Fast Algorithms for Power-optimal Buffering

Yu Hu and Jing Tong
Electrical Engineering Dept.
Tsinghua University, Beijing, P.R. China
{matrix98, jingtong}@tsinghua.edu.cn

King Ho Tam and Lei He
Electrical Engineering Dept.
Univ. of California, Los Angeles, CA 90095, USA
{ktam, lhe}@ee.ucla.edu

ABSTRACT

This paper presents efficient algorithms to tackle the power optimal buffer insertion and buffered routing tree construction problems using dual V_{dd} buffers. We show that the sophisticated data-structures which has good amortized complexity do not necessarily benefit the runtime, and that the key to runtime reduction is to reduce propagated options. We present three speedup techniques, namely pre-buffer slack pruning, predictive min-delay pruning, and 3D sampling, and obtain an effectively linear time algorithm at 1% and 2% of delay and power optimality loss, respectively. In addition to these, we enhance the power-optimal buffered tree construction by introducing routing grid reduction. Experimental results show that we obtain over 50x and 100x speedup compared to the most efficient existing algorithms for dual V_{dd} buffer insertion and buffered tree construction respectively in the literature to-date.

1. INTRODUCTION

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

Simultaneous buffer insertion and tree topology generation has also been studied for delay optimization and more recently for power optimization. [5, 6] study buffered tree construction problem for multi-sink nets, without considering buffer stations (BS) or blockages. They construct interconnect trees while explore a few topologies for delay minimization, although the buffered routing is not necessarily delay-optimal. [7, 8] present two construction approaches to account for blockage and BS and quickly explore a few alternative routes for the purpose of delay minimization. [9] presents a delay-optimal routing algorithm based on maze routing over Hanan grid which also considers BS and blockages, while [10] enhances it with several speed-up techniques. [4] presents the first power-optimal buffered routing algorithm again based on routing over Hanan grid, but it can only handle nets of only up to a few sinks, as in [9, 10], due to the explosion of the number of options.

In this paper, we study the power-optimal dual V_{dd} buffer insertion and buffered tree construction problems. Our proposed algorithm targets at orders of magnitude speedup of the power-optimal dual V_{dd} buffer insertion and buffered tree construction problems. Our algorithm is based on [4] with some important speedup techniques. Our main contributions include

1. proposing three speedup techniques for power-optimal dual V_{dd} buffer insertion, which are Pre-buffer Slack Pruning (PSP) extended from the one presented in [11, 12] for dual V_{dd} buffer insertion, Predictive Min-delay Pruning (PMP) and 3D sampling, resulting in a practically linear time algorithm with respect to the tree-size; and

2. introducing grid reduction to further speedup the power-optimal 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 achieve a combined speedup of more than 100x over the exact power-optimal buffer insertion algorithm [2] at the expense of 1% delay and 2% power optimality loss. We also expand the power-optimal routing capability to handle 10-sink nets, which cannot be handle by the D-Tree algorithm in [4], in about one hour.

The rest of the paper is organized as follows. Section 2 presents modeling and problem formulations for DVB and D-Tree. Our speedup techniques for DVB and D-Tree are described in details in Section 3. Section 4 proposes our Fast dBIS and Fast dTree algorithms for DVB and D-Tree problem, respectively. Experimental results and some discussions are given in Section 5, and the paper is concluded in Section 6.

2. PRELIMINARIES

2.1 Delay, power, and slew model

We use distributed Elmore delay model as in [7, 8, 9, 10]. The delay $d(l)$ due to a piece of wire of 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 resistance of the interconnect 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 [4]. 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 always within up to only a few tens ps of the upper bound. Therefore we model buffer delay with negligible error by approximating input slew rate using the upper bound as in [4].

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 collapse per switch short-circuit and dynamic power consumed by a buffer into a single value E_{buf} , which is a function of both V_{dd} and buffer size.

2.2 Dual Vdd Technique

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 saving without worsening the delay of the overall interconnect tree.

As in [4], we have the following constraint of dual V_{dd} buffers – we only allow high-Vdd 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 only placed at high V_{dd} sinks driven by low V_{dd} buffers.

2.3 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 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 n.

Dual-Vdd buffer insertion (DVB) – 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 DVB 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 $\bar{s}$.

