

Path-Based Buffer Insertion

C. N. Sze, Charles J. Alpert, Jiang Hu, and Weiping Shi

Abstract—Along with the progress of very-large-scale-integration technology, buffer insertion plays an increasingly critical role on affecting circuit design and performance. Traditional buffer insertion algorithms are mostly net based and therefore often result in suboptimal delay or unnecessary buffer expense due to the lack of global view. In this paper, we propose a novel path-based-buffer-insertion (PBBI) scheme which can overcome the weakness of the net-based approaches. We also discuss some potential difficulties of the PBBI approach and propose solutions to them. A fast estimation on buffered delay is employed to improve the solution quality. Gate sizing is also considered at the same time. Experimental results show that our method can efficiently reduce buffer/gate cost significantly (by 71% on average) when compared to traditional net-based approaches. To the best of our knowledge, this is the first work on path based buffer insertion and simultaneous gate sizing.

Index Terms—Integrated circuit layout, performance optimization, physical design, repeaters, timing optimization, very large scale integration (VLSI).

I. INTRODUCTION

Buffer insertion is widely recognized as an essential technique for interconnect optimization [1] while interconnect is a fundamental limit [2] for very-large-scale-integration (VLSI) technology progress. The importance of buffer insertion has resulted in numerous algorithmic and methodological works. Perhaps the most influential work is the classic van Ginneken's algorithm [3]. Given a Steiner tree spanning a signal net and candidate buffer locations on the tree, van Ginneken's dynamic-programming (VGDP) algorithm can find the maximum timing slack solution optimally in quadratic time. This algorithm is extended to handle buffer cost and buffer library in [4]. The noise avoidance issue is addressed in buffer insertion in [5]. Higher order delay models are adopted in buffer insertion in [6]. For two-pin nets, quadratic programming-based approach [7] and closed form buffering solutions are proposed in [8] and [9]. Recently, an $O(n \log^2 n)$ buffer insertion algorithm was developed [10].

Recently, an industry study [11] predicted that 35% of the cells on a chip will be buffers at 65 nm. The huge number of buffers may affect various aspects of circuit design and performance including timing [1], power dissipation [4], signal integrity [5], placement, and routing congestion [11]. Therefore, buffer insertion needs to be conducted in a more elaborate manner to push the envelope of performance.

In fact, most of the previous works on buffer insertion are net based, i.e., buffer insertion is performed on one net after another individually. In [12], it is proposed to apply VGDP for each net of the circuit in the order of criticality. However, the algorithm suffers from long execution time and the same author proposed several techniques

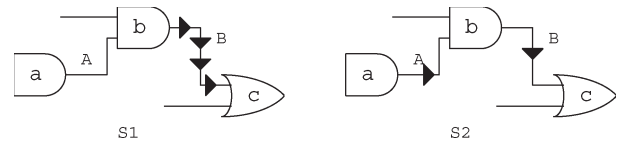


Fig. 1. Net-based buffer insertion solutions depend on net ordering.

in [13]. Even though the buffer insertion problem with net-based formulations is relatively easy to solve, it may lead to suboptimal path delay or unnecessary buffer usage due to the lack of global view. The weakness of net-based buffer insertion can be illustrated through a very simple example in Fig. 1. Consider two nets A and B along a critical path in the circuit. If we perform buffer insertion on net B first, four buffers are needed on B and then no buffer is needed on A for satisfying the critical path timing constraint. This is denoted by solution $S1$. However, the constraint can also be satisfied by putting one buffer on both A and B , denoted as $S2$. Optimizing the entire path may get a better solution and certainly can lead to a better deployment of buffering resources. Usually, net-based dynamic-programming algorithms such as [4] do not have a global view, so nets which are processed first tend to overconsume buffer resources.

Despite their current popularity, the net-based buffer insertion methods, without global view of the whole combinational circuit, will become inadequate for future technologies. In [14] and [15], network-based buffer insertion algorithms are proposed. In these approaches, buffer insertion is performed on all nets between primary input (PI)/registers and registers/primary output (PO) simultaneously through Lagrangian relaxation. However, both works include a restrictive assumption that buffers are inserted at every branch node to simplify the calculation of delay. In practice, whether or not buffers are necessary at a certain branch node depends on timing constraints of related paths. Due to path reconvergence, it is very difficult to perform network-based buffer insertion without any assumption. Although the network-based algorithms usually produce good results, they do not scale very well. Indeed, in [14], the CPU time consumption explodes for the larger testcases. Based on the same problem formulation, [16] proposed the min-cost flow and min-cut techniques, which are shown to improve both the solution quality and efficiency. When buffer insertion is considered beyond the limit of nets or gates, it is natural to consider gate sizing at the same time. In [17], a greedy heuristic on integrated gate sizing and buffer insertion is proposed. However, it neglects wire delays. The network-based methods make severe oversimplifications in order to achieve a solution, such as ignoring wire delays entirely. Our method takes advantage of some global optimization without sacrificing the modeling accuracy that the net-based approaches provide.

In this paper, we propose a path-based-buffer-insertion (PBBI) heuristic in order to minimize buffer/gate cost subject to path timing constraints. This approach is in the middle between net-based and network-based methods. However, it can achieve both better solution quality than the net-based method and faster computation runtime than the network-based scheme. Since our path-based approach can easily handle false paths, the solution quality can also be much better than network-based methods. Besides, instead of relying solely on static timing analysis (STA), a fast estimation on buffered delay is applied on the entire network so that a better global view is obtained. Our PBBI algorithm is based on the VGDP algorithm since it is robust and sophisticated enough to handle different instances. However, directly using VGDP may induce problems and we successfully solve those problems by a set of techniques such as off-path required-arrival-time

Manuscript received May 10, 2005; revised November 22, 2005, February 8, 2006, and June 27, 2006. This work was supported in part by the Semiconductor Research Corporation under Contract 2003-TJ-1124, the Design Automation Conference (DAC) Graduate Scholarship, and by the National Science Foundation (NSF) under Grants CCR-0098329, CCR-0113668, and EIA-0223785. This paper was recommended by Associate Editor J. Lillis.

