—*Placement and routing*; J.6 [**Computer-aided Engineering**]: Computer-aided design

General Terms

Algorithms, Performance, Design

Keywords

Buffer insertion, Skew optimization, Obstacle avoidance, Slew, Clock tree synthesis

1. INTRODUCTION

Clock tree synthesis is an important element and problem in physical design which controls the pace of the whole circuit. As VLSI technology moves into the nanometer territory along with feature shrinking, buffer insertion becomes unavoidable in the CTS flow to reduce delay and keep the signal integrity. Most existing buffer insertion strategies either embed buffer insertion into the clock tree construction process, or conduct buffer insertion after the clock tree is constructed. The first strategy can generate better balanced structure with available accurate downstream delay and load information, but the simultaneous clock tree construction and buffer insertion is easily trapped in local optimal solution. The second strategy which separates the two correlative parts leads to degraded solution quality due to the lack of a global view. In this paper, to minimize skew under constraints of slew and obstacles, we present a new CTS approach with separate stages of obstacle-avoiding clock tree construction and buffer insertion based on comprehensive considerations of their relevance, which enables better solution space exploration to obtain improved solution quality.

As interconnects get thinner, clock networks cannot work without buffers. Many researches have worked on buffer insertion as an independent stage based on a given tree topology. Dynamic programming (DP) is first introduced to optimize the path delay of buffer insertion based on an existing wiring tree in [1]. Afterwards, in [2], more diverse targets are taken into account for buffer insertion, such as power minimization, signal slew, area, etc. Yet, the skew problem after buffer insertion was not followed closely by then. [3] symmetrically inserts buffers based on a symmetric tree structure. The algorithm obtains notable optimization in skew, but the resource will be a problem in large scale benchmarks. [4] presents a simultaneous buffer insertion/sizing and wire sizing algorithm *ClockTune* which is claimed to guarantee zero skew and minimize power and delay. The algorithm is based on a D-C plane, where the sampling of the solutions on the plane is not subtly planned. In [6], an integrated process of clock tree construction and buffer insertion based on the *Deferred-Merge Embedding* (DME) framework [9] is studied. The paper highlights the delay model and the look-up table providing accurate delay/slew during buffer insertion, while the positions of buffers which only depends on slew limits the skew that can be optimized. And it may cost a lot of routing resources on balancing the branches since the buffers are only inserted when the slew on the wire is about to violate the constraint, which means, the delay changes caused by buffer insertion is not carefully analyzed and utilized. In our paper, buffers' positions are relatively flexible on a wire as there are redundant candidate buffer positions to keep the signal integrity and so the actual buffer insertion positions could give consideration to the delay balance.

[*]This work was supported partly by National Science Foundation of China (NSFC 60876026 and 60833004)

In the VLSI physical design flow, CTS is performed after placement, so the already placed cells, macro blocks, and IPs become obstacles for following buffer insertion process. Researches [5][7][8] have worked on obstacle-avoiding routing or buffer insertion. [5][7] process obstacles after the initial zero skew clock tree (ZST) is constructed. For avoiding obstacles, every on-obstacle subnet is rerouted independently, which probably destroys the initial tree's balance on skew, capacitance, etc. [8] uses a *Walk-Segment Breadth First Search* (WSBFS) method to realize routing, the algorithm can achieve an exact routing plan based on a pre-prepared wiring structure. As we know, avoiding buffers on obstacles is not difficult. The real difficulty lies in reducing the negative effects on skew/delay caused by obstacle avoidance. In this paper, we make efforts to predict and correct the negative effects of obstacle avoidance for skew-minimized buffer insertion.

Classical H-tree and X-tree are excellent ZSTs. However, they require harsh conditions for sinks' positions and capacitance. Yet, Shih et al. regained this as symmetric structure in [3]. They sophisticatedly constructed a symmetric tree with almost the same structure on each path, and therefore balanced buffer insertion is easy to realize. The algorithm in [3] displays the advantages of symmetric structure on small scale benchmarks. However, the algorithm might not be suitable for large-scale benchmarks with obstacles since the algorithm would realize the tree with all paths equal to the longest path. And as [3] uses pseudo sinks to form the symmetric tree, the resource for wiring along with buffers inserted on such wires might be a ticking time bomb for the chip. A trade-off should be found between the algorithm's performance and resource it will take. Thus, we do not adopt symmetric structure in our CTS flow considering the potential risk.

