Fast Dual-Vdd Buffering Based on Interconnect Prediction and Sampling *

Yu Hu, King Ho Tam, Tom Tong Jing and Lei He
Electrical Engineering Department, University of California, Los Angeles
Los Angeles, CA 90095, USA
hu@ee.ucla.edu, ktam@ee.ucla.edu, tomjing@ee.ucla.edu, lhe@ee.ucla.edu

ABSTRACT

This paper presents fast algorithms for power optimized interconnect synthesis based on interconnect prediction and sampling considering dual V_{dd} buffers. We present three pruning techniques including interconnect prediction based pruning (*pre-buffer slack pruning* and *predictive min-delay pruning*) and sampling (*3D sampling*), of which 3D sampling is effective but the other two improve both efficiency and accuracy of sampling. We also show that the key to runtime reduction is to reduce the number of propagated options, while the sophisticated data-structures that have good amortized complexity do not necessarily reduce runtime. We obtain an empirically linear time algorithm with less than 1% of delay and power increase but over 50x speedup compared with the most efficient algorithm for dual V_{dd} buffer insertion. In addition, we further enhance the power optimized buffered tree construction by introducing routing grid reduction. We apply our speedup techniques to buffered tree construction algorithm. Experimental results show that we obtain over 100x speedup compared with the most efficient existing algorithms for dual V_{dd} buffered tree construction.

Categories and Subject Descriptors: B.7.2[Hardware]: Integrated circuits – Design aids

General Terms: Algorithms, design

Keywords: Interconnect, low power, dual-V_{dd}, buffer insertion, routing

1. INTRODUCTION

Interconnect optimization is a critical component of typical VLSI design flows for timing closure. However, delay-optimal buffer insertion [1] incurs high power overhead. It is possible to achieve low power buffer insertion to given routed tree topologies through utilizing timing slacks of tree branches. [2] developed a power-optimal buffer insertion algorithm. In [2], the number of sub-solutions (i.e. options) at each node grew in a pseudo-polynomial manner, as computation progressed from sinks to source. The runtime for large nets was unacceptably high due to the uncontrolled option increase. [3] assumed a large buffer library with near continuous buffer sizes, and solved the power-optimal buffer insertion problem with 5x speedup over [2] and negligible loss of delay and power optimality. [4] proposed a power optimized buffer insertion algorithm which also considers dual V_{dd} buffer insertion. They achieved 17x speedup with no delay penalty and about 1% loss of power optimality over [2] when single V_{dd} buffers were considered, and saved an extra 23% power when dual V_{dd} buffers were considered.

Simultaneous buffer insertion and tree topology generation has also been studied for delay optimization and more recently for power optimization. [5, 6] studied the buffered tree construction problem for multi-sink nets, without considering buffer stations (BS) or blockages. They constructed interconnect trees while exploring a few topologies for delay minimization, although the buffered routing was not necessarily delay-optimal. [7, 8] presented two construction approaches to account for blockages and BS and quickly explored a few alternative routes for the purpose of delay minimization. [9] presented a delay-optimal routing algorithm based on maze routing over Hanan grid which also considered BS and blockages, while [10] enhanced it with several speed-up techniques. [11] proposed an efficient algorithm for delay optimal buffered tree routing. [4] presented the first power optimized buffered routing algorithm considering either single or dual V_{dd}. It was based on routing over Hanan grid, and could handle up to a few sinks, same as in [9, 10], due to the explosion of the number of options.

In this paper, we study the dual V_{dd} buffering problem for power reduction, including dual V_{dd} buffer insertion and sizing ($dBIS$) and buffered tree construction ($dTree$). Based on dynamic programming, our proposed algorithms target with orders of magnitude speedup of the power optimized dual V_{dd} buffer insertion and buffered tree construction problem. Our major contributions include

1. proposing three speedup techniques for power optimized dual V_{dd} buffer insertion based on interconnect prediction and sampling, which are Pre-buffer Slack Pruning (PSP) extended from the one presented in [12, 13], Predictive Min-delay Pruning (PMP), and 3D sampling, of which 3D sampling is effective but the other two improve both efficiency and accuracy of sampling. The experimental results show that the runtime grows linearly with respect to the tree-size;

2. incorporating the fast buffer insertion and grid reduction in power optimized buffered tree construction to further speedup the power optimized buffered tree construction algorithm.

The experimental results show that we can obtain more than 50x and 100x speedup over DVB and D-Tree algorithms in [4], respectively with only 1% worse delay. We also expand the power

*This paper is partially supported by NSF CAREER award CCR-0306682/0401682 and a UC MICRO grant sponsored by Intel. Address comments to lhe@ee.ucla.edu.

SLIP'07, March 17–18, 2007, Austin, Texas, USA.

optimized routing capability to handle 10-sink nets, which cannot be handled by the D-Tree algorithm in [4].

The rest of the paper is organized as follows. Section 2 presents modeling and problem formulations for the dBIS and dTree problem. Our fast buffering algorithms and experimental results for dBIS and dTree are described in detail in Section 3 and Section 4, respectively. Section 5 analyzes Fast dBIS and Fast dTree algorithms. The paper is concluded in Section 6.

2. PRELIMINARIES

2.1 Delay, power, and slew modeling

We use a distributed Elmore delay model as in [7, 8, 9, 10]. The delay $d(l)$ due to a piece of wire with length l is given by

$$d(l) = (\frac{1}{2} \cdot c_w \cdot l + c_{load}) \cdot r_w \cdot l \quad (1)$$

where c_w and r_w are the unit length capacitance and unit length resistance of the interconnect, respectively and c_{load} is the capacitive loading at the end of the wire.

We also use Elmore delay times $ln9$ as the slew rate metric [14]. The delay of a buffer d_{buf} is given by

$$d_{buf} = d_{int} + r_o \cdot c_{load} \quad (2)$$

where d_{int}, r_o and c_{load} are the intrinsic delay, output resistance and capacitive loading at the output of the buffer, respectively.

In the context of buffer insertion with upper bound on slew rate, we observe that slew rates at the buffer inputs and the sinks are consistantly within a few tens ps of the upper bound. Therefore, we model buffer delay with negligible error by approximating the input slew rate using the upper bound as in [14].

We measure interconnect power by energy per switch. The energy per switch E_w for an interconnect wire of length l is

$$E_w = 0.5 \cdot c_w \cdot l \cdot V_{dd}^2 \quad (3)$$

We use a signle value E_{buf} to represent short-circuit and dynamic power consumed by a buffer into a single value.