C. N. Sze and C. J. Alpert are with the IBM Austin Research Laboratory, Austin, TX 78758 USA (e-mail: csze@us.ibm.com).

J. Hu and W. Shi are with Texas A&M University, College Station, TX 77843 USA.

Digital Object Identifier 10.1109/TCAD.2006.888281

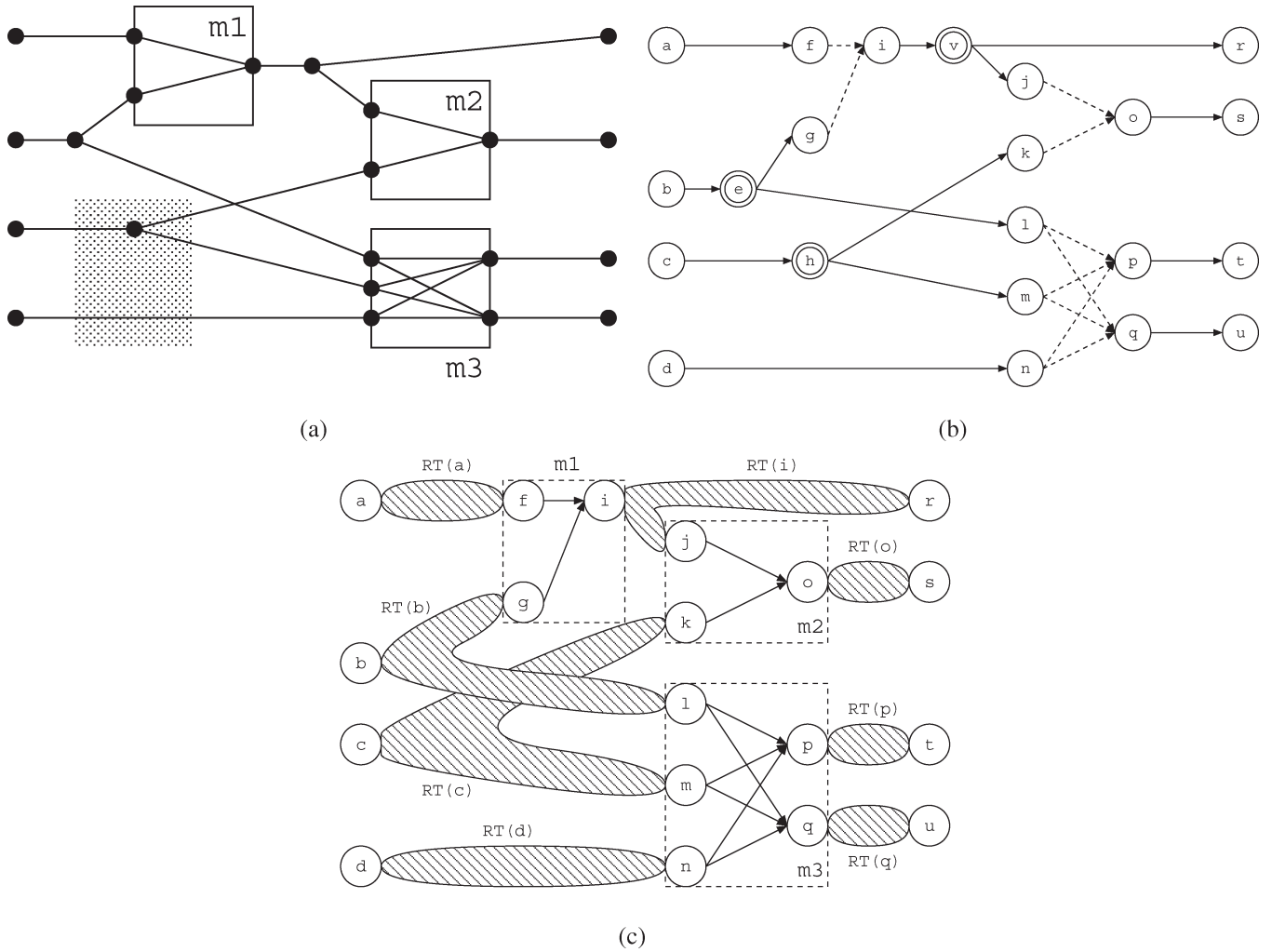


Fig. 2. Example of combinational circuit models. (a) An example of a combination circuit. (b) The corresponding DAG. (c) The circuit model with routing tree abstraction.

(RAT) estimation and gate sizing at sinks. Experimental results show that the usage in buffer/gate cost is reduced by 71% on average through our approach compared with traditional net-based algorithms. The runtime is also reasonably fast.

The conference version of our paper can be found in [18]. However, this paper contains more details in algorithm description and new experimental results. Moreover, experimental results on International Symposium on Circuits and Systems (ISCAS) benchmark circuits are also included.

II. PROBLEM FORMULATIONS

A placed and routed circuit can be formulated as a directed acyclic graph (DAG) $G = (V, E)$. An example is shown in Fig. 2. A vertex $v \in V$ can be either: 1) a PI/PO [e.g., nodes a and r in Fig. 2(b)]; 2) a pin of a module (e.g., f and i); 3) a Steiner node on the route [e.g., e , shown as double circle in Fig. 2(b)]; or 4) a candidate buffer insertion location [not shown in Fig. 2(b)]. An edge is either an interconnect wire (solid line) or an input-to-output path (dotted line) within a module.

In this paper, interconnect wires are modeled as a distributed RC network and we adopt the Elmore delay model [19]. Thus, an interconnect w is annotated by the corresponding resistance $R(w)$ and capacitance $C(w)$. A buffer b_i in the buffer library is defined by its

load capacitance $C(b_i)$, intrinsic delay $T(b_i)$, and output resistance $R(b_i)$, while $W(b_i)$ represents the buffer cost which can be either buffer area or power.

A module is identified by the delay $T_{j,k}$ for each input-to-output path, the capacitance C_j of the input pin, and the resistance R_k of the output pin. If x is the size of the module, $C_j = \hat{C}_j x + f_j$ and $R_k = \hat{R}_k / x$, where $\hat{R}_k, \hat{C}_j, f_j$ are the unit size output resistance, unit size gate area capacitance, and gate perimeter capacitance of the module. In this paper, the size of each gate is selected from the set $S = \{x_1, x_2, \dots, x_n\}$. Each PI is annotated with a user-defined arrival time (AT) while each PO is annotated with a RAT.

Buffer blockages may exist in the floorplan, for example, the shaded box in Fig. 2(a). When a buffer insertion candidate location overlaps with the region of the buffer blockages, it is restricted such that no buffer can be placed. In other words, there exists no buffer candidate location within the buffer blockage region.

Signal slew rate is also considered in our PBBI algorithm. For a signal propagating along a wire, we employ a simple metric of propagation_slew = $\ln 9 \cdot \text{Elmore_delay}$ [1]. The slew rate at the receiving end of a wire depends on both the propagation slew rate s_p and the launching slew rate s_l at the driving end of the wire and is given in [20] as $\sqrt{s_p^2 + s_l^2}$. In our buffer insertion algorithm, any buffer solution with receiving slew rate greater than a certain threshold will be discarded.

The problem of **simultaneous gate sizing and buffer insertion** is defined as follows. Given a DAG which represents a placed and routed combinational circuit, a buffer library, a set of buffer candidate locations, a set of buffer blockage regions, find a buffering solution such that the overall cost of buffers and gate sizes is minimized. Buffering solution is in terms of the locations and types of buffers inserted and sizes of the gates. At the same time, the solution is subject to the constraint of both the AT at each PI and the RAT at each PO as well as the slew rate requirement.

The problem definition is different from [12] and [13] in the sense that their works are minimizing the circuit delay, subject to local placement congestion constraints. This paper focuses on minimizing the total buffer cost subject to delay constraints. Minimizing the circuit delay sometimes results in overbuffering when the timing constraints are not very tight. Also, strictly obeying the local placement congestion constraints would make it impossible to close timing. For a timing critical net over a locally dense region, it would be better if we insert buffers for delay reduction and then move/resize other cells in that region. In fact, if we model the buffer cost based on local placement congestion, buffer will be placed on locally dense area only when it is critical for timing, therefore, the congestion constrained buffer insertion problem can be solve more smoothly in favor of timing improvement.

As mentioned in Section I, the VGDP approach is very flexible and efficient so that it can be easily applied to buffer blockage avoidance (by selectively setting candidate buffer locations) while considering buffer cost (i.e., area/power), buffer polarity, and slew rate [21]. In order to utilize the flexibility and efficiency of VGDP, we intend to use net-based buffer insertion algorithm as a building block for PBBI. In this way, we abstract the routing tree of the circuit and ignore all the details (i.e., Steiner node and interconnect tree structure, etc.) within the routing tree. An example of our circuit model is shown in Fig. 2(c). In our model, we abstract all interconnect routing so that vertices only represent PI/PO of the circuit and input/output pins of modules (we use this definition for vertex hereafter in this paper) while edges only for input-to-output paths within a module. The routing tree is identified by its root vertex. For example, the routing tree $RT(c)$ is rooted at vertex c and with sinks k and m . In a combinational circuit, the root of a routing tree is either a PI vertex or an output pin of a module, while a sink is either a PO vertex or an input pin of a module.

III. NET-BASED BUFFER INSERTION

For a routing tree, if the RAT at each sink vertex is given, VGDP algorithm traverses every candidate buffer location v_i of the tree in a bottom-up manner, propagating a set of solutions in the form of (c, q, w) which stands for downstream load capacitance, RAT, and total buffer cost, respectively. Each solution reflects the intermediate results of a buffering solution on the subtree rooted at v_i . When the propagation reaches the driver (i.e., root vertex), a set of solution with different cost-RAT tradeoff is obtained. If the AT at the root vertex is given, we pick the solution with minimum buffer cost while the timing requirement is satisfied.

Conventionally, net-based buffer insertion for the whole circuit is accomplished by iteratively performing the following steps.

- 1) STA and obtain the RAT and AT at every vertex.
- 2) Perform buffer insertion for a routing tree with the AT and RAT obtained from STA (according to a specific net order. Discussion of net order is in Section VII).
- 3) Update STA results (of the fan-in/fan-out cone of the buffered routing tree) and perform buffer insertion for the next routing tree.

However, this is in fact a greedy algorithm and in our experiment, we found that the buffering solution of this approach is far from optimal even when we let the iteration run unboundedly in order to refine the buffering solution for buffer cost reduction. Our observations to the problems of net-based buffering include the following.