2.4 Dual Vdd buffered tree construction

We measure the delay and power performance using the same metric as in the DVB 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 station (BS) which are the allocated space for buffer insertion. Therefore we have the following problem formulation.

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

3. SPEEDUP TECHNIQUES

3.1 DVB Problem

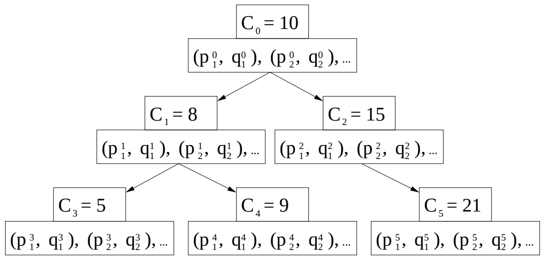
Our power-optimal buffer insertion algorithm is based on [4] with speedup techniques to improve runtime. Power-optimal solutions are constructed using partial solutions (i.e. options) from the subtrees. At each node of the given routing tree, a list of options for the sub-tree rooted at that node is generated by recursively traversing the tree in a bottom up fashion. In DVB 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 power-optimal buffer insertion problem. [2] shows that the growth of the number of options is psuedo-polynomial. The option sampling technique in [4] bounds the growth of options at each node, which help reduce the number of options at the expense of optimality. In the following we first discuss the data structure and then present techniques for effective option pruning.

3.1.1 Data structure

Advanced data-structures in [11] 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 algortihm to-date for power-optimal buffer insertion [4] makes use of augmented orthogonal search trees embedded in a balanced binary search tree (BST), based on which we initially develop our algorithm. In order to maintain a non-redundant set of options at each node n, we maintain a balance binary search tree BST_n sorted by downstream capacitance of the options, as shown in Figure 1. Each node $opList_c$ in BST_n is a set of (rat_n, pwr_n) pairs.



C0 = 10
(p01, q01), (p02, q02), ...
C1 = 8
(p11, q11), (p12, q12), ...
C2 = 15
(p21, q21), (p22, q22), ...
C3 = 5
(p31, q31), (p32, q32), ...
C4 = 9
(p41, q41), (p42, q42), ...
C5 = 21
(p51, q51), (p52, q52), ...

Figure 1: Data structure of DVB problem

We analyze this data-structure for buffer insertion algorithms by analyzing the statistics of options. The number of distinct values of capacitance is generally smaller than that of power due to the presence of slew rate bound. Figure 2 shows the statistics for the number of distinct values of capacitance and power in each tree node for net s4, which is a 99-sink net with 137 buffer candidate nodes. We can see that the number of distinct values of power is larger than that of capacitance in almost all tree nodes. Therefore, labeling the BST_n with capacitance results in the least number of tree nodes. Table 1 shows the runtime of five test cases calculated by DVB in [4] with power indexed and capacitance indexed data structure, respectively. We can see that the power indexed data structure (column "p-indexed") is much slower than the capacitance indexed counterpart (column "c-indexed"). Therefore, we use a capacitance-indexed binary search tree for the best runtime.

node#	sink#	p-indexed (s)	c-indexed (s)
86	19	36	15
102	29	28	21
142	49	142	40
226	99	1905	104
515	299	>3600	373

Table 1: Comparison of runtime between power-indexed and capacitance-indexed data structures

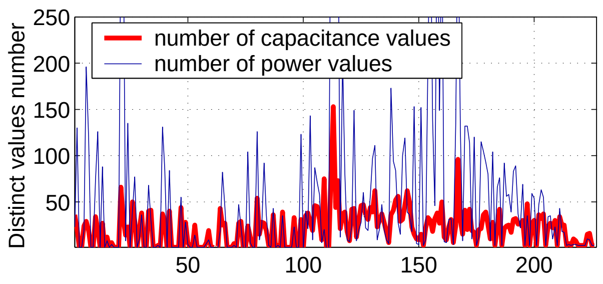


Figure 2: The number of distinct values of capacitance and power in each node

We also find that, 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 $opList_c$ ($> 80\%$) contains less than 10 options. Using sophisticated orthogonal search trees like those in [2, 4] only speeds up a very small portion of all $opList_c$ operations while significantly increases the runtime overhead of other cases. Therefore, we only 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.