2.2 Dual Vdd Buffering

Dual V_{dd} buffering uses both high V_{dd} and low V_{dd} buffers in interconnect synthesis. Designs using low V_{dd} buffers consume less buffer and interconnect power. Applying this technique to non-critical paths, we achieve power savings without increaseing the delay of the overall interconnect tree.

As in [4], we have the following constraint of dual V_{dd} buffers – we only allow high-V_{dd} buffers followed by low V_{dd} buffers. We assume that the driver at the source operates at high V_{dd} and V_{dd} level converters are only placed at high V_{dd} sinks driven by low V_{dd} buffers to avoid the power and delay overhead of level converters.

2.2.1 Dual Vdd buffer insertion

We assume that a loading capacitance and a required arrival time (RAT) q_n^s are given at each sink terminal n_s. We assume that a driver resistance at the source node n_{src} is given. We also assume that all types of buffers can be placed only at buffer candidate nodes n_b^k. We use the RAT at the source n_{src} to measure delay performance. Our goal is to minimize the power of the interconnect subject to the RAT constraint at the source n_{src}.

DEFINITION 1. *The required arrival time (RAT) q_n at node n is defined as*

$$q_n = \min_{n_s \forall s}(q_n^s - d(n_s, n)) \quad (4)$$

where $d(n_s, n)$ is the delay from the sink node n_s to node n.

Dual-V_{dd} buffer insertion and sizing (dBIS) – Given an interconnect fanout tree which consists of a source node n_{src}, sink nodes n_s, Steiner nodes n_p, candidate buffer nodes n_b and a connection topology among them, the dBIS problem is to find a buffer placement, a buffer size assignment and a V_{dd} level assignment solution such that the RAT q_n^{src} at the source n_{src} is met and the power consumed by the interconnect tree is minimized, while slew rate at every input of the buffers and the sinks n_s are upper bounded by the slew rate bound $\bar{s}$.

2.2.2 Dual Vdd buffered tree construction

We measure the delay and power performance using the same metric as in the dBIS formulation. Assuming that a floorplan of the layout is available, we can identify the locations and shapes of rectangular blockages and the locations of the buffer stations (BS) which are the allocated space for buffer insertion. Therefore we have the following problem formulation.

Dual V_{dd} Buffered Tree Construction (dTree) – Given the locations of a source node n_{src}, sink nodes n_s, blockages and BS, the dTree problem is to find the minimum power embedded rectilinear spanning tree with a buffer placement, buffer sizes and a V_{dd} assignment that satisfy the RAT q_n^{src} constraint at the source node n_{src} and the slew rate bound $\bar{s}$ at every input of the buffers and the sinks n_s.

3. FAST BUFFERING WITH SPEEDUP TECHNIQUES

A dynamic programming framework is used. Power-optimal solutions are constructed using partial solutions (i.e. options) from the subtrees. At each node of the given routing tree, a list of options for the sub-tree rooted at that node is generated by recursively traversing the tree in a bottom up fashion. In the dBIS problem, an option Φ_n at the node n is denoted as $\Phi = (rat, cap, pwr, \theta)$, where rat, cap, and pwr are the required arrival time, the downstream capacitance and the downstream sub-tree power dissipation at node n, and θ signifies whether there exists any high V_{dd} buffers at the downstream of node n. We say an option Φ is redundant if it is dominated by another option, and we can safely drop Φ without losing the optimality of the solution.

DEFINITION 2. *In node n, option $\Phi_1 = (rat_1, cap_1, pwr_1, \theta)$ dominates $\Phi_2 = (rat_2, cap_2, pwr_2, \theta)$, if $rat_1 \geq rat_2$, $cap_1 \leq cap_2$, and $pwr_1 \leq pwr_2$.*

The key to an efficient power-optimal buffer insertion algorithm is to reduce the number of options as early and as much as possible. The delay-optimal buffer insertion algorithm [1] creates as many options as the number of nodes, but this is no longer true in the power-optimal buffer insertion problem. [2] shows that the growth of the number of options is pseudo-polynomial. The option sampling technique in [4] bounds the growth of options at each node, which helps to reduce the number of options at the expense of optimality. In the following of this section, we first present a practical and efficient data structure for pruning. Then we propose an effective speedup approach, 3D sampling. After that, we employ two predition-based pruning rules, PSP and PMP, to further improve both accuracy and efficiency. At last, we present the Fast dBIS algorithm and give the experimental results.

3.1 Data structure for pruning

Advanced data-structures in [12] for delay-optimal buffer insertion cannot be applied to power-optimal buffer insertion as they only accomodate up to two option labels, which are RAT and capacitance. The fastest algorithm to-date for power-optimal buffer

insertion [4] makes use of augmented orthogonal search trees embedded in a balanced binary search tree (BST), which, however, does not necessarily benefit the runtime in practice. In this section, we analyze the statistics of options and provide an efficient and practical pruning structure (see Figure 1) for dBIS problem. We maintain a balanced binary search tree BST_n sorted by downstream capacitance of the options to store a non-redundant set of options at each node n. Each node $opList_c$ in BST_n is a set of (rat_n, pwr_n) pairs.

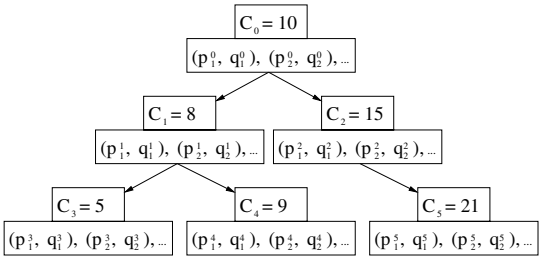


Figure 1: Data structure of dBIS problem

First, we analyze the index of options in pruning structure. Figure 2 shows the runtime of five test cases calculated by DVB in [4] with power indexed and capacitive indexed data structures, respectively. We can see that the power indexed data structure is much slower than the capacitive indexed counterpart. The reason is that, the number of distinct values of capacitance is generally smaller than that of power due to the presence of a slew rate bound, which results in less search trees in the pruning structure used in [4]. Therefore, we use a capacitive indexed binary search tree for the best runtime.

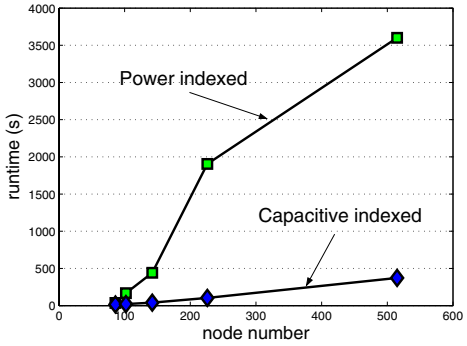


Figure 2: Runtime for power indexed and capacitive indexed organization

We also find that, using sophisticated orthogonal search trees like those in [2, 4] only speeds up a very small portion of all $opList_c$ operations while significantly increaseing the runtime overhead of other cases. On the contrary to [4], which expects a large option list $opList_c$ under each node in BST_n, the $opList_c$ list is quite small. Figure 3 shows the distribution of the number of options in all $opList_c$ for net s4. We can see that most $opList_c$ ($> 80\%$) contain less than 10 options. Therefore, we only need to maintain the BST_n in our implementation by keeping the (rat, pwr) tuples in $opList_c$ as linked lists, which has the lowest runtime and memory overhead.