- 1) STA is usually not buffer aware so the timing estimations at the first iterations are inaccurate and the circuit is very timing critical. Hence, the first processed nets tend to overconsume buffer resources (the case of net B in $S1$ of Fig. 1).
- 2) Since the algorithm is in nature greedy, a poor buffering decision to a routing tree $RT(a)$ due to incorrect timing estimation may lead to a poor buffering decision at other routing trees $RT(b)$ where b is a transitive fan-in/fan-out of a . Thus, earlier buffering decisions may have degraded the quality of the whole buffering process, which cannot be improved in later iterations. This is especially true for those nets along a critical path from PI to PO.
- 3) Buffering solution of a routing tree is highly dependent on the criticalities of the sinks, which in turn depend on buffering of their fan-out routing tree. Therefore, the criticality can be substantially different from the results of STA due to buffer blockages, which brings significant error in final buffering solution.

IV. PBBI

In this section, we propose our PBBI algorithm as described in this section. The key elements of our buffer insertion algorithm are: 1) buffer aware STA; 2) path-based VGDP buffer insertion; and 3) off-path RAT estimation.

Different from net-based buffering insertion schemes, our PBBI algorithm starts with buffer aware STA. The steps are as follows.

- 1) The buffer aware STA takes care of buffer blockages and therefore the resultant AT and RAT is comparatively much more accurate than ordinary STA and it would not overconsume buffer resources even at the very beginning.
- 2) A list of k most critical paths is obtained based on the buffer aware STA.
- 3) Merge all routing trees of the vertices along the path into one big routing tree and then VGDP is applied to the “merged trees.”
- 4) Repeat step 3) for every path found in step 2).

For example, in Fig. 3(a), if the critical path is $\{b, g, i, r\}$, the “merged tree” is formed from $RT(b)$ and $RT(i)$, shown in Fig. 3(b). In our approach, the root (e.g., b) and the sink (e.g., r) along the path have a fixed AT and RAT, respectively, which in turn produces a relatively good buffering solution. For all other sinks, namely off-path sinks (e.g., j, l), we propose an approach to adjust their RAT values to become more accurate and then the adjusted RAT are fed into the VGDP algorithm. In this section, we assume an efficient VGDP algorithm is given such that it considers buffer blockage, buffer polarity, buffer area/power, and slew rate.

A. Buffer Aware STA

The critical path method is widely used as a tool for STA [22]. It propagates the static delay information throughout the circuit. However, the delay along interconnect changes during the buffer insertion process which is a main source of error of net-based buffer insertion algorithms, as mentioned in Section I. A work which predicts the postbuffering delay is [23]. The work derives delay equations along a buffered wire segment considering buffer blockages and applies the equations for delay estimation upon multipin nets. Experimental

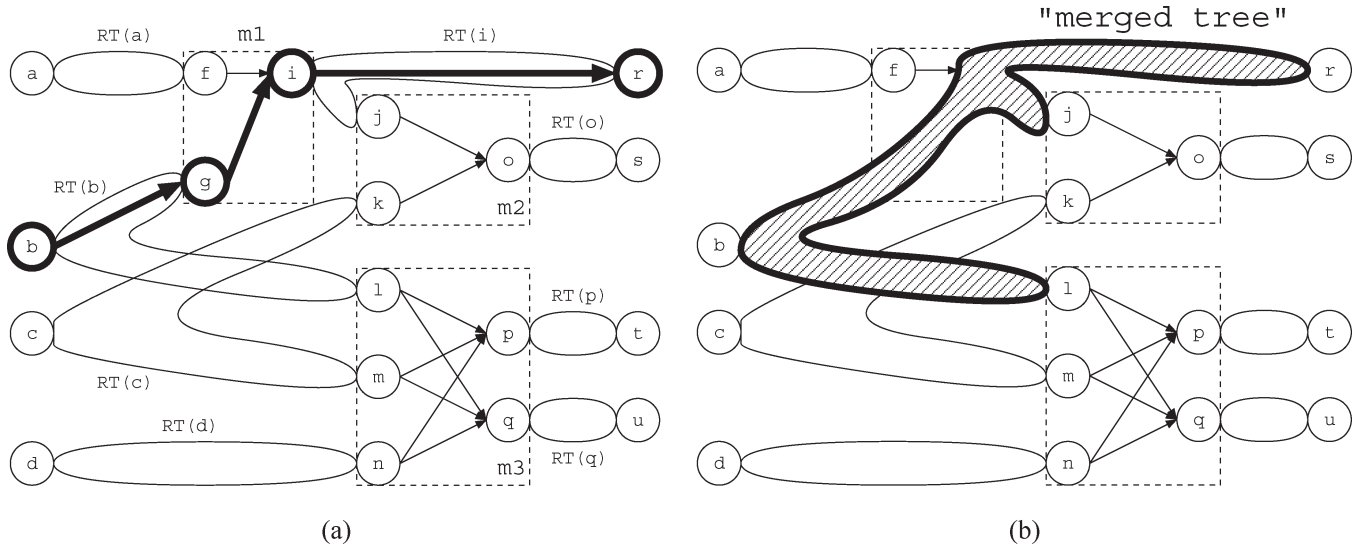


Fig. 3. Example of algorithm overview. (a) Critical path on a combinational circuit. (b) The “merged tree.”

results show that the delay estimation merely produces insignificant errors. Toward our problem, we verify in our experiments that integrating this buffer aware delay estimation with STA provides a good guide for buffer insertion using VGDP. With buffer aware STA, early buffering step will not overconsume buffer resource which trap the overall solution into a local optimum.

Buffer aware STA not only provides a good basis for VGDP algorithm on a path, but it is also a crucial element of the whole PBBI algorithm. For example, if we are performing buffer insertion on a critical path p_c from PI to PO, it propagates a set of solutions from the PO vertex with accurate RAT information (since the RAT at PO is fixed by user specification). During the propagation, some branch paths may exist such that the RAT of those paths is needed to compute the solution sets. In such a way, if the RAT of branch paths is not accurate since STA is not aware of buffering along those branches, VGDP may think that the branches are more critical than p_c , which destroys the solution quality of the PBBI algorithm.

B. Buffer Insertion on “Paths”

In order to accomplish PBBI, a list of distinct paths must be first obtained. With the help of buffer aware STA, k most critical paths can be found using a polynomial-time algorithm in [24] (k can be changed during the progress of the algorithm). The parameter k is a tradeoff between quality and speed while k can be determined on the fly—stop finding the next critical path if the slack of the path is greater than a specific value. Note that the accuracy of our algorithm can also be improved when false paths are detected and only sensitizable critical paths are selected. In the whole circuit, each routing tree has to be processed once. Therefore, if the lists of paths are overlapping with each other, we delete the common vertices from the less-critical path and cut it into different distinct paths. For example, if the three most critical paths in Fig. 2(c) are $\{b, g, i, r\}$; $\{a, f, i, j, o, s\}$; $\{c, k, o, s\}$, after removing common vertices, the list of distinct paths becomes $\{b, g, i, r\}$; $\{a, f\}$; $\{j, o, s\}$; $\{c, k\}$.

After getting a list of distinct paths, for each path, the algorithm treats all routing trees along the path as one big routing tree. The merging process is simple since the routing trees are cascaded together such that the sink (an input pin of a module) along the path merges into the root (an output pin of the same module) of the fan-out routing tree. In Fig. 3(b), the sink pin of RT(b) at g merges with the source

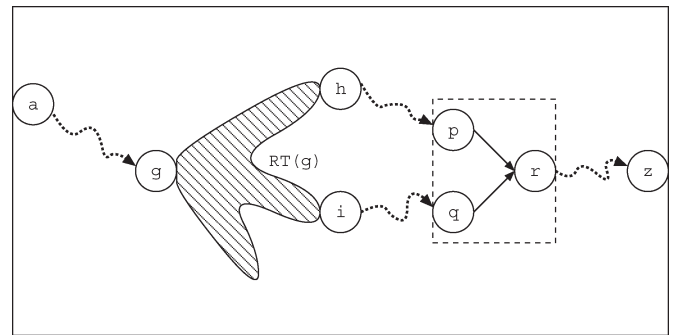


Fig. 4. Problem of PBBI algorithm.

pin at i of RT(i) along the thickened path. The merged vertex (e.g., g together with i) is treated as a candidate buffer location such that a special buffer must be inserted. The parameters of the special buffer correspond to the capacitance/delay/resistance of the pin-to-pin path on the path $\{b, g, i, r\}$ consists of three sinks l, j, r rooted at b . g and i are merged into one buffer location with a special buffer b_s according to input-to-output path of $M1$. (The delay model of a buffer is similar to that of an input-to-output path of a module.) Following the merging process of each path, the VGDP algorithm is applied to the merged routing tree. After all paths have been processed, there may be some routing trees in which no buffer insertion has been performed. They are all comparatively noncritical and net-based buffer insertion can be carried out for each of those nets. In summary, PBBI only processes each of merged trees (which are generated by distinct paths) once. In other words, our algorithm performs buffer insertion on each net only once in one single pass.