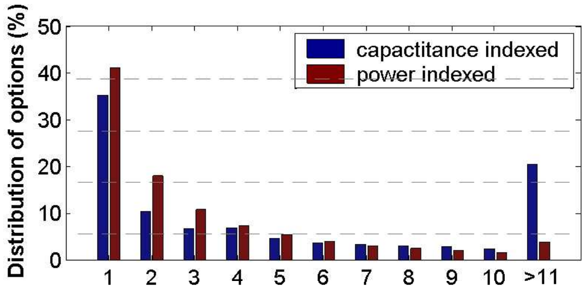


Figure 3: The distribution of options indexed by power and capacitance

3.1.2 *Pre-buffer slack pruning*

The aggressive pre-buffer slack pruning (APSP) in [13, 12] prune redundancy 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 help 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)). [12] 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 DVB problem is much larger than that of the min-cost problem, we expect more speedup from using PSP.

3.1.3 *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 continuous number of buffers and buffer sizes. If such delay does not meet the delay specification at the source, the option is dropped to save the algorithm from 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

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

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

$$delay_{opt} = 2\sqrt{r_s c_o r c}(1 + \sqrt{\frac{1}{2}(1 + \frac{c_p}{c_o})}) \tag{6}$$

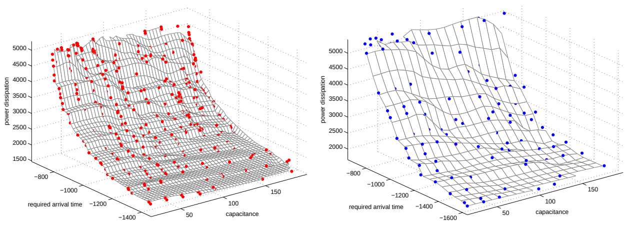
We pre-compute a unit length minimum delay table indexed by buffer, unit length resistance and capacitance, and the path length from the source to each tree node. We assume high V_{dd} buffers to calculate the unit length minimal delay, such that we get a lower bound when both high V_{dd} and low V_{dd} 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. A newly generated option $\Phi = (rat, cap, pwr, \theta)$ is pruned if $rat - dlb(v) < RAT_0$.

We arrive at some interesting observation about PMP through extensive experimentation. 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 [14]. 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 needed. 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 an 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 p-PMP rule takes 2x



(a) Before sampling (b) After sampling

Figure 4: 3D sampling for non-redundant options

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.

3.1.4 3D sampling pruning

2D sampling in [4] picks a fixed number of options in each $opList_c$ in Figure 1 under each 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 are uniform and the slew rate bound is tight. However, we observe that this number is not that small for large testcases. Table 2 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% nodes carry over 50 distinct capacitive values and over 10% nodes carry more than 100 capacitive values. When the scale of the test case becomes larger, tree nodes carry larger number of distinct capacitive values. Therefore, we need to explore more effective sampling rules by taking capacitance into consideration.

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

Table 2: The percentage of the nodes carrying a large number of distinct capactive values

We extend the power-delay sampling [4] to 3D sampling, during which we get option samples based on all 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) a pre-sample and (b) an after-sample option sets. Each dot corresponds to an option. We divide each side 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).

Compared to the 2D power-delay sampling in [4], 3D sampling can prune over 75% more options, and achieve about 5 times speedup, while keeping the solution quality (delay and power) within 1% from the optimal under a sampling grid of 20x20x20. Therefore, we control the number of options in each tree node by 3D sampling. That is, given the number of sampling grid on one side 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 DVB problem by using 3D sampling.

3.2 D-Tree problem

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. 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 insertion 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 $\mathcal{S}$ and a reachability set $\mathcal{R}$. An option for D-Tree is denoted as $\Phi = (\mathcal{S}, \mathcal{R}, rat, cap, pwr, \theta)$, and we re-define 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 set. 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 to see if any options in the full sink set (i.e. the sink set which includes all sinks) domainates Φ, and if Φ dominates any option under its own sink set $\mathcal{S}$.

In addition to the speedup techniques presented in Section 3.1, we also apply the following heuristic to further narrow the search space.