In this paper, we focus on the construction of a buffered clock tree subject to the constraints of obstacles and the input slew of buffers and sinks. We compare our results with a recent work [6] which addresses the same problem as us except for obstacles and also with the intermediate results of one ISPD'09 contest winner [5], in which our problem is defined as a sub-problem of their whole flow, our CTS approach proves to be more effective. The contributions of this work are summarized as follows:

- An improved obstacle-aware algorithm called OBB is proposed to generate the clock tree topology based on BB [9] algorithm and the analysis of obstacles.
- An effective heuristic distribution algorithm of the candidate buffer positions (CBP) is adopted considering the negative impaction caused by obstacle avoidance on the paths' balance.
- An efficient sampling technique is adopted to speedup the DP algorithm on the basis of skew optimization during the process of slew constrained buffer insertion.

The rest of this paper is organized as follows. Section 2 describes the problem to be solved in this paper and gives the overall flow of our algorithm. Section 3 explains the details of the flow. Section 4 reports the experimental results. Finally section 5 concludes our work.

2. ALGORITHMIC FLOW

In this paper, we concentrate on buffered clock tree synthesis problem. Given a set of sinks SI, a set of obstacles OB, the task is to construct a buffered clock tree $T = (V, E)$

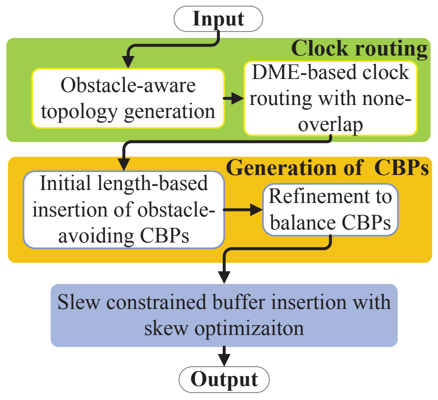
Input
Clock routing
Obstacle-aware topology generation
DME-based clock routing with none-overlap
Generation of CBPs
Initial length-based insertion of obstacle-avoiding CBPs
Refinement to balance CBPs
Slew constrained buffer insertion with skew optimizaiton
Output

Figure 1: The algorithmic flow

satisfying the following requirements: (1) no inserted buffers overlap with the obstacles; (2) the input slew of all the sinks and buffers are within a prespecified limit. We name a tree structure satisfying conditions (1) and (2) as a *legal* structure. The optimization goal is to optimize the skew of T.

For a clock tree T, some basic notations are defined here. $v \in V, SI \subseteq V, e \in E$. e_v is the edge from v to its parent node. B denotes the buffer library, and we assume *none-buffer* as a special buffer type in B, which denotes that no buffer is inserted.

Figure 1 shows our buffered clock tree synthesis algorithm which includes three steps: (1) obstacle-aware clock tree construction; (2) generation of obstacle-avoiding CBPs; (3) buffer insertion. For benchmarks without obstacles, the experiments show that the traditional zero-skew routing algorithms can lead to good skew after our buffer insertion strategy. But for benchmarks with obstacles, they seem not to work well since the wire snaking for obstacle avoidance will destroy the initial balance state. Our CTS approach features a global view on obstacles for the final skew. And it mainly reflects in the efforts we make to keep the potential balance character of the tree, especially before buffers are inserted.

The main job of step (1) is to route a ZST in a rectangle. Our algorithm is based on BB+DME [9], where BB is short for *Balanced Bipartition* and DME is short for *Deferred-Merge Embedding*. In this stage, a heuristic topology generation algorithm named OBB is proposed for obstacle avoidance. OBB predicts the obstacles' impaction on real routing, and attempts to raise the tree level impacted by obstacles. Paths' difference at upper levels is generally easier to deal with than that at lower levels. This is because the unbalanced delays induced by upper-level wires' differences involve less wires, and thus would be easier to repair. Note that in this stage the merge nodes have no overlaps with obstacles, but our routing method which directly connects two nodes may generate wires crossing the obstacles. We will reroute if CBPs appear on the obstacles based on the CBP insertion algorithm of the next stage. All lengths and distances are Manhattan distance in this paper.

Step (2) is the preparation stage for step (3), and it aims to generate a tree structure with CBPs for legal zero-skew buffered tree (ZSBT), i.e., after step (2), a legal ZSBT would potentially exist in the solution space for step (3). Our fundamental algorithm to insert CBPs is a length-based insertion with a fixed length constraint. But experiments have shown that if benchmarks contain obstacles, fixed-length-based insertion does not always work well. The reason is that the obstacle-avoidance preparation for CBP insertion would probably change the initial zero-skew routing strategy to make sure that every CBP on wires be off the obstacles. I.e., for obstacle avoidance, some path might be too long after the wire snaking, and therefore more buffers would be inserted on this path for slew constraint which implies more delay than that we have planned. To cover the potential legal ZSBT solution in the solution space of buffer insertion, we add a refinement stage after the initial CBP insertion to balance the tree structure and the CBPs' distribution. The algorithm is detailed in Section 3.2.