C. Off-Path RAT Estimation

Since the PBBI algorithm performs buffer insertion in a path-by-path basis, the buffering solution may violate the timing constraints. An example is shown in Fig. 4. The straight line represents input/output path within a module and dotted curly line stands for a path, which may go along different routing trees and modules. Assume that a is a PI vertex and z is a PO vertex and the algorithm processes the path $p_1 = \{a \rightsquigarrow g \rightsquigarrow h \rightsquigarrow p \rightsquigarrow r \rightsquigarrow z\}$ prior to another path

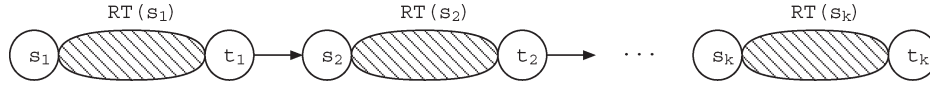


Fig. 5. Example of a circuit for Theorem 1.

$p_2 = \{i \rightsquigarrow q\}$. In a DAG, we define $x \rightarrow y$ as a directed edge from node x to node y while $x \rightsquigarrow y$ as a path of directed edges from x to y which may pass through other nodes. We also define the off-path sinks of a path p_1 as the sinks of the merged routing tree derived from p_1 such that the sinks are not along p_1 . For example, for the path $p_1 = \{a \rightsquigarrow g \rightarrow h \rightsquigarrow p \rightarrow r \rightsquigarrow z\}$, i is an off-path sink of p_1 . When applying VGDP on the path p_1 , actual buffering solution may reduce the delay along $g \rightarrow h \rightsquigarrow p$ to a value which is less than our delay estimation using buffer aware STA. Since the delay between a and z is bounded above by the user specified RAT and AT, the delay along the paths $a \rightsquigarrow g$ and $r \rightsquigarrow z$ could be larger than our estimation. In such a case, it could happen that even with the minimum-delay buffering along the path $i \rightsquigarrow q$, the delay along $p_3 = \{a \rightsquigarrow g \rightarrow i \rightsquigarrow q \rightarrow r \rightsquigarrow z\}$ still violates the timing constraint. After each time we perform path-based buffering insertion along a path, the AT and RAT of fan-in and fan-out cone for each vertex of the path are updated, so the only situation which causes the problem is that there exists reconvergence along the path we are performing buffer insertion. As in the previous example, we are inserting a buffer along the path p_1 , assuming that there exists a path $i \rightsquigarrow q$ in the list of distinct paths. The RAT of the off-path sink i (denoted as RAT_i) does not reflect the buffering solution along $r \rightsquigarrow z$ since the final buffering solution is unknown until the whole path p_1 is done. In this consideration, an equation for spreading out the slack of a path to all the routing tree is needed, which is presented in Theorem 1. In the following, we first derive the equation and revisit the problem in the example at the end of this section.

Consideration of a circuit which only contains a cascade of two-pin routing trees $\{\text{RT}(s_1), \text{RT}(s_2), \dots, \text{RT}(s_k)\}$ is shown in Fig. 5. For each routing tree $\{\text{RT}(s_i)\}$ with root s_i , the only sink is t_i . With optimal buffering, we can find the minimum delay d_i for each $\text{RT}(s_i)$. Based on the minimum delay, for each s_i , AT_{s_i} and RAT_{s_i} are propagated from PI and PO, respectively, according to the user specified timing constraint. Note that since the circuit is a sequence of two-pin nets, the slack $\text{RAT}_{s_i} - \text{AT}_{s_i}$ must be the same for all s_i . If $\text{AT}_{s_i} = \text{RAT}_{s_i}$, the circuit has zero slack and only the minimum-delay buffering solution can fulfill the timing constraint. However, if slack > 0 , different buffering with smaller buffer cost is possible. We can denote $\text{RAT}_{s_i} - \text{AT}_{s_i}$ as “useful slack” resource such that buffering algorithm uses it to reduce the total buffer cost. Intuitively, since the ratio of timing improvement to buffer cost is usually decreasing when the buffer cost increase along the cost-delay tradeoff curve, a buffering solution for the whole circuit with minimum cost tends to spread the “useful slack” to each $\text{RT}(s_i)$, which is to maximize the ratio of timing improvement to buffer cost for each $\text{RT}(s_i)$. Based on a buffering solution B with minimum cost, we can calculate the delay d_i^B , $\text{AT}_{s_i}^B$ and $\text{RAT}_{s_i}^B$ for each $\text{RT}(s_i)$ accordingly. Note that $\text{AT}_{s_1}^B = \text{AT}_{s_1}$ and $\text{RAT}_{t_k}^B = \text{RAT}_{t_k}$. Ideally, $\text{AT}_{s_i}^B = \text{RAT}_{s_i}^B$ since the solution B would consume “useful slack” completely.

The following theorem is to quantify the spreading of “useful slack” and it matches with our experimental results of buffering for minimum cost. The minimum delay to PO is $\text{RAT}_{t_k}^B - \text{RAT}_{s_i}^B$ with buffering solution B and that is $\text{RAT}_{t_k} - \text{RAT}_{s_i}$ with optimal buffering solution for minimum delay.

Theorem 1: For a path of k two-pin routing trees $\text{RT}(s_i)$ for $i = 1, \dots, k$, if the “useful slack” evenly distributes to every routing tree in a manner that the ratio of minimum delay to PO with buffering solution

TABLE I
SUMMARY OF TEST CASES

circuit	Circuit Size			
	# mod	# edge	total	# B candidate
a1	53	68	121	1449
a2	154	160	314	1880
a3	259	272	531	3190
a4	328	352	680	3897
a5	465	480	945	4316
a6	564	592	1156	6625
a7	742	768	1510	6810
a8	766	816	1582	8658
a9	893	928	1821	8083
a10	999	1072	2071	11623
a11	1958	2136	4094	20432
a12	2983	3120	6103	25738

B to that with optimal buffering solution is a constant among all s_i , then

$$\text{RAT}_{s_i}^B = \text{RAT}_{s_i} - \frac{(\text{RAT}_{t_k} - \text{RAT}_{s_i})(\text{RAT}_{t_k} - \text{AT}_{t_k})}{\text{AT}_{t_k} - \text{AT}_{s_1}}. \quad (1)$$

Proof: From the assumption that the delay ratio is a constant, we have

$$\frac{\text{RAT}_{t_k}^B - \text{RAT}_{s_i}^B}{\text{RAT}_{t_k} - \text{RAT}_{s_i}} = \frac{\text{RAT}_{t_k}^B - \text{RAT}_{s_1}^B}{\text{RAT}_{t_k} - \text{RAT}_{s_1}} \quad (2)$$

$$\frac{\text{RAT}_{s_i} - \text{RAT}_{s_i}^B}{\text{RAT}_{t_k} - \text{RAT}_{s_i}} = \frac{\text{RAT}_{s_1} - \text{RAT}_{s_1}^B}{\text{RAT}_{t_k} - \text{RAT}_{s_1}} \quad (3)$$

$$\text{RAT}_{s_i} - \text{RAT}_{s_i}^B = \frac{(\text{RAT}_{t_k} - \text{RAT}_{s_i})(\text{RAT}_{s_1} - \text{AT}_{s_1})}{\text{RAT}_{t_k} - \text{RAT}_{s_1}} \quad (4)$$

$$\text{RAT}_{s_i} - \text{RAT}_{s_i}^B = \frac{(\text{RAT}_{t_k} - \text{RAT}_{s_i})(\text{RAT}_{t_k} - \text{AT}_{t_k})}{\text{RAT}_{t_k} - \text{RAT}_{s_1}} \quad (5)$$

$$\text{RAT}_{s_i} - \text{RAT}_{s_i}^B = \frac{(\text{RAT}_{t_k} - \text{RAT}_{s_i})(\text{RAT}_{t_k} - \text{AT}_{t_k})}{\text{AT}_{t_k} - \text{AT}_{s_1}}. \quad (6)$$