3.2.1 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 3.1.3, 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 which 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 $j = 1, 2, \cdots$. Figure 5 shows an example of (b) a reduced grid from (a) a full escape grid.

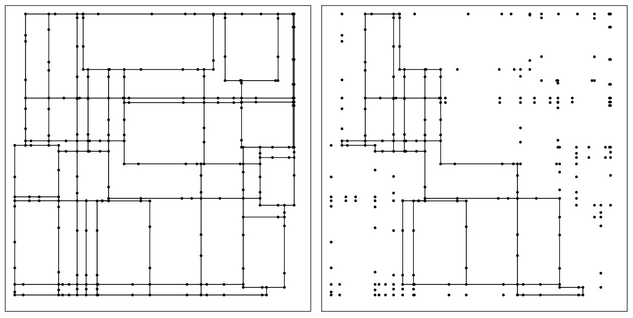

(a) Original grid (b) After reduction

Figure 5: Escape grid reduction for grid.5

4. ALGORITHMS AND ANALYSIS

4.1 Fast buffer insertion algorithm for DVB 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 nodes n and 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 3) 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).

4.2 Fast buffered tree construction algorithm for D-Tree problem (Fast dTree)

We apply our grid erduction 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 needed 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 impelentation, dominated options under our pruning rules (line 19–20) or filtered options by 3D sampling (line 12) are freed to save memory.

```
Algorithm 1: Fast-dBIS ($T_n$)
1:  $Set(\Phi_n) = (c_n^s, 0, q_n^s, false)$ if $n$ is a sink, else (0,0,$\infty$,false)
2:  for each child $v$ of $n$
3:     $Set(\Phi_v)$ = 3DSampling Fast-dBIS($v$)
4:     $Set(\Phi_{temp}) = Set(\Phi_n)$
5:     $set(\Phi_n) = \phi$
6:     for each $\Phi_i \in Set(\Phi_v)$
7:        for each $\Phi_t \in Set(\Phi_{temp})$
8:           for each buffer $b_k \in Set(B)$
9:              if $b_k = \phi$
10:                $V_n = V_H$ if $\theta_i$ or $\theta_t$ is true, else $V_L$
11:                $\Phi_{new}$=$(c_i + c_t, p_i + p_t + E_{n,v},$
                       $\min(q_t, q_i - d_{n,v}), \theta_i$ or $\theta_t)$
12:             else if i. $V_b^k$ is high; or ii. $V_b^k$ is low and $\theta_i$ is false
13:                $\Phi_{new}$=$(c_i + c_t, p_i + p_t + E_{n,v}(V_b^k) + E_b^k,$
                       $\min(q_t, q_i - d_{n,v} - d_b^k(c_i + c_{n,v})),$
                       $\theta_t or(if V_b^k = V_H))$
14:             else goto line 8
15:             if i. slew rate violation at downstream buffers; or
                 ii. $\Phi_{new}$ is redundant (by PMP); or
                 iii. $\Phi_{new}$ is dominated by any $\Phi_z \in Set(\Phi_n)$ (by PSP)
16:                drop $\Phi_{new}$
17:             else
18:                remove all $\Phi_z \in Set(\Phi_n)$ dominated by $\Phi_{new}$
19:                $Set(\Phi_n) = Set(\Phi_n) \cup \Phi_{new}$
25: Return $Set(\Phi_n)$
```

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

4.3 Analysis of Fast dBIS and Fast dTree algorithms

As we have mentioned in the previous section, the key to runtime reduction is to reduce propagated options in the algorithm. In Section 3.1.4, 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 6 shows the number of non-redundant options generated by DVB, PSP+DVB, PSP+PMP+DVB, 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. The PSP and PMP pruning rules also help reduce the number of solutions, but it does not guarantee a controlled growth of options. 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 linearly with respect to the tree size. Figure 7 shows the runtime growing trend with respect to the number of node, and it is clear that Fast dBIS has a roughly linear runtime complexity.

Applying all proposed pruning techniques and the grid reduction heuristic in the Fast dTree algorithm helps significantly reduce the number of options during runtime. This is evident from the fact that the Fast dTree algorithm can handle up to 10-sink net as opposed to only a few sink net in [4], which is to be demonstrated in Section 5. However, 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, although routing nets of up to 10-sink sufficiently covers most of the global net instances in practical microprocessor designs.