Step (3) focuses on skew-optimized and slew constrained buffer insertion. Slew is a hard restriction because it directly affects the function of a circuit. In [10], the slew constrained and minimum cost buffer insertion is studied. Based on a slew model, the illegal solution will be pruned from one node's solution set. In our algorithm, the slew model is referenced and proved to be useful. The general cost function of [10] is adaptable for objectives like power, area, and delay. For this situation, we have done an experiment in which the cost function is set to optimize skew. Yet, the algorithm seems not to be suitable for skew minimization. The slew buffering [10] assumes all paths are independent from other paths while the skew problem is actually a match problem, which means, different paths are not irrelevant to each other when they are setting their delays.

DP is popular for buffer insertion. Nevertheless, as the depth of the tree increases, DP would generate more and more solutions and causes a runtime issue. Shi and Li have done good research works on speeding up the DP algorithm [11]-[14]. The essential factor in their theory is the *dominate* concept that controls the solution space at one node. However, the dominate concept is not applicable for skew optimization because no complete solution at non-root nodes could dominate the other solutions according to the delay, or the temporary skew. In this paper, we propose an effective sampling technique to reduce the runtime and the clock skew. This sampling technique borrows the concept of A/D converter in communications, which converts the data from analog to digital through sampling and keeps the signal waveform. In our case, the delay values in the original solution set are not continuous as analog signal, but they are dense enough to form a delay curve. We will sample this dense solution set to be sparser and meanwhile keep the shape of the delay curve of the original solution set.

3. DETAILS OF THE ALGORITHMS

3.1 Obstacle-aware clock tree construction

BB+DME in [9] generates a ZST with minimum wire length. We adopt it as our basic algorithm for clock tree construction. Since obstacles are not considered in [9], we improve the basic BB+DME algorithm to consider obstacle avoidance.

Heuristic BB (OBB): BB is detailed in [9]. For simplicity, we borrow notations in [9] to explain our improved clock tree topology generation approach. For a sink set G, *Oct(G)* is its bounded *octagon*, and *REF*, the reference set to partition G, is a subset of ordered *Oct(G)*. For each *REF*, every sink in G has a weight defined as $minval$+$maxval$ $=\min\{d_r(p_i,p_{oi})|p_{oi}\in REF\}+\max\{d_r(p_i,p_{oi})|p_{oi}\in REF\}$, and $d_r(p_i,p_j)$ is the Manhattan distance between points p_i and p_j. All the sinks are inserted into a sorted list in non-decreasing order of their weights. Then all the sinks in G are partitioned into two subsets based on load balance. This method is sufficient for benchmarks without obstacles. But for benchmarks with obstacles as shown in Figure 2a, the classic BB algorithm may not find a good solution. We improve the BB algorithm to consider obstacle avoidance. The key is to change the sinks' order in the list.

Figure 2a shows an example. We assume all sinks are of the same capacitance. If REF consists of sinks s_1, s_2, s_3, s_4(see Figure 2a), BB will partition the sink set by the solid line (see Figure 2b) where sink ss will finally be connected by the dotted line. But because an obstacle exists, the actual connection will be the dotted line in Figure 2c. In this situation, the obstacle directly affects the lowest level of the tree which will propagate to all the other paths for delay balance. In our OBB, the weight of a sink is modified to $minOBval + minval + maxval$, where $minOBval$ is the estimation of the actual wire length connecting p_i and p_{min-oi} (see the dotted line shown in Figure 2c), where p_{min-oi} is defined as the point with $\min\{d_r(p_i,p_{oi})|p_{oi}\in REF\}$ without consideration of obstacles. The way to calculate $minOBval$ is simplified from our obstacle-avoiding algorithm in section 3.2 just for estimation. The reason we care about p_{min-oi} is that BB tends to partition p_i with p_{min-oi} in future since they are close to each other. As shown in Figure 2c, our OBB algorithm will partition the set with the solid line. Obstacle's impaction happens at top level and will just propagate among the top-level wires.