Equations (2)–(6) show the derivation of Theorem 1. From (2) to (3), we substitute $\text{RAT}_{t_k}^B$ with RAT_{t_k} and subtract 1 from both side. Equation (4) is based on the fact that $\text{RAT}_{s_1}^B = \text{AT}_{s_1}$ with zero slack and (5) is due to equal slack along the two-pin routing trees. Finally, we have (6) because $\text{RAT}_{t_k} - \text{RAT}_{s_1} = \text{AT}_{t_k} - \text{AT}_{s_1}$ which is delay from s_1 to t_k . ■

Theorem 1 provides a method for adjusting the RAT of i in Fig. 4 and the equation is shown in (7), where RAT_z is the RAT at z and AT_z is the AT at z based on buffer aware STA. Although the slack $\text{RAT}_z - \text{AT}_z$ is due to the path $\{a \rightsquigarrow g \rightarrow h \rightsquigarrow p \rightarrow r \rightsquigarrow z\}$ which is less than the slack at i , we can use $\text{RAT}_z - \text{AT}_z$ as a lower bound estimation. In such case, spreading the value of $\text{RAT}_z - \text{AT}_z$ over path $a \rightsquigarrow g \rightarrow i \rightsquigarrow q \rightarrow r \rightsquigarrow z$ gives a good adjustment to RAT_i and makes the sink i a little bit more critical in the routing tree $\text{RT}(g)$

$$\text{adjusted RAT}_i = \text{RAT}_i - \frac{(\text{RAT}_z - \text{RAT}_i)(\text{RAT}_z - \text{AT}_z)}{\text{AT}_z - \text{AT}_a}. \quad (7)$$

For example in Fig. 4, if the delay along the most critical path $\{a \rightsquigarrow g \rightarrow h \rightsquigarrow p \rightarrow r \rightsquigarrow z\}$ is 100 units and $\text{RAT}_z = 120$, the worst

TABLE II
COMPARISON OF NET-BASED AND PBBI

circuit	Net based (min delay)		Net based (min cost)		Path based (PBBI)			Net based w/ BaSTA
	B cost	min D (ns)	B cost	CPU(s)	B cost	% redu	CPU(s)	B cost
a1	204	10.5	127	0.7	96	24.41	0.7	105
a2	274	20	144	0.5	138	4.17	1.1	142
a3	484	21	216	1.0	177	18.06	2.0	206
a4	592	26.5	317	1.3	288	9.15	2.8	325
a5	645	43	357	1.0	338	5.32	3.6	354
a6	1009	28.3	285	2.1	232	18.60	4.3	262
a7	1033	61.5	384	1.5	307	20.05	8.4	384
a8	1314	33	434	2.6	359	17.28	5.5	430
a9	1238	84	461	1.8	363	21.26	16.1	462
a10	1700	54	492	3.7	414	15.85	13.6	489
a11	2991	105	841	5.9	766	8.92	39.0	803
a12	3925	122	1048	5.1	852	18.70	38.1	1046
Total	15409	—	5106	27.1	4330	(avg)15.20	135.2	5008

slack would be 20 units. Assume $RAT_i = 40$. Then, the adjusted RAT_i equals $40 - [(40 \times 20)/100] = 32$.

V. SUMMARY OF PBBI

The PBBI algorithm framework is summarized as pseudocodes shown in Algorithm 1. It starts with the buffer aware STA. Based on the timing estimation, we construct a list of critical paths and generate a list of distinct paths. After that, each path is converted into a merged routing tree, which matches the input of VGDP problem [4]. With off-path RAT estimation technique, the RATs for some sinks are adjusted and VGDP algorithm is performed. The circuit timing is then updated and the next path is iterated.

Algorithm 1 PBBI

- 1: perform a buffer aware STA
- 2: obtain a list of k most critical paths
- 3: generate a list of distinct paths sorted by the criticality
- 4: **for** each of the distinct paths **do**
- 5: merge all routing trees along the path
- 6: **for** each non-PO off-path sink t **do**
- 7: let m be the module with input pin t
- 8: **if** the routing tree driven by m is traversed **then**
- 9: continue
- 10: **end if**
- 11: apply off-path RAT estimation described in Section IV-C
- 12: **end for**
- 13: Apply fast VGDP implementation on the merged tree
- 14: update the timing information at the fan-in/fan-out cones
- 15: **end for**
- 16: For each of other nontraversed nets, applies the VGDP algorithm

VI. PBBI AND SIMULTANEOUS GATE SIZING

The framework of our PBBI algorithm provides us the flexibility in integrating gate sizing into PBBI. It is due to the fact that when we cascade several routing trees into one merged tree, input pin and output pin of a module along the processing path is treated as one single vertex v , which is a special candidate buffer location and a buffer must be inserted at v while the resistance/delay/capacitance characteristic of the buffer is derived from that of the module's input-to-output path. In this point of view, if we also consider gate sizing with n choices of size, then we can derive n special buffers according to the sizes. Thus, by restricting the VGDP algorithm to insert one and only one buffer at v choosing from the n special buffers, path-based simultaneous buffer insertion and gate sizing is accomplished.

TABLE III
COMPARISON OF NET-BASED AND PBBI CONSIDERING BUFFER BLOCKAGES

circuit	Net based (min cost)		Path based (PBBI)		
	B cost	CPU(s)	B cost	% redu	CPU(s)
a1	103	0.5	83	19.42	0.7
a2	128	0.5	114	10.94	1.1
a3	195	0.9	151	22.56	1.9
a4	294	1.1	241	18.03	2.8
a5	309	0.9	304	1.62	3.8
a6	272	1.8	186	31.62	4.3
a7	343	1.3	248	27.70	8.5
a8	401	2.2	314	21.70	5.7
a9	423	1.7	294	30.50	16.1
a10	463	3.3	336	27.43	13.4
a11	787	5.6	657	16.52	31.2
a12	961	5.2	712	25.91	33.9
Total	4679	25.2	3640	(avg)22.21	123.4

Indeed, treating the gate sizing as selecting a buffer from a buffer library may cause errors because a gate can have more than one input and more than one output. When the algorithm changes the size of a gate along a path, for those input pins and output pins which are not along the path, their capacitance or resistance values change simultaneously without considering the impact resulted from the change. For example, in Fig. 3(a), the gate $m1$ can be sized along with the buffer insertion process on the path $b \rightarrow m1 \rightarrow r$. However, the change in gate size for $m1$ may in turn alter the capacitance of pin f on the path $a \rightarrow m1 \rightarrow m2 \rightarrow s$ which is assumed to be less critical. However, such sacrifice helps maintain the solution quality of the processing path (which must be more critical than the unprocessed paths) and we have verified this claim by our experimental results such that the algorithm can substantially reduce overall area of buffers and gates.

A. Gate Sizing at the Sinks

Along a path, using path-based simultaneous buffer insertion and gate sizing on the merged routing tree cannot solve the problem when the root of the merged routing tree also need gate sizing. It is especially true when the root of the merged routing tree is an output pin of a module. For example, in Fig. 2(c), after we already finished buffer insertion for the merged routing tree of $RT(b)$ and $RT(i)$, if we are processing the path $\{o, s\}$, the merged routing tree is just $RT(o)$ itself. In addition to applying VGDP to $RT(o)$, we have to perform gate sizing for the module $m2$ with pins $\{j, k, o\}$.