3.2 3D sampling pruning

We extend the power-delay sampling [4] to 3D sampling to further control the growth of options of each tree node in the bottom-up sub-solution propagation process. Power-delay sampling in [4] picks a fixed number of options in each $opList_c$ in Figure 1 under each

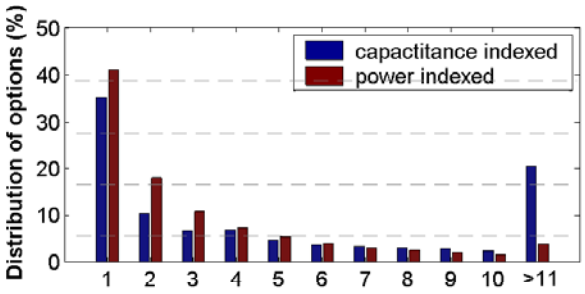


Figure 3: The distribution of options indexed by power and capacitance

capacitance, which has been shown to bring significant speedup. [4] claims that the number of distinct capacitive values is small when the distance between buffer insertion locations is uniform and the slew rate bound is tight. However, we observe that this number is not that small for large testcases. Table 1 shows the statistics of the percentage of the nodes carrying a large number of distinct capactive values for 4 nets. We can see from this table that over 50% of the nodes carry over 50 distinct capacitive values and over 10% of the nodes carry more than 100 capacitive values. When the scale of the test case becomes larger, tree nodes carry larger numbers of distinct capacitive values. Therefore, we need to explore more effective sampling rules by taking capacitance into consideration.

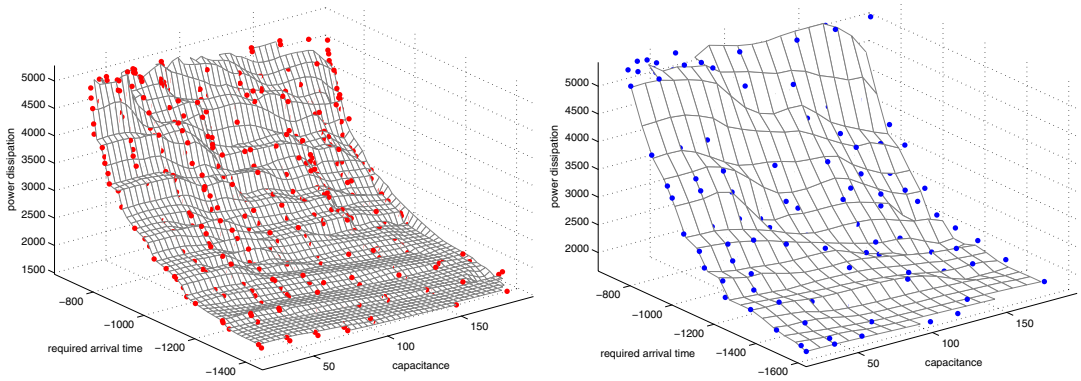
Table 1: The percentage of the nodes carrying a large number of distinct capactive values

node#	sink#	> 100	> 50
515	299	14%	62%
784	499	17%	64%
1054	699	28%	65%
1188	799	33%	71%

In 3D sampling, we get option samples based on power, delay, and capacitance. The idea is to pick only a certain number of options among all options uniformly over the power-delay-capacitance space for upstream propagation. Figure 4 shows (a) pre-sample and (b) after-sample option sets. Each dot corresponds to an option. In each tree node, we divide each dimension of the bounding box of all options into equal segments such that the entire power-delay-capacitance domain is superposed by a cubic grid. For each grid-cube shown in Figure 4 (a), we retain only one option if there is any. By also including the smallest power option and the largest RAT option for each capacitance value, we obtain the sampled non-dominated option set shown in Figure 4 (b).

Obviously, we can control the number of options in each tree node by 3D sampling. That is, given the number of sampling grids on one dimension, b, the upper bound of option number in a tree node is b^3, and the maximum number of options retained at all nodes is no more than $b^3 \cdot n$ for a n-node tree. As b is a constant, the growth of options is effectively linear for dBIS problem by using 3D sampling. If we treat the buffer library size as a constant, dBIS is expected to be solved in linear complexity with 3D sampling.

To test the efficiency of 3D sampling, we conduct some experiments and compare the results with DVB [4], which use only power-delay sampling. We use 9 test cases, s1–s9, generated by randomly placing source and sink pins in a 1cm x 1cm box. We use the GeoSteiner package [15] to generate the topologies of the test cases. The characters of our testcases are shown in Table 2. We will use these testcases in the rest of the paper. We also break interconnect between nodes longer than $500\mu m$ by inserting degree-2 nodes. We set the RAT at all sinks to 0 and the target RAT at the source to $101\% \cdot RAT*$, where $RAT*$ is the maximum achievable RAT at the source, so that the objective becomes minimizing the



(a) Before sampling (b) After sampling

Figure 4: 3D sampling for non-redundant options

power under 1% delay slack. We use the same technology related settings as in [4]. The slew rate bound $\bar{s}$ is set to $100ps$.

We have made buffers using an inverter cascaded with another inverter that is four times larger. There are 6 buffers (high V_{dd} and low V_{dd} buffers of 16x, 32x, and 64x) in our buffer library. 20x20 and 20x20x20 sampling grids are used in DVB and 3D sampling, respectively. (We will evaluate the effectiveness of the size of the sampling grids in Section3.6). All experiments are performed on Linux with Intel PM 1.4 Ghz CPU and 1Gb memory.

Table 2 shows the comparison results of both solution quality and runtime between the power-delay sampling in DVB [4] and 3D sampling. To make comparison clear, we normalize all values to DVB. Compared with the DVB, 3D sampling can achieve 9x speedup on average. However, 3D sampling introduces 6% increase for delay[1], which makes it prohibitive to simply employ 3D sampling in practical usage. In the next sections, we will present two additional pruning techniques. By employing them along with 3D sampling, we can achieve further speedup associated with improvement on accuracy.