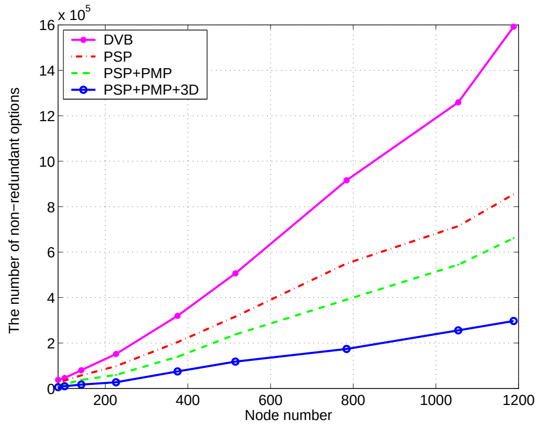


Figure 6: The option increase trends under different pruning strategies

5. EXPERIMENTAL RESULTS

5.1 Experimental results of Fast dBIS

We test our algorithm on 9 test cases s1–s9 generated by randomly placing source and sink pins in a 1cm x 1cm box. We use GeoSteiner package [15] to generate the topologies of the test cases. We also break interconnect between nodes longer than $500\mu m$ by inserting degree-2 nodes. In this experiment we assume that every non-terminal nodes are candidate buffer 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 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 which 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. In PSP, we use the 16x buffer (high V_{dd} and low V_{dd}) to perform pruning. In PMP, we calculate the unit-

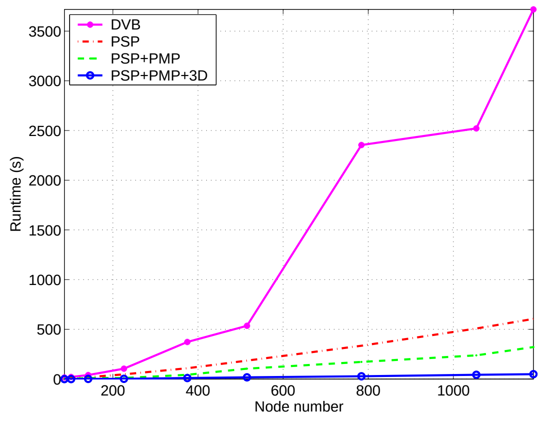


Figure 7: The runtime trends under different pruning strategies.

length-min-delay with the settings under 65nm technology node [16]. We set the 3D sampling grid to $20 \times 20 \times 20$, which we have found to give good accuracy-runtime trade-off. The experiments are performed on Linux with Intel PM 1.4Ghz CPU and 1Gb memory.

To illustrate the effectiveness of our three speedup techniques, we perform tests by adding them into the baseline DVB algorithm [4] incrementally, and collect the runtime in Table 3. Compared with DVB, PSP achieves 5x speedup (column "PSP"), PSP+PMP achieves another 2x speedup (column "SM"), and 3D sampling (column "3D") achieves an additional 5x speedup. We observe that PMP is a relatively weak pruning rule by itself, but it achieves a speedup of 2x when used on top of PSP. 3D sampling is orthogonal to other rules in terms of pruning capability. Moreover, applying 3D sampling with PSP and PMP yields better buffer insertion solution in both delay and power than using 3D sampling alone. This is due to the fact that PSP and PMP leaves a pool of options which are more densely populated with more superior delay and power, from which 3D sampling is able to pick good options. In the case without PSP and PMP, the option pool from which 3D sampling picks is littered with options that will be dominated subsequently by other better options with respect to the predictive pruning rules, which are lost in the process of 3D sampling.

nets	DVB	PSP	SM	Fast dBIS
s1	15	8	1	0
s2	21	11	3	0
s3	40	19	10	1
s4	104	47	10	2
s5	373	109	43	10
s6	536	185	104	17
s7	2354	335	172	28
s8	2521	508	238	43
s9	3719	605	322	49
x	1	5x	10x	50x

Table 3: Runtime of different combination of our speedup techniques for buffer insertion