Extended Trr: In [9], Titled rectangular region (Trr) is a critical concept on computing the parent node's merging segment (ms) to realize zero skew in DME process. $Trr_i\cap Trr_j$ is always a *Manhattan arc* (*Marc*) in [9] because it takes the minimum Manhattan distance dd (see the dotted line in Figure 3a) between $core(Trr_i)$ and $core(Trr_j)$ to be $kval$ ($kval = radius_i + radius_j$). If we enlarge $kval$ to be greater than dd, the result will be a Trr region (proved in [9]), and here we call it *Extended* Trr. Since the same equations to compute $radius_i$ and $radius_j$ as [9] are adopted to balance the delays, the points in the intersection area (see Figure 3b) are all candidate merging nodes for zero skew. When there are obstacles, an ms with original $kval$ may overlap with obstacles. We can remove the overlapping part and take the remaining part as the ms. But if the initial ms entirely overlaps with obstacles, an alternative ms is needed. *Extended* Trr is useful in this case. In our algorithm, when entire overlap happens, $kval$ is iteratively enlarged by a certain step length until we find a Trr where available points exist off obstacles. The overhead of this algorithm is the increased wire length, but the advantage is that the feature of zero skew is maintained with obstacles.

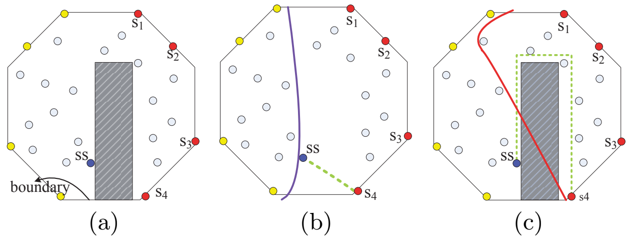

Figure 2: Obstacle-avoiding clock tree topology generation. (a) Example with a large obstacle. (b) The solid line shows the top partition based on BB without considering obstacles. (c) The solid line shows the partition based on OBB.

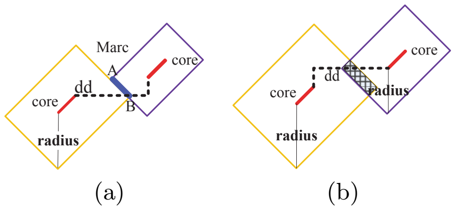

Figure 3: The intersection of $Trrs$. (a) $Trr_i \cap Trr_j$ with $radius_i + radius_j = dd$. The result is Marc, shown as the overlap segment AB. (b) $Trr_i \cap Trr_j$ with $radius_i + radius_j > dd$. The result is a Trr region.

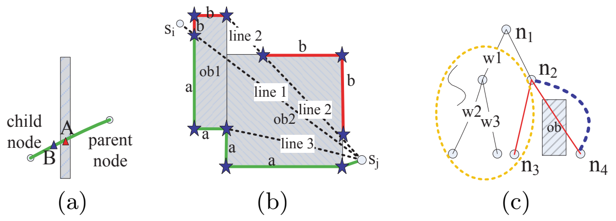

Figure 4: Reroute for obstacle avoidance. (a) If $l_{cross} \leq L_{Limit}$, ignore the obstacle. If CBP happens in the obstacle as point A, point B close to the child node will substitute it. (b) Complex overlap: the bold segments on the boundary with stars as the endpoints are detour candidates between s_i and s_j. (c) Wire snaking for obstacle avoidance impacts the original balance of the tree.

3.2 Generation of obstacle-avoiding CBPs

In this paper, a CBP refers to a candidate position for buffers. CBPs are constrained to slew and obstacles. The slew constraint requires the CBPs be distributed on both nodes and wires. The nodes have already been guaranteed to be off obstacles in the clock tree construction stage, while rerouting is required if CBPs on wires appear on obstacles.

A natural-avoiding algorithm is adopted for rerouting. In the algorithm, a fixed length L_{limit} determined by the slew constraint and the buffer's drive capability is set to be *ability-bound* for a wire, i.e., when a wire crosses an obstacle with length $l_{cross} \leq L_{Limit}$, the obstacle would be ignored (see Figure 4a). If $l_{cross} > L_{Limit}$, the wire should detour. Our algorithm processes obstacles crossed by wires one by one. For example, in Figure 4b, line 1 is the original wire connecting nodes s_i and s_j. The bold segments marked as a or b are two detour choices. Detour with smaller length is favorable. For the detour marked as a, after avoiding obstacle ob1, the wire becomes line 3 which overlaps with obstacle ob2, and for the detour marked as b, the wire becomes line 2. Next, this algorithm will process line 2 or line 3. This natural-avoiding algorithm could not guarantee the minimum length, but it can process various obstacles with complex shapes.