Sizing up a gate reduces the output resistance and so the delay is reduced. At the same time, the input capacitance increases, which in turn raises the delay of upstream interconnect. Such a delay increase can be handled by adding a delay penalty when doing gate sizing as

TABLE IV
COMPARISON OF NET-BASED AND PBBI CONSIDERING GATE SIZING

circuit	Net based (min cost)				Path based (PBBI+GS)					(PBBI+GS) w/o sink sizing		
	B cost	Δ G	Δ area	CPU(s)	B cost	Δ G	Δ area	% redu	CPU(s)	B cost	Δ G	Δ area
a1	115	36	151	0.6	81	8	89	41.06	1.5	109	9	118
a2	189	20	209	0.5	102	24	126	39.71	2.5	100	28	127
a3	278	44	322	0.9	151	10	161	50.00	7.0	161	12	173
a4	391	72	463	1.2	222	94	316	31.75	10.3	224	108	332
a5	488	112	600	0.9	218	61	279	53.50	15.7	328	70	398
a6	641	159	800	1.9	197	16	213	73.38	18.1	224	18	243
a7	779	198	977	1.3	235	34	269	72.47	44.9	250	39	289
a8	839	166	1005	2.4	310	52	362	63.98	19.2	416	60	476
a9	973	310	1283	1.7	279	42	321	74.98	61.9	299	48	348
a10	1089	216	1305	3.5	322	39	361	72.34	38.7	394	45	439
a11	2097	331	2428	6.2	587	68	655	73.02	93.2	664	78	742
a12	3149	837	3986	6.0	709	110	819	79.45	119.4	723	127	850
Total	11028	2501	13529	27.2	3413	558	3971	(avg)70.65	432.5	3892	642	4533

proposed in [25]. However, the main problem of this issue is that the increase in load capacitance may alter the other path delays of the previously buffered routing tree. In the above example, if we size up the module m_2 , the increase in input capacitance at j may increase the downstream load capacitance of $RT(i)$ and it may in turn increase the delay along the path $\{i, r\}$.

In order to fix this problem, we perform gate sizing at all sinks of the merged routing tree while we apply VGDP. If the sink vertex is not a PO, and if the module at the sink is not sized previously, we perform gate sizing for the module at the sink. Please note that similar to PBBI, each gate (for sizing) or routing tree (for buffer insertion) would only be processed once. At a sink s without gate sizing, VGDP starts to propagate one solution with the corresponding load capacitance, the RAT RAT_s , and zero cost (which means no buffer has been inserted up to s). With gate sizing at s , we propagate n solutions (n is the total number of different gate sizes), each of which have a scaled load capacitance, the gate size as its cost, and an modified RAT $RAT'_s(x_i)$. Assume that the original output resistance of the gate is R_0 and that of a sized gate is R_i , while R_b and C_b is the output resistance and input capacitance of the buffer used in buffer aware STA. Based on Elmore delay [19], we assume that there is a linear relationship between the delay change ($RAT'_s(x_i) - RAT_s$) and the change in resistance ($R_i - R_0$). Empirically, we found that (8) gives a good and effective calculation for $RAT'_s(x_i)$, where L_{opt} is the optimal buffer interval which is also used in [23]. Intuitively, $(R_i - R_0)(L_{opt}C + C_b)$ gives the change in delay if there exists a buffer in the downstream of s while the length between s and the buffer is L_{opt}

$$RAT'_s = RAT_s + (R_i - R_0)(L_{opt}C + C_b)$$

$$RAT'_s = RAT_s + (R_i - R_0) \left(\sqrt{\frac{2R_b C_b C}{R}} + C_b \right). \quad (8)$$

VII. EXPERIMENTAL RESULTS

We have implemented a very efficient VGDP buffer insertion algorithm according to [26] which uses approximation techniques to improve the efficiency. We have performed experiments on 12 combinational circuits which are randomly generated based on real nets from IBM. For simplicity, buffer cost refers to the number of buffers inserted, which is a rough approximation of the buffer area.¹ For our PBBI implementation, we find that searching for $k = 40000$ most critical paths is empirically good at trading-off between the quality and speed. All experiments were run on a Debian Linux machine with 2.4-GHz processor and 1-GB RAM. The size of each testcase

¹Practically in a buffer library, the variation in buffer area can be very large.

TABLE V
SUMMARY OF ISCAS PLACED AND ROUTED CIRCUITS

circuit	Circuit Size		
	# mod	# edge	# B candidate
c1355	619	1096	2377
c1908	938	1523	4647
c2670	1642	2292	7982
c3540	1741	2961	8971
c432	203	343	1015
c499	275	440	1413
c5315	2608	4509	13427
c7552	3828	6253	19291
c880	469	755	1944

TABLE VI
COMPARISON OF NET-BASED AND PBBI

circuit	Net based (min delay)		Net based (min cost)		Path based (PBBI)		
	B cost	min D (ns)	B cost	CPU(s)	B cost	% redu	CPU(s)
c1355	585	23.9	74	0.19	67	9.46	0.18
c1908	1027	50	106	0.3	82	22.64	0.38
c2670	2030	65	85	0.47	82	3.53	0.54
c3540	2082	82.7	100	0.54	77	23.00	0.64
c432	212	19.9	24	0.06	17	29.17	0.12
c499	300	19.6	39	0.09	31	20.51	0.14
c5315	3322	74.7	150	0.85	140	6.67	1.02
c7552	4411	75.6	240	1.15	187	22.08	1.50
c880	395	16.9	21	0.14	21	0.00	0.16
Total	14364	—	839	3.79	704	(avg)16.09	4.68

TABLE VII
SUMMARY OF TESTCASES IN [14]

circuit	Circuit Size			Delay constraint(ns)
	# modules	# edges	# B candidates	
n1	20	17	227	19.81
n2	40	37	402	28.79
n3	100	98	1095	32.03
n4	200	197	2398	43.76
n5	400	398	4417	50.96
n6	800	799	8803	56.90
n7	1200	1200	13078	57.06
n8	2000	1997	22721	65.43
n9	4000	3997	44471	77.80

is summarized in Table I. The second column refers to the number of modules in the circuit. The third column refers to the number of edges, which equals the total number of sinks for all routing trees. The column “total” is “mod” + “edge” which is a measure of the circuit size. “# B candidate” is the total number of buffer candidate locations for each circuit.

A. PBBI

We first compare our PBBI algorithm with the net-based buffer insertion algorithm using the same implementation of VGDP.

TABLE VIII
COMPARISON OF PBBI, NET-BASED, AND NETWORK-BASED BUFFER INSERTION

circuit	Network based (min cost)		Net based (min cost)		Path based (PBBI)			
	B cost (α)	CPU(s)	B cost (β)	CPU(s)	B cost (λ)	$\frac{\lambda-\alpha}{\alpha}$ (%)	$\frac{\lambda-\beta}{\beta}$ (%)	CPU(s)
n1	84	0.19	105	0.00	76	9.52	27.62	0.01
n2	120	0.39	91	0.02	64	46.67	29.67	0.07
n3	306	1.01	247	0.03	159	48.04	35.63	0.06
n4	420	2.10	356	0.06	336	20.00	5.62	0.15
n5	753	12.08	452	0.20	377	49.93	16.59	0.32
n6	1575	10.29	985	0.45	953	39.49	3.25	0.76
n7	2264	24.40	1300	0.63	1280	43.46	1.54	1.09
n8	3456	48.52	2550	1.12	2126	38.48	16.63	1.91
n9	6858	188.31	4907	2.23	4301	37.28	12.35	3.26
Total	15836	287.29	10993	4.72	9672	38.92	12.02	7.63

In order to know the minimum achievable delay of the circuit with optimal buffer insertion, for each net in the circuit, a minimum-delay buffer insertion algorithm is applied, namely “Net based (min delay).” The minimum delay in n_s and the number of buffers inserted are listed in the second and third columns of Table II. We then set the delay constraint of a circuit to be its minimum achievable delay. For all the following comparisons between VGDP and PBBI, the worst slacks of the circuit for both methods are similar and are not shown in the tables. Note that, “Net based (min delay)” applies buffer insertion for all nets in order to get the minimum achievable delay, hence it is not the same algorithm described in Section III. The net-based buffer insertion algorithm in Section III refers to the column “Net based (min cost).” For net-based buffering, we tried several different orderings (see Appendix) and made the conclusion that, on average, all tested orderings perform similarly in terms of total buffer cost. As a result, in our experimental results, we used the ordering based on ascending order of worst slack for comparison. The results are shown in Table II. The column “B cost” stands for the total number of buffers inserted while “% redu” is the percentage of buffer cost reduction when PBBI is compared to net-based buffer insertion (min cost) which minimizes buffer cost while subjects to delay constraint, as described in Section III.

We also performed similar experiments on net-based buffer insertion with buffer aware STA. The result is shown in the last column of Table II. From the results, net-based buffer insertion (min cost) with/without buffer aware STA performs similarly in terms of buffer resource allocation. This implies the buffer resource allocation of net-based buffer insertion cannot be easily improved by only applying simple slack distribution methods. In our experiments, we also found that if off-path RAT estimation is not applied, the resultant buffering solution generated by PBBI would still violate the timing constraint, which matches our speculation in Section IV-C and proves the effectiveness of the off-path RAT estimation technique.