Table 2: 3D sampling vs. power-delay sampling

net	node#	sink#	speedup	delay	power
s1	86	19	3x	1.01	1.06
s2	102	29	3x	1.01	0.98
s3	142	49	2x	1.02	0.94
s4	226	99	3x	1.02	1.00
s5	375	199	8x	1.05	0.99
s6	515	299	8x	1.06	1.04
s7	784	499	9x	1.08	0.98
s8	1054	699	10x	1.08	0.99
s9	1188	799	13x	1.08	1.04
ave	497	299	9x	1.06	1.00

3.3 Pre-buffer slack pruning

The aggressive pre-buffer slack pruning (APSP) in [16, 13] prune dominated options by predicting upstream buffer delay. Therefore, we "preview" the relative optimality of the current options at a node, which allows us to drop options that will be dominated after propagation. This prevents options from being populated at the upstream and therefore helps to reduce the time complexity.

Pre-buffer Slack Pruning (PSP): Suppose R_{min} is the minimal resistance in the buffer library. For two non-redundant options $\Phi_1 = (rat_1, cap_1, pwr_1, \theta_1)$ and $\Phi_2 = (rat_2, cap_2, pwr_2, \theta_2)$, where $rat_1 < rat_2$ and $cap_1 < cap_2$, then Φ_2 is pruned, if $(rat_2 - rat_1)/(cap_2 - cap_1) \geq R_{min}$.

R_{min} refers to the minimal resistance of the buffer library for single V_{dd} buffers, and has to be redefined for dual V_{dd} buffer insertion for optimal pruning. To handle dual V_{dd} buffers, we choose a proper high/low V_{dd} buffer resistance R_H/R_L for PSP. When there exists some high V_{dd} buffers in the downstream of the current option $\Phi = (rat, cap, pwr, \theta)$, i.e. $\theta = true$, we use R_H in PSP. Otherwise, we use R_L. As $\theta = true$ indicates no low V_{dd} buffer is to be placed in upstream, it is overly aggressive to perform PSP by using R_L ($> R_H$). On the other hand, it makes PSP more effective (to prune more) by using R_L if there is no high V_{dd} buffer downstream from Φ. To make the algorithm even faster, we may use a resistance larger than R_{min} (i.e. Aggressive Pre-buffer Slack Pruning (APSP)). [13] shows that we can get substantial (more than 50%) speedup at a cost of 5% loss of optimality for min-cost (buffer number) buffer insertion problem. As the number of options in the dBIS problem is much larger than that of the min-cost problem, we expect more speedup from using PSP.

3.4 Predictive min-delay pruning

We also try to predict whether the option leads to a valid solution at the source by introducing the predictive min-delay pruning (PMP). This rule makes use of analytical formulae to calculate the lower bound of delay from any node to the source, which assumes a continuous number of buffers and buffer sizes. If such delay does not meet the delay specification at the source, the option is dropped to avoid unyielding option propagation. Consider an interconnect segment of unit length resistance r and unit length capacitance c. It is driven by a buffer of size s with unit driving resistance r_s, unit input capacitance c_p, and unit output capacitance c_o. We assume that the interconnect (with length l) is terminated at the other end with another repeater of identical size. [14] presented that the unit length delay is optimal when

$$l_{opt} = \sqrt{\frac{2r_s(c_o + c_p)}{rc}}, s_{opt} = \sqrt{\frac{r_s c}{r c_o}} \quad (5)$$

where l_{opt} and s_{opt} are the optimal buffer insertion length and the optimal buffer size, respectively.

The optimum unit length delay $delay_{opt}$ is given by

$$delay_{opt} = 2\sqrt{r_s c_o rc}(1 + \sqrt{\frac{1}{2}(1 + \frac{c_p}{c_o})}) \quad (6)$$

We pre-compute a unit length minimum delay table indexed by

[1] In some cases, such as s2 in Table 2, the power dissipation produced by 3D sampling is even smaller than the optimal one. Theoretically, power dissipation decreases while delay slack becomes large, hence it is possible that smaller power dissipation can be expected with unoptimal approaches.

buffer, unit length resistance and capacitance, and the path length from the source to each tree node. We assume high V_{dd} buffers to calculate the unit length minimal delay, such that we get a lower bound when both high V_{dd} and low V_{dd} buffers are used. We define PMP as

Predictive Min-delay Pruning (PMP) – Given a required arrival time RAT_0 at the source, for a tree node v, its upstream delay lower bound is given by $dlb(v) = delay_{opt} \cdot dis(v)$, where $dis(v)$ is the distance of the path from the source to node v in the routing tree. A newly generated option $\Phi = (rat, cap, pwr, \theta)$ is pruned if $rat - dlb(v) < RAT_0$.

We arrive at some interesting observations about PMP through extensive experiments. We note that PMP prunes more options when RAT_0 is larger (i.e. the delay constraint is tight). Therefore, PMP essentially prevents unnecessary solution exploration when there is little room for power optimization. We have also explored enhancing PMP by considering the theorectical minimum power buffered interconnect from analytical methods [17]. We define the following pruning rule:

Predictive Min-power Pruning (p-PMP) – Given two options $\alpha_1 = (pwr_1, rat_1, cap_1)$ and $\alpha_2 = (pwr_2, rat_2, cap_2)$, α_1 can be pruned if $pwr_1 + pre\_pwr_1 > pwr_2$ and $rat_1 + pre\_d_1 < rat_2$, where pre_pwr_1 and pre_d_1 are the min-power and min-delay between the source and the current node.

However, our experimental experience shows that the small extra gain in pruning power from p-PMP does not justify the overhead of table lookup and additional calculation. To perform p-PMP, we pre-calculate the unit length min-delay table as in PMP. In addition, we also need to prepare another table to store the unit length min-power with respect to the timing slack available, which yields a big table indexed by V_{dd}, buffer size and slack. We have performed a few experiments using the p-PMP rule. For instance, we test s4 (a 99-sink net with 137 nodes) by PMP and p-PMP, respectively. We have found that p-PMP only prunes 3% more options while the runtime with the p-PMP rule takes 2x longer. We observe that the analytical min-power buffered interconnect tends to give a very loose lower bound for power and is therefore not effective for the purpose of pruning.

3.5 Fast buffer insertion algorithm for dBIS problem (Fast dBIS)

We integrate our new pruning rules with the DVB algorithm proposed in [4], which is summarized in the pseudocode of Alg. 1. An option is denoted as $\Phi = (c, p, q, \theta)$, where c, p, q and θ correspond to cap, pwr, rat and θ in Section 3.1, respectively. Moreover, we use c_b^k, E_b^k, V_b^k, and $d^b(c_{load})$ to denote input capacitance, power, V_{dd} level, and delay with output load c_{load} of the buffer b_k. $d_{n,v}$ and $E_{n,v}(V)$ are the delay and the power of the interconnect between node n and node v operating at voltage V. The set of available buffers $Set(B)$ contains both low V_{dd} and high V_{dd} buffers. We first call **Fast-dBIS** at the source node n_{src}, which recursively visits the child nodes (line 2) and enumerates all possible options (line 6–19) in a bottom up manner until the entire tree is traversed.