After the wire-on-obstacle-check and detour are conducted, CBPs can be inserted on the wire from child node to parent node by an interval L_{limit}. This fixed-length-based insertion strategy leads to a roughly uniform distribution of CBPs on the whole tree. However, experiments show that the severe unbalance of delays is often introduced by unbalanced wire snaking for obstacle avoidance. As shown in Figure 4c, obstacle ob conceptually exists among nodes n_2, n_3 and n_4. It is possible that wire e_{n_4} needs to snake as the dotted curve for obstacle avoidance, but wire e_{n_3} doesn't. So, unbalance happens between the path $n_4 - n_1$ and other paths in the dotted ellipse. To maintain the balance, we have to make compensations on the paths in the dotted ellipse. For a binary node which is defined as a node having two children, our measure is as follows: firstly, check if the node's two children are unbalanced on delay (including the delay of the wire connecting to the parent node). If they are unbalanced, calculate the difference *difflen* of *increased* lengths caused by obstacle avoidance, and the number of CBPs *incCBP* that should be compensated can be computed by *difflen*$/L_{limit}$; then uniformly add *incCBP* CBPs to the path with smaller delay. Note that this compensation process should be done in a bottom-up manner, so the unbalance at lower levels could be propagated to upper levels.

Table 1: Formulas to recursively calculate the six-tuple solution of single branch node and binary node.

Elements of six-tuple	Formula	
	single branch node	binary node
$c(v)$	$c(v_c) + c(e_{v_c}) + c(b_i)$	$c(v_l) + c(e_{v_l})$ $+ c(v_r) + c(e_{v_r}) + c(b_i)$
$d(v)$	$d(v_c) + d(e_{v_c}) + d(b_i)$	$[d(v_l) + d(e_{v_l})$ $+ d(v_r) + d(e_{v_r})]/2 + d(b_i)$
$maxD(v)$	$maxD(v_c)$ $+ d(e_{v_c}) + d(b_i)$	$max(maxD(v_l) + d(e_{v_l}),$ $maxD(v_r) + d(e_{v_r}))$
$minD(v)$	$minD(v_c)$ $+ d(e_{v_c}) + d(b_i)$	$min(minD(v_l) + d(e_{v_l}),$ $minD(v_r) + d(e_{v_r}))$
$sk(v)$	$sk(v_c)$	$maxD(v) - minD(v)$
$sl(v)$	refer to [10]	refer to [10]

3.3 Buffer insertion with skew optimization

Our buffer insertion is implemented through two phases: the bottom-up phase to generate candidate solutions for each node, and the top-down phase to deploy the buffers. During the bottom-up phase, node v's candidate solution set is formulated as $S_v = \{s_1, s_2, \cdots, s_n\}$, where s_i is a six-tuple $(c, d, maxD, minD, sk, sl)$. c denotes the total capacitance of sub-tree T_v; $d, maxD$, and $minD$ denote the average, the maximum and the minimum delay values from v to its leaves, respectively; sk denotes skew of sub-tree T_v, and sl for slew is calculated as [10] which denotes the accumulated slew degradation from the last downstream buffer. If v is a sink node, c is sink capacitance, $d = maxD = minD = 0, sk = 0, sl = 0$; if v is a single branch node with buffer b_i whose child is v_c, the six-tuple of v is computed as shown in Table 1; if v is a binary node, and v_l, v_r are its left and right child, the six-tuple of v is calculated by equations in Table 1. $sl(v)$'s calculation is detailed in [10], and the buffer's intrinsic delay and output slew in the formulation are achieved through a look-up table.

$$\begin{aligned} \min(sk(v)) &= \min(maxD(v) - minD(v)) \\ &= \min(maxD(v)) - \max(minD(v)) \quad (1) \end{aligned}$$

For a binary node v, we assume S_L and S_R are its left and right child's solution sets, $S_L = \{s_{l_1}, s_{l_2}, \ldots, s_{l_m}\}, S_R = \{s_{r_1}, s_{r_2}, \ldots, s_{r_n}\}$; then v's solution set is generated from combining s_{l_i} and $s_{r_j}, i \in [0, m], j \in [0, n]$. There are totally $m*n$ solutions. For node v, the minimum skew is calculated by Equation (1). It indicates that we have to minimize the maximum delay and maximize the minimum delay to get the optimal skew. So both large delays and small delays might contribute to skew minimization. Therefore, if the slew of a node for all solutions is within SL_{limit}, $|B| * |S|$

(S is the child's solution set) solutions will be introduced at each single-branch buffer position; at a binary node, $|B| * |S_L| * |S_R|$ solutions will be introduced. Overall, the number of solutions increases exponentially. So, we have to prune the solution set to solve the runtime issue, and meanwhile try to keep the optimal solution in the solution set.