To show the performance of Fast dBIS, we compare three algorithms, which are (i). power-optimal buffer insertion (PB) algorithm [2] considering only single (high) V_{dd} buffers, (ii). SVB/DVB [4], which correspond to the single V_{dd} and dual V_{dd} buffer insertion algorithms in [4] with 2D sampling grid set to 20×20, and (iii). our Fast sBIS/dBIS, which correspond to our single and dual V_{dd} buffer insertion algorithms. Table 4 shows experimental results of these three algorithms. We find that Fast sBIS/dBIS is on average over 50x faster than SVB/DVB. We compare the solution quality of our Fast sBIS (in terms of power dissipation and RAT at source) with SVB. From Table 4, we find that the RAT at source of Fast sBIS are only than 1% smaller than those of SVB, and the power values of Fast sBIS are 1% larger than those of SVB. [4] reports that their algorithm incurs no loss of optimality in RAT and 1% larger power than the optimal, therefore our algorithm is 1% and 2% away from the optimal delay and power, respectively. In some cases, Fast sBIS gives even smaller power values than SVB, as PSP and PMP improves the quality of options after sampling.

Table 6 shows the experimental results of a group of extremely large test cases for dual V_{dd} buffer insertion, which all other power-optimal buffer insertion methodologies [4, 2] fail to handle. Note that we need only less than six minutes to find a min-power dual V_{dd} buffer insertion solution for a 3k-sink net with 23k buffer candidate nodes.

net	nodes#	sinks#	Fast sBIS(s)	Fast dBIS(s)
r1	2895	267	13	35
r2	5799	598	27	81
r3	7602	862	37	162
r4	15512	1903	78	239
r5	23499	3101	119	352

Table 6: Runtime for buffer insertion of large test cases

5.2 Experimental results of Fast dTree

For buffered tree construction, we create 6 test cases by randomly generating source and sink pins in a 1cm × 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 5.1.

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/dTree (column "sTree/dTree") runs over 100x Faster than S-Tree/D-Tree with solutions 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 last 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.

6. CONCLUSIONS

We have presented efficient algorithms to tackle the power optimal buffer insertion and buffered routing tree construc-

test cases			runtime(s)					RAT*(ps)		power(fJ)			
net	nodes#	sinks#	PB	SVB	sBIS	DVB	dBIS	SVB	sBIS	SVB	sBIS	DVB	dBIS
s1	86	19	14	4	0	15	0	-712	-716	5930	5841	4925	4763
s2	102	29	17	7	0	21	0	-811	-819	6827	6283	5671	5535
s3	142	49	39	15	0	40	1	-1119	-1127	10109	9565	8901	8213
s4	226	99	460	60	1	104	2	-963	-965	14043	14042	11671	11502
s5	375	199	2461	149	3	373	10	-1848	-1857	20045	19591	14813	14940
s6	515	299	3744	227	6	536	17	-1742	-1749	25241	25417	18328	18214
s7	784	499	-	448	11	2354	28	-3023	-3041	36339	35436	26065	25327
s8	1054	699	-	1033	15	2521	43	-2484	-2486	41033	41483	27316	27667
s9	1188	799	-	1946	23	3719	49	-2395	-2405	43011	42654	28432	28766
				1	< 1/50	1	< 1/50	1	> 99%	1	< 101%	1	< 101%

Table 4: Comparison of runtime and performance for buffer insertion (PB vs. DVB vs. Fast dBIS)

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

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

tion problems using dual V_{dd} buffers. We show that the sophisticated data-structures which has good amortized complexity do not necessarily benefit the runtime, and that the key to runtime reduction is to reduce propagated options. We present three speedup techniques, namely prebuffer slack pruning, predictive min-delay pruning, and 3D sampling, and obtain an effectively linear time algorithm at 1% and 2% of delay and power optimality loss, respectively. In addition to these, we enhance the power-optimal buffered tree construction by introducing routing grid reduction. Experimental results show that we obtain over 50x and 100x speedup compared to the most efficient existing algorithms for dual V_{dd} buffer insertion and buffered tree construction respectively in the literature to-date.

7. REFERENCES

[1] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," in *Proc. IEEE Int. Symp. on Circuits and Systems*, 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] W. Shi and Z. Li, "An o(nlogn) time algorithm for optimal buffer insertion," in *DAC*, Jun 2003.

[12] 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.

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

[14] 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.

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

[16] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors*, 2003.