To speedup the algorithm, we call **3DSampling** in line 3 to apply our 3D sampling heuristic on the returned options from the children nodes. When a new option is generated (line 9–14), we test the redundancy of this option based on **PMP** and **PSP** (line 15–16). If it is not redundant, we use this option to prune others based on **PSP** (line 17–18).

3.6 Studies on the speedup techniques

To evaluate the speedup capability and the effect on the solution qualities of our speedup techniques (PSP, PMP and 3D sampling),

Algorithm 1: **Fast-dBIS** (T_n)

```
1: Set(Φ_n) = (c_n^s, 0, q_n^s, false) if n is a sink, else (0,0,∞,false)
2: for each child v of n
3:   Set(Φ_v) = 3DSampling (Fast-dBIS(v))
4:   Set(Φ_temp) = Set(Φ_n)
5:   set(Φ_n) = φ
6:   for each Φ_i ∈ Set(Φ_v)
7:     for each Φ_t ∈ Set(Φ_temp)
8:       for each buffer b_k ∈ Set(B)
9:         if b_k = φ
10:          V_n = V_H if θ_i or θ_t is true, else V_L
11:          Φ_new=(c_i + c_t, p_i + p_t + E_{n,v},
                    min(q_t, q_i − d_{n,v}), θ_i or θ_t)
12:        else if i. V_b^k is high; or ii. V_b^k is low and θ_i is false
13:          Φ_new=(c_i + c_t, p_i + p_t + E_{n,v}(V_b^k) + E_b^k,
                    min(q_t, q_i − d_{n,v} − d_b^k(c_i + c_{n,v})),
                    θ_t or (if V_b^k = V_H))
14:        else goto line 8
15:        if i. slew rate violation at downstream buffers; or
             ii. Φ_new is redundant (by PMP); or
             iii. Φ_new is dominated by any Φ_z ∈ Set(Φ_n) (by PSP)
16:          drop Φ_new
17:        else
18:          remove all Φ_z ∈ Set(Φ_n) dominated by Φ_new(by PSP)
19:          Set(Φ_n) = Set(Φ_n) ∪ Φ_new
20: Return Set(Φ_n)
```

we run Fast dBIS by using each of them individually. In PSP, we use the 16x buffer (high V_{dd} and low V_{dd}) to perform pruning. In PMP, we calculate the unit-length-min-delay with the settings under $65nm$ technology node [18]. To avoid over aggressive pruning, we use $105\% RAT*$ as the RAT_0 in PMP. To compare the solution qualities, we run power-optimal buffer insertion (PB) algorithm [2]. (We modified PB to handle dual V_{dd} buffer insertion.)

Due to the limitation of the computational capability of PB, we do not give the large scale results. Table 3 shows the comparisons of runtime and solution qualities. Note that PSP and PMP give the same solution as PB, so we list the results of PB/PSP/PMP together in Table 3. From Table 3, we can find that PSP and PMP (column psp and pmp) can achieve slight speedup over PB (column PB) without loss of optimality. DVB [4] (column DVB) and 3D sampling (column sam) can achieve tens of times speedup while introducing some increase of delay and power (within 3% for small scale testcases).

We then test Fast dBIS to study on the effectiveness of combining sampling and the two prediction based pruning rules (PSP and PMP). 20x20x20 grid are used for 3D sampling. The results are shown in Table 4, where 'sam', 'ms', 'ss' and 'all' stand for 3D sampling, pmp+3D sampling, psp+3D sampling and combining all speedup techniques, respectively. We also list the results produced by employing 3D sampling only in Table 4. To make the comparison clear, all results (delay and power) are normalized based on solutions prodcued by DVB. From Table 4, we make the following interesting observations.

(A) 3D sampling itself (column sam) introduces large increase of delay and power for large testcases, though it brings substantial speedup.

(B) Both runtime and solution quality are improved by combining PSP and 3D sampling (column psp+sam). We get over 2x speedup on average. However, there still exist some increase of delay and power for large scale testcases (s7, s8 and s9).

(C) When we combine PMP and 3D sampling together (column pmp+sam), much better solutions are obtained. This is because PMP prunes many redundant options and keeps a bound of delay for existing options, so that 3D sampling can always select option samples from a good candidate pool, which

Table 3: Comparisons of individual speedup techniques and baseline algorithm (PB)

	runtime (s)					$RAT*$ (ps)			power (fJ)		
net	PB[2]	pmp	psp	DVB[4]	sam	PB/pmp/psp	DVB	sam	PB/pmp/psp	DVB	sam
s1	176	184	146	36	15	-1444.99	-1444.99	-1465.03	13075.5	13077.6	13798.7
s2	329	276	235	62	19	-1600.45	-1600.45	-1619.07	14521	14611.3	14379.8
s3	2356	1759	1416	96	36	-2221.89	-2211.89	-2268.45	20411.1	20452.2	19478.4
s4	>10000	8390	4391	264	50	-1804.55	-1804.55	-1841.09	29641.7	30066.9	28828.4
	1	3/4	1/2	1/40	<1/100	1	1.00	1.02	1	1.01	0.97

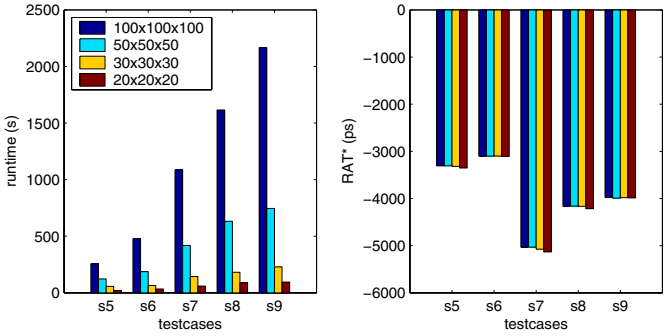
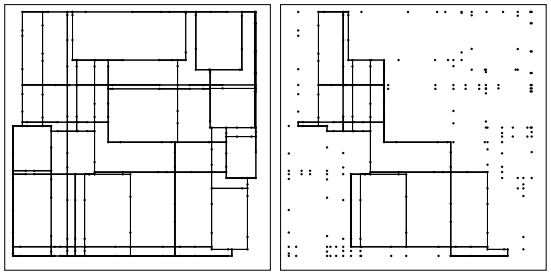


Figure 5: The effect of the grid size in sampling



(a) Original grid (b) After reduction

Figure 6: Escape grid reduction for testcase grid.5

makes the solution quality improved as well as providing some speedup.