Sampling technique: Equation (1) tells that node v's skew $sk(v)$ directly depends on its maximum delay and minimum delay, which we name as *boundary delay.* The method to optimize skew is to minimize the maximum delay and maximize the minimum delay of the tree. Targeting to minimize the whole tree's skew, only the difference of $maxD$ and $minD$ at the root node needs to be considered. The delay at the root node is the result of the delay's growth at the non-root nodes along the path. To include balanced delays at root node, we should concentrate on the *boundary potential* of delays at the internal nodes. Here *boundary potential* is defined as the potential ability of a delay value to become a *boundary delay.* Of course, the internal nodes' $maxDs$ and $minDs$ are important potential *boundary delays* at the root node. But the delays between *boundary delays* might also become the future $maxD$ or $minD$. In order not to prune the potential $maxD$ and $minD$ during the bottom-up process, we will sample the solution set to maintain the diversity of delays at the internal nodes. The sampling technique used on node v's solution set S_v includes three parts. The delay values ($minD$, $maxD$, d) are the main criteria for sampling, and we define α as the sampling interval.

Algorithm 1 *Sample d*

```
Input: $S'_v$ : $S_v$ after doing Sample minD and maxD
Output: $S''_v$: v's simplified solution set
 1: $minAD = \min(d)$, $maxAD = \max(d)$ from $S'_v$
 2: calculate the number of solution groups based on $\alpha$:
          $solGroup = (maxAD - minAD)/\alpha + 1$
 3: generate a 2-D vector $CSols[solGroup]$
 4: for $i = 0$ to $|S'_v| - 1$ do
 5:     push back $S'_v[i]$ to $CSols[floor((S'_v[i].d - minAD)/\alpha)]$
 6: end for
 7: $sta[i]$: the number of all original solutions in $CSols[i]$
 8: $dis[i]$: the allocated solution number for $CSols[i]$
 9: for $i = 0$ to $solGroup - 1$ do
10:     $dis[i] = floor(sta[i] * solGroup/|S'_v| + 0.5)$
11:     if $dis[i] = 0$ and $sta[i] > 0$ then
12:         $dis[i] = 1$
13:     end if
14: end for
15: for $i = 0$ to $i = solGroup - 1$ do
16:     randomly select $dis[i]$ solutions from $CSols[i]$
17: end for
```

(1) *Sample minD:* Each solution has a $minD$. For all solutions, there are a minimum $minD$ and a maximum $minD$. S_v is ordered in a $minD$-ascending list. This sampling $minD$ technique is to start from the first solution in ordered S_v, and select the next solution whose $minD$ is at least α larger than the previous solution. If more than one solution exist at a certain $minD$ value, the selection can be done randomly or according to the target. In our algorithm, random selection is adopted.

(2) *Sample maxD:* This part is similar to *Sample minD* part except that the sampling criterion changes from $minD$ to $maxD$, and the initial solution set does not contain the selected solutions in *Sample minD*.

(3) *Sample d:* As mentioned before, the delay d might have *boundary potential* to become a *boundary delay.* So, we continue to pick some solutions out of the remaining solution set S'_v. The specific algorithm is shown in Algorithm 1. Here, α is used to divide the solutions into different groups based on delay values d (see details in steps 1-6). During delay interval of α, there are $sta[i]$ solutions, but our sampling technique will only allow $dis[i]$ solutions in this interval. $dis[i]$ is calculated based on steps 9-13. And the solutions could be selected randomly or according to specific targets. In our algorithm, target-driven selection is adopted. For a binary node, solutions with smaller skew are favorable, and for a single-branch node, solutions with smaller power are chosen.

The value of α significantly affects the sampling, especially *Sample minD* and $maxD$. α is configured mainly depending on the path-delay range and the scale of the problem. The combination of the selected solutions from the three parts of *sampling technique* makes of the final solution set of node v.

4. EXPERIMENTAL RESULTS

We implemented our algorithm in C++ on a 2.33GHz Intel Xeon Linux workstation with 8GB memory. This paper focuses on obstacle-avoiding buffered clock tree construction without considering wire sizing and process variation. So, all configurations of the ISPD benchmarks remain unchanged except that the wire library consists only wire 0 with unit resistance 0.0001Ohm/nm and unit capacitance 0.0002fF/nm, and the parameters about process variation are disabled. The final results are simulated by NGSPICE. Note that ISPD'09 benchmarks (*09f11, 09f12, 09f21, 09f22, 09f31, 09f32, 09fnb1, 09fnb2*), and ISPD'10 benchmarks (*10f01-10f08*, where *10f0X* is for the original benchmark *0X.in*) are adopted to test our algorithm, in which, *09f31, 09f32, 09fnb1, 09fnb2, 10f01-10f05* all consist obstacles.