From Table II, we have the following conclusions.

- 1) The average reduction in buffer area is 15% for PBBI algorithm when comparing to the net-based buffer insertion (min cost).
- 2) The average reduction is 72% for PBBI when comparing to the net-based buffer insertion (min delay). At the same time, the average reduction is 66% for net-based buffer insertion (min-cost). The results are expected because most paths would not be critical and require fewer buffers.
- 3) Although there is more than a $5 \times$ increase in CPU-time, the total CPU time for the biggest circuit with more than 3000 modules is still less than 1 min. In fact, the CPU time is empirically linear to the size of the circuit and the buffer candidate locations.

In our next experiment, we consider buffer blockages and the results are shown in Table III. From the results, we obtained an even greater buffer cost reduction of 22% when comparing to the net-

TABLE IX
COMPARISON OF NET-BASED AND PBBI WITH DIFFERENT TIMING CONSTRAINT

timing constraint	Net based (min cost)		Path based (PBBI)		
	B cost	CPU(s)	B cost	% redu	CPU(s)
1.00	5106	27.13	4330	15.20	135.20
1.05	4647	27.08	3644	21.58	124.75
1.10	4443	27.43	3322	25.23	119.06
1.15	4226	28.34	2955	30.08	114.45
1.20	4152	27.36	2756	33.62	108.82
1.25	4041	27.47	2541	37.12	104.80
1.30	3766	27.34	2342	37.81	101.49
1.35	3537	27.73	2147	39.30	98.54
1.40	3426	27.26	2036	40.57	96.17
1.45	3283	27.73	1833	44.17	94.15
1.50	3154	28.21	1714	45.66	92.29

based buffering. From our observation, it is due to the fact that buffer insertion with blockages becomes more complicated, and an algorithm with a global view would perform better.

B. Simultaneous Gate Sizing and Buffer Insertion

We have performed similar experiments for simultaneous gate sizing and buffer insertion. For a net-based approach, we have implemented the delay penalty scheme [25] which includes driver sizing into the net-based buffer insertion process. However, the delay penalty formula used in this paper is mainly for solution comparison in VGDP. When applying the delay penalty as a mean to estimate the delay increase in an upstream interconnect, we have to scale the delay penalty empirically. In our implementation, the delay penalty is scaled by 0.3. The results are shown in Table IV. In the table, “ ΔG ” is the total size change in all gates, “ $\Delta area$ ” is the total increase in cost (area) from the buffer insertion/gate sizing, and “% redu” is the percentage of total cost reduction (i.e., “ $\Delta area$ ” reduction) when PBBI + GS is compared to the net-based approach.

From Table IV, we found that the overall cost reduction by PBBI algorithm is 71% when comparing to net-based gate sizing/buffer insertion. As the same time, our PBBI + GS algorithm is reasonably efficient as the runtime is within 2 min for the biggest testcase.

Our novel technique of performing gate sizing at sink results in significant improvement in solution quality. For [25], the delay penalty is calculated at the driver of a routing tree. It is simple and effective but it only reflects the delay increase in the upstream interconnect toward the gate and the routing tree itself. We have performed experiments for our PBBI algorithm using different gate sizing schemes, and results show that gate sizing using delay penalty takes 14% more area than our gate sizing at sink technique (shown in the last three columns of Table IV). In conclusion, the lack of a global view results in being trapped into a local optimal solution and the error exacerbates when the complexity of the problem increases.

TABLE X
BUFFER INSERTION RESULTS FOR 20 TESTCASES WITH DIFFERENT NET PROCESSING ORDER

order- ing	Total number of buffers inserted																			
	test circuits																			
	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16	c17	c18	c19	c20
a	148	117	54	128	126	86	84	180	167	133	118	99	104	80	65	47	227	559	467	382
b	135	117	82	135	113	85	83	180	163	145	133	116	100	61	52	37	229	455	375	329
c	116	103	69	120	119	89	82	172	135	127	114	82	106	101	74	44	236	446	378	292
d	145	125	64	131	130	104	114	178	180	153	120	100	104	98	66	42	228	483	487	417
e	136	110	73	128	126	104	81	169	135	124	114	92	106	95	74	44	227	443	380	287
f	116	103	70	128	119	89	65	168	140	115	113	85	106	102	72	45	236	456	392	294
g	146	115	44	128	128	107	78	168	132	110	84	83	105	92	67	57	227	526	498	429
h	136	110	73	128	126	104	81	169	132	116	114	92	107	96	75	52	227	445	382	289
i	136	105	52	118	118	104	78	164	143	130	103	96	109	86	49	42	160	424	381	289
j	132	101	52	118	118	104	78	164	143	124	103	96	109	81	45	42	159	424	381	289
avg	135	111	63	126	122	98	82	171	147	128	112	94	106	89	64	45	216	466	412	330
	Percentage difference compared to the average (%)																			
a	10	6	-15	1	3	-12	2	5	14	4	6	5	-2	-10	2	4	5	20	13	16
b	0	6	30	7	-8	-13	1	5	11	14	19	23	-5	-32	-19	-18	6	-2	-9	0
c	-14	-7	9	-5	-3	-9	0	0	-8	-1	2	-13	0	13	16	-3	9	-4	-8	-11
d	8	13	1	4	6	7	38	4	22	20	8	6	-2	10	3	-7	6	4	18	26
e	1	-1	15	1	3	7	-2	-1	-8	-3	2	-2	0	7	16	-3	5	-5	-8	-13
f	-14	-7	11	1	-3	-9	-21	-2	-5	-10	1	-10	0	14	13	0	9	-2	-5	-11
g	8	4	-30	1	5	10	-5	-2	-10	-14	-25	-12	-1	3	5	26	5	13	21	30
h	1	-1	15	1	3	7	-2	-1	-10	-9	2	-2	1	8	17	15	5	-5	-7	-12
i	1	-5	-18	-6	-4	7	-5	-4	-3	2	-8	2	3	-4	-23	-7	-26	-9	-8	-12
j	-2	-9	-18	-6	-4	7	-5	-4	-3	-3	-8	2	3	-9	-30	-7	-26	-9	-8	-12

C. PBBI on ISCAS Circuits

From [27], we obtained the layout and parasitic information for some placed and routed ISCAS85 circuits in 180-nm technology. In order to emulate the latest technology, we scale the interconnect by a constant factor. The size of each testcase is summarized in Table V.

We performed experiments on the ISCAS circuit with settings similar to Table II. The results are shown in Table VI. We found that the results are similar to Table II while PBBI outperforms net-based buffer insertion for buffer cost minimization by 16% on average. One special observation is that runtime of PBBI is actually very similar to the net-based counterpart.

We found that the efficiency is due to that fact that the PI to PO paths in the ISCAS circuits are comparatively shorter than our random testcases in terms of number of modules along the paths. In such a case, the resultant merged routing trees for VGDP are smaller.

D. Comparison of PBBI, Net-Based and Network-Based Buffer Insertion

In this section, we compare our PBBI algorithm with the net-based and network-based buffer insertion [14] on a set of testcases randomly generated in [14]. The experiments are based on the 100-nm technology. The information of all testcases is summarized in Table VII.

The comparisons are shown in Table VIII. The columns show the buffer cost and CPU runtime for all three circuit-buffering schemes. The columns $((\lambda - \alpha)/\alpha)$ and $((\lambda - \beta)/\beta)$ also list the percentage reduction of PBBI over the network-based and net-based buffering algorithms, respectively. The table reveals that both net-based buffer insertion and PBBI outperform the network-based scheme in terms of both CPU time and buffer resource. We can also conclude from the table that:

- 1) PBBI takes on average 12% less buffer resources than net-based buffer insertion, which matches the experiment results on our testcases and ISCAS circuits;
- 2) PBBI requires 39% less buffer resources over the network-based scheme while net-based buffer insertion requires 31% less buffer resource;

- 3) both PBBI and net-based scheme runs much faster than the network-based algorithm. The speedup is $61\times$ for net-based buffer insertion and $38\times$ for PBBI; and
- 4) PBBI takes less than $2\times$ CPU runtime when comparing to net-based buffer insertion.