(D) By combining PSP, PMP and 3D sampling together (column *all*), Fast dBIS achieves the best performance on both solution quality and runtime compared with sam, psp+sam and pmp+sam. Similar to (C), PSP and PMP prune redundancy as much as possible, which means we can always sample good candidates.

(E) Both (PSP+3D) and (PMP+3D) can run much faster than (3D) for small testcases (s1 and s2), but the speedup ratio degrades for larger testcases. This is because there exist few options in each node of the buffered tree for small testcases, and 3D sampling will not work until the sampling bound is reached.

To further study the effect of grid size in 3D sampling, we change the sampling grid size and collect the experimental results data, where PSP and PMP are employed with 3D sampling, and 5 larger scale testcases are tested. Figure 5 gives us a clear view of the effect of the different grid sizes for sampling on runtime and solution quality. We find that the runtime is reduced substantially (20x faster) when we use sparser sampling grids. On the other side, there is only little loss of solution quality (within 1%) when the sampling grid becomes sparser. In practice, we find that a grid size of 20x20x20 can get a good tradeoff between runtime and accuracy. Also, we find that Fast dBIS shows an effective linear runtime because of the bound of option number by 3D sampling.

4. SPEEDUP TECHINIQUES FOR DTREE

As the starting point, we build a grid using the "escape node algorithm" in [10], and then generate an *escape grid* by looking for intersection points between buffer stations and the grid lines. An escape grid, or Hanan grid, is formed by shooting horizontal and vertical lines from net terminals. The intersections of these grid lines form Steiner points, which does not allow buffer insertion in our formulation. We insert buffer insertion points whenever a grid line hits a buffer station, which are rectangular regions scattered across the floorplan. In the tree growing process in D-Tree, we need to record all non-redundant options in each node of the escape gird. To keep track of the sinks and the other nodes that the current options covered (to avoid cycles), each option needs to store a sink set $\mathcal{S}$ (i.e., all sinks that are covered by the current option) and a reachability set $\mathcal{R}$ (i.e., all nodes that are covered by the current option). An option for D-Tree is denoted as $\Phi = (\mathcal{S}, \mathcal{R}, rat, cap, pwr, \theta)$, and we redefine the *domination* of two options as

DEFINITION 3. *In node n, option $\Phi_1 = (\mathcal{S}_1, \mathcal{R}_1, rat_1, cap_1, pwr_1, \theta_1)$ dominates $\Phi_2 = (\mathcal{S}_2, \mathcal{R}_2, rat_2, cap_2, pwr_2, \theta_2)$, if $\mathcal{S}_1 \supseteq \mathcal{S}_2$, $rat_1 \geq rat_2$, $cap_1 \leq cap_2$, and $pwr_1 \leq pwr_2$.*

In each node of escape grid, options are divided into subsets indexed by covered sink sets. Under each subset, a balanced search tree (see Section 3.1) is maintained. Once a new option $\Phi = (\mathcal{S}, \mathcal{R}, rat, cap, pwr, \theta)$ is generated in an escape grid node, the most desirable option pruning strategy is to test the redundancy of Φ in all subsets indexed by the sink set $\mathcal{S}_i \supseteq \mathcal{S}$. However, we can have up to 2^n (where n is the number of sinks of the net) sink sets in a node, which we cannot afford to search for all related sink subsets for each option creation. In our implementation, we check if any options in the full sink set (i.e., the sink set that includes all sinks) dominate Φ, and if Φ dominates any option under its own sink set $\mathcal{S}$.

In addition to the speedup techniques presented in Section 3, we also apply the following heuristic to further narrow the search space.

4.1 Escape grid reduction

As the number of options grows exponentially with of the number of grid nodes, we can reduce the number of options substantially by using grid reduction. Inspired by PMP proposed in Section 3.4, we retain those grid nodes p such that $dis(p, n_s^i) + dis(p, n_{src}) = dis(n_s^i, n_{src})$ for any sink n_s^i, where n_{src} is the source and $dis(x, y)$ is the path length from node x to node y. This rule implies that we delete all grid nodes that are not in any rectangles formed by any sink-source pairs. This is reasonable in buffer tree construction since a long-distance wire snaking is harmful to delay and power. We note that grid reduction may sometimes hamper the routability, as the within-bounding box grids get completely blocked by obstacles. To tackle this, we make the sink-source bounding boxes larger in progressive steps until we get a connected reduced escape grid. We modify the reduction rule as: retain those grid nodes p such that $dis(p, s_i) + dis(p, source) = dis(s_i, source) + j \cdot dieSize/10$ for any sink s_i, where $j = 1, 2, \cdots$. As shown in Figure 6, (a) is a full escape grid for testcase grid.5 (see Table 5) and (b) is a reduced grid from (a).

Table 4: Combine sampling and other pruning rules togeter and compare to DVB

	runtime (s)					delay				power			
net	DVB	sam	pmp+sam	psp+sam	all	sam	pmp+sam	psp+sam	all	sam	pmp+sam	psp+sam	all
s1	36	15	2	3	1	1.01	1.01	1.01	1.01	1.06	1.00	1.06	0.99
s2	62	19	4	5	2	1.01	1.00	1.01	1.00	0.98	1.01	0.97	1.00
s3	96	36	10	7	4	1.02	1.00	1.01	1.00	0.94	0.98	0.96	0.98
s4	264	50	16	14	6	1.02	1.00	1.01	1.00	1.00	0.99	1.01	1.04
s5	640	71	46	32	20	1.05	1.00	1.03	1.01	0.99	0.99	0.95	0.98
s6	987	101	77	42	34	1.06	1.01	1.03	1.01	1.04	1.00	1.05	1.01
s7	2232	209	135	80	59	1.08	1.00	1.06	0.99	0.98	0.95	1.00	0.99
s8	3427	309	219	127	89	1.08	1.00	1.07	1.00	0.99	1.00	0.95	0.97
s9	5625	327	256	133	95	1.08	1.01	1.08	1.01	1.04	1.03	1.02	1.03
ave	1485	128	85	49	34	1.06	1.00	1.04	1.00	1.00	0.99	1.00	1.00
	1	1/10	1/15	1/30	1/50								

4.2 Fast buffered tree construction algorithm for dTree problem (Fast dTree)

We apply our grid reduction technique in addition to the pruning heuristics to the D-Tree algorithm [4]. The pseudocode is given in Alg. 2. Our Fast dTree algorithm starts by building a grid using the escape node algorithm in [10] (line 1). It then performs our escape grid reduction heuristic (line 2). A queue Q is maintained for options that need to be propagated. Options stop propagating once the source is reached. Each time an option is popped from Q, it tries to propagate to all its neighbors (line 9–28). As we enumerate all possible topologies of the routing tree in our algorithm, the number of options grows exponentially, even with pruning strategies. In our implementation, dominated options under our pruning rules (line 19–20) or filtered options by 3D sampling (line 12) are freed to save memory.