The parameters L_{limit} and α used in our algorithm is set to 500000nm and 1ps in all the ISPD benchmarks.

Table 2 shows the comparison of our algorithm and [6] on benchmarks without obstacles. *09fXX'* is a benchmark generated from *09fXX* with obstacles removed. Since no obstacles exist, our techniques to eliminate the negative impaction of obstacles do not work, and the effectiveness of the buffer

Table 2: Comparison of [6] and our work.

Benchmark	No. sinks	Worst slew(ps)		Skew(ps)		Max Latency(ns)	
		ours	[6]	ours	[6]	ours	[6]
09f11	121	80.5	99.2	22.2	45.2	0.92	2.26
09f12	117	75.4	83.6	18.7	45.8	0.90	1.92
09f21	117	83.5	99.2	32.2	51.1	0.97	2.16
09f22	91	80.8	100	16.5	42.4	0.77	1.62
09f31'	273	88.0	98.1	60.1	65.1	1.28	4.22
09f32'	190	80.9	85.2	26.9	52.3	1.26	3.38
09fnb1'	330	73.3	80	31.0	68.6	0.44	4.67
Avg.comparison		0.875	1.000	0.544	1.000	0.368	1.000

Table 3: Comparison of [5] and our work on skew (ps).

Benchmark	09f11	09f12	09f21	09f22	09f31	09f32	09fnb1	Ava.
[5]	46.78	66.24	76.31	33.65	129.2	98.27	21.13	-
ours	38.84	31.93	48.51	41.66	84.78	46.44	34.05	-
ours/[5]	0.830	0.482	0.636	1.238	0.656	0.473	1.612	0.847

Table 4: Comparison of OBB and BB

Benchmark	No. sinks	Skew (ps)		Max Latency (ns)		power ($\times 10^3$fF)		CPU (s)	
		OBB	**BB**	**OBB**	**BB**	**OBB**	**BB**	**OBB**	**BB**
09f31	273	76.8	187.1	1.680	1.377	181.0	183.0	2814	111
09f35	193	22.3	94.2	1.392	1.429	126.2	136.4	76	66
09fnb2	440	48.3	37.1	0.652	0.622	58.5	56.0	133	118
10f01	1107	71.4	87.0	0.869	1.050	158.9	152.2	1404	4416
ob04.in	1499	30.8	36.9	0.495	0.495	68.2	71.1	501	688
ob07.in	1780	33.7	43.7	0.502	0.502	73.9	70.2	527	550
Avg.comparison		0.729	1.000	1.002	1.000	1.012	1.000	4.939	1.000

Table 5: Comparison of our algorithm on benchmarks with (yes) and without (no) obstacles.

benchmark	No.	No.	Skew(ps)		Power($\times 10^3$fF)		Max Latency(ns)		CPU(s)		Skew/Max latency	
	sinks	obstacles	yes	no	yes	no	yes	no	yes	no	yes	no
09f31	273	88	76.8	60.1	181.0	180.0	1.680	1.344	2814	66	4.6%	4.5%
09f32	190	99	33.2	26.9	134.1	133.9	1.291	1.291	53	46	2.6%	2.1%
09fnb1	330	53	23.9	31.0	27.4	26.6	0.520	0.468	106	99	4.6%	6.6%
09fnb2	440	1346	48.3	38.9	58.5	54.7	0.652	0.625	133	122	7.4%	6.2%
10f01	1107	4	71.4	60.6	158.9	149.8	0.869	0.819	1404	318	8.2%	7.4%
10f02	2249	1	55.2	55.9	293.6	307.9	0.966	0.980	723	706	5.7%	5.7%
10f03	1200	2	38.4	37.9	55.8	55.4	0.471	0.470	381	353	8.2%	8.1%
10f04	1845	2	54.6	54.4	81.1	81.1	0.520	0.520	624	593	10.5%	10.5%
10f05	1016	1	32.7	28.9	38.2	38.0	0.494	0.499	334	301	6.6%	5.8%
Avg.comparison			1.093	1.000	1.015	1.000	1.049	1.000	6.070	1.000	6.48%	6.31%

insertion strategy can be tested. It is apparent that our algorithm performs better than [6]. The skew and max latency of [6] are reduced by 45.6% and 63.2% on average, respectively (Avg. comparison is calculated by $Avg(\sum(ours/[6])))$. In [5], an excellent flow CONTANGO for ISPD 2009 contest is completed. The flow before wire sizing addresses our slew constrained and obstacle-avoiding CTS problem with consideration about the signal polarity. Table 3 shows the skew result of our algorithm plus [5]'s signal polarity strategy. It is compared with the result after TBSZ displayed in Table III in [5]. The data reveals the advantages of our algorithm on skew by average 15.3% reduction.