E. Experiments With Different Timing Constraints

In the previous experiments, we adopt a tight timing constraints. We first find the minimum achievable circuit delay and set the RAT accordingly. This can demonstrate the ability of PBBI in allocating the buffer resource when the slack of the circuit is very small. However, one would also like to compare the buffering results with different timing constraints. We use our 12 testcases shown in Table I and the results are shown in Table IX. In the table, the first column is the multiplier to the minimum achievable delay applied. For example, 1.35 represents the experiment settings when the timing constraints are 35% larger than the minimal ones. For a1 in Table II, the minimum achievable delay is 10.5, and the timing constraints becomes $10.5 \times 1.35 = 14.175$ ps for the "1.35" case. Table IX also shows the total buffer cost and total CPU runtime for both net-based buffer insertion and PBBI, respectively. The column "% redu" lists the buffer resource reduction of PBBI over the net-based scheme.

From Table IX, we discover that when the timing constraint of the circuit is getting looser from 1.00 to 1.50:

- 1) buffer resource reduction of PBBI over net-based buffer insertion increases from 15.20% to 45.66%;
- 2) CPU time used for net-based buffer insertion remains similar;
- 3) CPU time used for PBBI decreases from 135 to 92 s.

The results show that for a circuit with loose timing constraint, PBBI performs even better in allocating the buffer resources due to its advantage of having global view for the circuit timing.

VIII. CONCLUSION

The VLSI technology scaling requests increasingly more buffers in circuit designs and therefore buffer insertion needs to be carried in a more elaborate manner. As a departure from traditional net-based

buffer insertion methods, we propose a PBBI approach which can obtain a better buffer usage efficiency due to its global view. To the best of our knowledge, this is the first work on PBBI. Compared to network-based methods, our approach is more practical in terms of computation complexity. Several techniques are proposed along with the path-based buffering including buffer aware STA, slack spreading along off-paths, and simultaneous sink sizing. Experimental results show that our approach can reduce buffer/gate cost by 71% on average compared to net-based methods.

APPENDIX

DIFFERENT ORDERING FOR NET-BASED BUFFER INSERTION

In order to find out the best ordering for net-based buffer insertion in terms of buffer cost minimization subject to timing constraints, we performed a set of experiments on 20 randomly generated circuits. In the experiment, we have tested the following ordering:

- a) topological order from PI to PO;
- b) topological order from PO to PI;
- c) first process the net with the smallest worst slack;
- d) random;
- e) pick a path according to the descending order of its criticality, then pick a net from PI to PO;
- f) pick a path according to the descending order of its criticality, then pick a net from PO to PI;
- g) according to the descending order of the total load capacitance of the nets;
- h) pick a path according to the descending order of its criticality, then according to the descending order of the total load capacitance of the nets;
- i) first process the net such that most critical paths are passing through it;
- j) assign a cost value to each path representing its criticality, then first process the net such that it has the biggest total cost from all the paths passing through it.

The results are shown in Table X. In the upper half, each column represents one circuit and all numbers shown in the table are numbers of buffer inserted. In the lower half, the percentage difference is shown. From the table, we observed that no single ordering outperforms other orderings for most testcases. In fact, all tested ordering performs on average similarly in terms of total buffer cost.

REFERENCES

- [1] H. B. Bakoglu, *Circuits, Interconnections and Packaging for VLSI*. Reading, MA: Addison-Wesley, 1990.
- [2] J. A. Davis, R. Venkatesan, A. Kaloyeros, M. Beylansky, S. J. Souri, K. Banerjee, K. C. Saraswat, A. Rahman, R. Reif, and J. D. Meindl, "Interconnect limits on gigascale integration (GSI) in the 21st century," *Proc. IEEE*, vol. 89, no. 3, pp. 305–324, Mar. 2001.
- [3] L. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," in *Proc. IEEE Int. Symp. Circuits and Syst.*, 1990, pp. 865–868.
- [4] J. Lillis, C. K. Cheng, and T. Y. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," *IEEE J. Solid-State Circuits*, vol. 31, no. 3, pp. 437–447, Mar. 1996.
- [5] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer insertion for noise and delay optimization," in *Proc. ACM/IEEE Des. Autom. Conf.*, 1998, pp. 362–367.
- [6] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer insertion with accurate gate and interconnect delay computation," in *Proc. ACM/IEEE Des. Autom. Conf.*, 1999, pp. 479–484.
- [7] C. C. N. Chu and D. F. Wong, "A quadratic programming approach to simultaneous buffer insertion/sizing and wire sizing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 6, pp. 787–798, Jun. 1999.
- [8] S. Dhar and M. A. Franklin, "Optimum buffer circuits for driving long uniform lines," *IEEE J. Solid-State Circuits*, vol. 26, no. 1, pp. 32–38, Jan. 1991.
- [9] C. C. N. Chu and D. F. Wong, "Closed form solution to simultaneous buffer insertion/sizing and wire sizing," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 6, no. 3, pp. 343–371, Jul. 2001.
- [10] W. Shi and Z. Li, "A fast algorithm for optimal buffer insertion," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 6, pp. 879–891, Jun. 2005.
- [11] P. Saxena, N. Menezes, P. Cocchini, and D. A. Kirkpatrick, "Repeater scaling and its impact on CAD," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 4, pp. 451–463, Apr. 2004.
- [12] R. Murgai, "Layout-driven area-constrained timing optimization by net buffering," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2000, pp. 379–386.
- [13] R. Murgai, "Improved layout-driven area-constrained timing optimization by net buffering," in *Proc. Int. Conf. VLSI Des.*, 2005, pp. 97–102.
- [14] I.-M. Liu, A. Aziz, D. F. Wong, and H. Zhou, "An efficient buffer insertion algorithm for large networks based on Lagrangian relaxation," in *Proc. IEEE Int. Conf. Comput. Des.*, 1999, pp. 614–621.
- [15] I.-M. Liu, A. Aziz, and D. F. Wong, "Meeting delay constraints in DSM by minimal repeater insertion," in *Proc. Des., Autom. and Test Eur. Conf.*, 2000, pp. 436–441.
- [16] R. Chen and H. Zhou, "Efficient algorithms for buffer insertion in general circuits based on network flow," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2005, pp. 322–326.
- [17] Y. Jiang, S. S. Sapatnekar, C. Bamji, and J. Kim, "Interleaving buffer insertion and transistor sizing into a single optimization," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 6, no. 4, pp. 625–633, Dec. 1998.
- [18] C. N. Sze, C. J. Alpert, J. Hu, and W. Shi, "Path based buffer insertion," in *Proc. ACM/IEEE Des. Autom. Conf.*, 2005, pp. 509–514.
- [19] W. C. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *J. Appl. Phys.*, vol. 19, no. 1, pp. 55–63, Jan. 1948.
- [20] C. V. Kashyap, C. J. Alpert, F. Liu, and A. Devgan, "Closed-form expressions for extending step delay and slew metrics to ramp inputs for RC trees," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 4, pp. 509–516, Apr. 2004.
- [21] J. Lillis, "Algorithms for performance driven design of integrated circuits," Ph.D. dissertation, Univ. California-San Diego, San Diego, CA, 1996.
- [22] S. S. Sapatnekar, *Timing*. Boston, MA: Kluwer, 2004.
- [23] C. J. Alpert, J. Hu, S. S. Sapatnekar, and C. N. Sze, "Accurate estimation of global buffer delay within a floorplan," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2004, pp. 706–711.
- [24] Y.-C. Ju and R. Saleh, "Incremental techniques for the identification of statically sensitizable critical paths," in *Proc. ACM/IEEE Des. Autom. Conf.*, 1991, pp. 541–546.
- [25] C. J. Alpert, C. Chu, G. Gandham, M. Hrkic, J. Hu, C. Kashyap, and S. T. Quay, "Simultaneous driver sizing and buffer insertion using delay penalty estimation technique," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 1, pp. 136–141, Jan. 2004.
- [26] Z. Li, C. N. Sze, C. J. Alpert, J. Hu, and W. Shi, "Making fast buffer insertion even faster via approximation techniques," in *Proc. Asia and South Pacific Des. Autom. Conf.*, 2005, pp. 13–18.
- [27] X. Lu and W. Shi, *Layout and Parasitic Information for ISCAS Circuits*, 2003. [Online]. Available: <http://dropzone.tamu.edu/~xiang/fiscas.html>