Algoritm 2. **Fast-dTree** $(n_{src}, Set(n_s), Set(Blockage), Set(BS))$

1: $\{Set(n), \aleph(Set(n))\} = Grid(Set(n), Set(Blockage), Set(BS))$
2: **Grid_Reduction**$(Set(n_p), \aleph(Set(n)))$
3: for each sink $n_s \in Set(sinks)$
4: $\quad \Phi_n^s = (\{n_s\}, \{n_s\}, c_n^s, 0, q_n^s)$
5: $\quad Set(n_s) = \{\Phi_n^s\}$
6: $\quad$ push Φ_n^s into Q
7: while $Q \neq \phi$
8: $\quad$ if (Φ_n^{cur} = pop Q) has been dropped, continue
9: $\quad$ for each neighbor $n_j \in \aleph(n_{cur})$
10: $\quad\quad$ for each subset $Set(\Phi_n^j)[S_i]$ indexed by sink set S_i in n_j
11: $\quad\quad\quad$ if $S_i \cap \Phi_n^{cur}.\mathcal{S} \neq \phi$
12: $\quad\quad\quad\quad \{Set_{samples}\} =$ **3DSampling**$(Set(\Phi_n^j)[S_i])$
13: $\quad\quad\quad\quad$ for each option $\Phi_n^j \in Set_{samples}$
14: $\quad\quad\quad\quad\quad$ if $\Phi_n^j.\mathcal{R} \cap \Phi_n^{cur}.\mathcal{R} \neq \phi$
15: $\quad\quad\quad\quad\quad\quad$ form Φ_{new} similar to line 8 to 14 in Fast-dBIS
16: $\quad\quad\quad\quad\quad\quad \Phi_{new}.\mathcal{R} = (\Phi_n^j.\mathcal{R}) \cup (\Phi_{new}.\mathcal{R})$
17: $\quad\quad\quad\quad\quad\quad \Phi_{new}.\mathcal{S} = (\Phi_n^j.\mathcal{S}) \cup (\Phi_{new}.\mathcal{S})$
18: $\quad\quad\quad\quad\quad\quad$ if i. slew rate violation at downstream buffers; or
19: $\quad\quad\quad\quad\quad\quad\quad$ ii. Φ_{new} is redundant (by **PMP**); or
20: $\quad\quad\quad\quad\quad\quad\quad$ iii. Φ_{new} dominated (by **PSP**)by any
21: $\quad\quad\quad\quad\quad\quad\quad\quad \Phi_n^j : (\Phi_{new}.\mathcal{S}) \subseteq (\Phi_n^j.\mathcal{S}), \Phi_n^j \in Set(\Phi_n^j)$
22: $\quad\quad\quad\quad\quad\quad\quad$ drop Φ_{new}
23: $\quad\quad\quad\quad\quad\quad$ else
24: $\quad\quad\quad\quad\quad\quad\quad$ remove $\{\Phi_n^j : (\Phi_{new}.\mathcal{S}) \supseteq (\Phi_n^j.\mathcal{S}), \Phi_n^j \in Set(\Phi_n^j)\}$
25: $\quad\quad\quad\quad\quad\quad\quad\quad$ dominated by Φ_{new}
27: $\quad\quad\quad\quad\quad\quad\quad$ $Set(\Phi_n^j) = Set(\Phi_n^j) \cup Set(\Phi_{new})$
28: $\quad\quad\quad\quad\quad\quad\quad$ push Φ_{new} into Q if $n_j \neq n_{src}$

4.3 Experimental results of Fast dTree

For buffered tree construction, we create 6 test cases by randomly generating source and sink pins in a $1cm \times 1cm$ box. We also randomly generate blockages so that it consumes approximately 30% of the total area of the box. Horizontal and vertical BS are randomly scattered in the box so that the average distance between two consecutive BS is about 1000um. All other settings are the same as those in Section 3.2.

Table 5 shows the comparison betwen our Fast dTree algorithm and the S-Tree/D-Tree algorithms [4]. The experimental results show that our Fast sTree(single-V_{dd} version)/dTree (column "sTree/dTree") runs over 100x Faster than S-Tree/D-Tree with solutions, while having only 1% larger power than that produced by S-Tree/D-Tree. Fast dTree can get a solution for 10-sink net among 426 nodes grids in about one hour, while S-Tree/D-Tree fails to finish routing after one day. Moreover, we see the speedup obtained by our grid reduction heuristic from this table. Column "nl" shows the number of nodes left after grid reduction. Column "unreduced" shows the runtime without grid reduction. We find that the grid reduction achieves about 2x speedup for the first 5 test cases. As for the largest test case (426 nodes and 10 sinks), we cannot even get a solution without grid reduction. Note again that the largest examples that can be routed by existing delay-optimal [9, 10] and power-optimal [4] methodologies are only up to 6-sink nets.

5. ALGORITHMS ANALYSIS

As we have mentioned in the previous section, the key to runtime reduction is to reduce propagated options in the algorithm. In Section 3.2, 3D sampling gives a constant upper bound on the number of options in each tree node. Therefore, the growth of options in Fast dBIS is effectively linear. Figure 7 shows the number of non-redundant options generated by DVB [4], PSP+DVB, PSP+PMP+DVB, and Fast dBIS (PSP+PMP+3D sampling), respectively. We find that the increase of the options in Fast dBIS is much slower than that in DVB, and it increases in a nearly linear fashion in Fast dBIS, which demonstrates the effectiveness of 3D sampling with enhancement by PSP and PMP pruning as discussed in Section 3. Since each node now has roughly the same number of options, it therefore takes approximately the same time to propagate all options from one node to the other, making the runtime growth linear with respect to the tree size. Figure 8 shows the runtime growth trend with respect to the number of nodes, and it is clear that Fast dBIS has a roughly linear runtime complexity.