In Table 4, the effectiveness of OBB is tested. Here OBB represents the flow proposed in this paper, and BB represents our flow with BB generating the tree's topology. Only four original benchmarks are listed because the results of OBB and BB on the rest benchmarks are identical. The reasons for no difference may be (1) no obstacles exist; (2) obstacles are too small; (3) obstacles are near the periphery. In one word, the obstacles do not affect the topology generation in the rest benchmarks. Benchmarks *ob04.in*, *ob07.in* in Table 4 are the same as *10f04* and *10f07* except that a large obstacle is inserted at the center part of the chip, respectively. From the table, skew is 27.1% improved on average than BB with a little more cost on power and max latency.

Table 5 shows the simulation results of all the benchmarks with obstacles. In order to show the effect of our obstacle-avoiding techniques, benchmarks are also tested with all obstacles removed. Most of the *Avg* numbers are calculated similar to Table 2 except for $Skew/MaxLatency$ which is the average of the numbers on its column. 6.48% and 6.31% show that though the skew with obstacles becomes 1.09X larger, the percentages they share the max latency is almost the same with the cost of power and max latency within 5%.

All the above experiments demonstrate that our algorithmic flow is effective and robust on obstacle-avoiding and slew constrained clock tree synthesis.

5. CONCLUSION

We have proposed a three-stage algorithm to generate a slew constrained and obstacle-avoiding buffered clock tree with skew optimization, which features the global perspective to deal with the obstacles' impaction on the skew. The experimental results in Section 4 show its effectiveness and robustness. In future, we will try to involve more considerations like wire sizing and process variation.

6. REFERENCES

[1] L.P.P.P. van Ginneken, Buffer placement in distributed RC-tree net- works for minimal Elmore delay, In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 865-868, 1990.

[2] J. Lillis, C. K. Cheng, and T. T. Lin, Optimal wire sizing and buffer insertion for low power and a generalized delay model, In *IEEE Journal of Solid-State Circuits*, vol. 31, pages 437-447, Mar. 1996.

[3] X.-W Shih, Y.-W. Chang, Fast timing-model independent buffered clock-tree synthesis, In *Proc. DAC*, pp. 80-85, 2010

[4] J. L. Tsai, T.-H. Chen, and C. C.-P. Chen, Zero skew clock tree optimization with buffer insertion/sizing and wire sizing, In *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 23, pages. 565-572, 2004.

[5] Dongjin Lee, I. L. Markov, CONTANGO: Integrated optimization of SoC clock network, In *Proc. DATE*, pp. 1468-1473, 2010

[6] Y.-Y Chen, C. Dong, Deming Chen, Clock tree synthesis under aggressive buffer insertion, In *Proc. DAC*, pp. 86-89, 2010

[7] W.-H. Liu, Y.-L. Li, H.-C. Chen, Minimizing clock latency range in robust clock tree synthesis, In *Proc. ASPDAC*, pp.389-394, 2010

[8] X.-W. Shih, C.-C. Cheng, Y.-K. Ho, and Y.-W. Chang, Blockage-avoiding buffered clock-tree synthesis for clock latency range and skew minimization, In *Proc. ASPDAC*, pp. 395-400, 2009

[9] T. H. Chao, Y. C. Hsu, J. M. Ho, K. D. Boese, and A. B.Kahng, Zero skew clock routing with minimum wirelength, In *IEEE Trans. Circuits Syst.*, vol. 39, pp. 799-814, 1992.

[10] Shiyan Hu, C. J. Alpert, J. Hu, S. Karandikar, Z. Li, Weiping Shi, and C. N. Sze, Fast algorithms for slew constrained minimum cost buffering, In *IEEE Trans. Computer-Aided Design*, vol.26, pp. 2009-2022, 2007.

[11] W. Shi and Z. Li, A fast algorithm for opitmal buffer insertion, In *IEEE Trans. Computer-Aided Design*, vol. 24, no. 6, pp. 879-891, 2005.

[12] Z. Li and W. Shi, An $O(bn^2)$ time algorithm for buffer insertion with b buffer types, In *Proc. DATE*, pp. 1324-1329, 2005.

[13] W. Shi, Z. Li, and C. J. Alpert, Complexity analysis and speedup techniques for optimal buffer insertion with minimum cost, In *Proc. ASPDAC*, pp. 609-614, 2004.

[14] Z. Li, W. Shi, An O(mn) Time Algorithm for Optimal Buffer Insertion of Nets With m Sinks, In *Proc. ASPDAC*, pp.320-325, 2006.