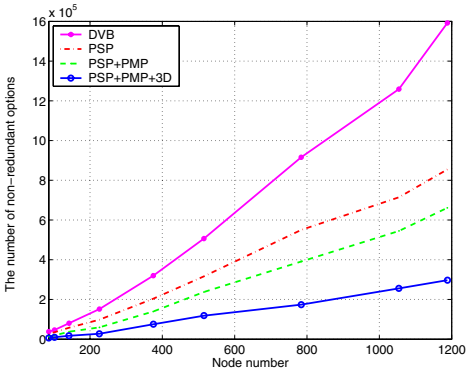


Figure 7: The option increase trends under different pruning strategies

Applying all proposed pruning techniques and the grid reduction heuristic in the Fast dTree algorithm helps significantly to reduce the

Table 5: Comparison of runtime and performance for buffered tree construction (D-Tree vs. Fast dTree)

test cases				runtime(s)					RAT*(ps)		power(fJ)			
name	n#	s#	nl#	S-Tree	sTree	D-Tree	dTree	unreduced	S-Tree	sTree	S-Tree	sTree	D-Tree	dTree
grid.2	97	2	36	0	0	0	0	0	-223	-224	1492	1492	1430	1430
grid.3	165	3	142	19	1	102	5	8	-604	-608	3908	3456	3907	3456
grid.4	137	4	82	44	2	297	8	23	-582	-583	3426	3426	3131	3131
grid.5	261	5	162	2849	8	5088	37	65	-532	-533	4445	4355	3979	3989
grid.6	235	6	143	5200	25	13745	115	193	397	-399	4919	4718	4860	3718
grid.10	426	10	267	-	2346	-	3605	-	-	-625	-	7338	-	5915
				1	$< 1/100$	1	$< 1/100$		1	$> 99\%$	1	$< 101\%$	1	$< 101\%$

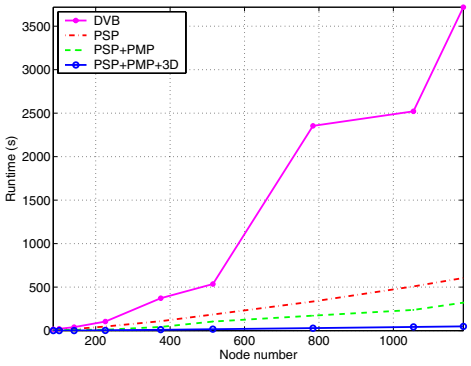


Figure 8: The runtime trends under different pruning strategies.

number of options during runtime. This is evident from the fact that the Fast dTree algorithm can handle up to 10-sink nets as opposed to only a few sink nets in [4], which has been demonstrated in Section 4.3. However, the path search problem is intrinsically an NP-hard problem, therefore the runtime increases exponentially with the number of sinks, as the number of options with non-overlapping or partially-overlapping sink set increases exponentially. Therefore, Fast dTree remains an exponential algorithm. However, for those routing nets of over 10 sinks, we can use the hierarchical approach similar to C-Tree [19], i.e., we decompose the large net into several small nets (<10 pins), and use our approach to compute the solution recursively.

6. CONCLUSIONS

We have presented efficient algorithms to buffer insertion and buffered tree construction problems considering dual V_{dd} buffers for power optimization. We have presented three pruning techniques including interconnect prediction based pruning (*pre-buffer slack pruning* and *predictive min-delay pruning*) and sampling (*3D sampling*), of which 3D sampling is effective but the other two improve both efficiency and accuracy of sampling. Also we show that the sophisticated data-structures that have good amortized complexity do not necessarily reduce runtime, and the key to runtime reduction is to reduce propagated options. In addition to the above techniques, we further speedup buffered tree construction by introducing routing grid reduction. Experimental results show that we obtain over 50x and 100x speedup compared with the most efficient existing algorithms for dual V_{dd} buffer insertion and buffered tree construction, respectively. In the future, we will further improve the efficiency of buffered tree construction.

7. REFERENCES

[1] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," in *ISCAS*, pp. 865–868, 1990.

[2] J. Lillis, C. Cheng, and T. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," in *ICCAD*, Nov. 1995.

[3] R. Rao, D. Blaauw, D. Sylvester, C. Alpert, and S. Nassif, "An efficient surface-based low-power buffer insertion algorithm," in *ISPD*, Apr 2005.

[4] K. Tam and L. He, "Power optimal dual-vdd buffered tree considering buffer stations and blockages," in *DAC*, Jun 2005.

[5] T. Okamoto and J. Cong, "Buffered Steiner tree construction with wire sizing for interconnect layout optimization," in *ICCAD*, Nov. 1996.

[6] J. Lillis, C. Cheng, and T. Lin, "Simultaneous routing and buffer insertion for high performance interconnect," in *GLVLSI Symp.*, 1996.

[7] C. Alpert, G. Gandham, J. Hu, J. Neves, S. Quay, and S. Sapatnekar, "Steiner tree optimization for buffers, blockages and bays," in *ISCAS*, May 2001.

[8] J. Hu, C. Alpert, S. Quay, and G. Gandham, "Buffer insertion with adaptive blockage avoidance," *TCAD*, vol. 22, no. 4, pp. 492–498, 2003.

[9] J. Cong and X. Yuan, "Routing tree construction under fixed buffer locations," in *DAC*, Jun 2000.

[10] W. Chen, M. Pedram, and P. Buch, "Buffered routing tree construction under buffer placement blockages," in *ASP-DAC*, Jan 2002.

[11] S. Dechu, Z. C. Shen, and C. Chu, "An efficient routing tree construction algorithm with buffer insertio n, wire sizing and obstacle considerations," *TCAD*, vol. 24, no. 4, pp. 600–608, 2005.

[12] W. Shi and Z. Li, "An o(nlogn) time algorithm for optimal buffer insertion," in *DAC*, Jun 2003.

[13] Z. Li, C. Sze, C. Alpert, J. Hu, and W. Shi, "Making fast buffer insertion even faster via approximation techniques," in *ASP-DAC*, Jan 2005.

[14] H. B. Bakoglou, *Circuits, Interconnections, and Packaging in VLSI*. Reading, MA: Addison-Wesley, 1990.

[15] D. Warme, P. Winter, and M. Zachariasen, "Geosteiner," in *http://www.diku.dk/geosteiner*, 2003.

[16] W. Shi, Z. Li, and C. Alpert, "Complexity analysis and speedup techniques for optimal buffer insertion with minimum cost," in *ASP-DAC*, Jan 2005.

[17] K. Banerjee and A. Mehrotra, "A power-optimal repeater insertion methodology for global interconnects in nanometer designs," *TCAD*, vol. 49, no. 11, pp. 2001–2007, 2002.

[18] Semiconductor Industry Association, *ITRS*, 2003.

[19] C. J. A. et al, "Buffered steiner trees for difficult instances," *TCAD*, vol. 21, no. 1, pp. 3–14, 2002.