Efficient Algorithms for Buffer Insertion in General Circuits Based on Network Flow*

Ruiming Chen and Hai Zhou
Electrical and Computer Engineering
Northwestern University
Evanston, IL 60208

Abstract

With shrinking VLSI feature sizes and increasing overall chip areas, buffering has emerged as an effective solution to the problem of growing interconnect delays in modern designs. The problem of buffer insertion in a single net has been the focus of most previous researches. However, efficient algorithms for buffer insertion in whole circuits are generally needed. In this paper, we relate the timing constrained minimal buffer insertion problem to the min-cost flow dual problem, and propose two algorithms based on *min-cost flow* and *min-cut* techniques, respectively, to solve it in combinational circuits. We compare our approaches to a traditional approach based on Lagrangian relaxation. Experimental results demonstrate that our approaches are efficient and effective. On the average, our approaches achieve 45% and 39% reduction, respecitively, on the number of buffers inserted in comparison to the traditional approach.

1 Introduction

Interconnect delays have become dominant in modern deep sub-micron designs with shrinking VLSI feature sizes and increasing chip areas. Buffer insertion is widely used to reduce interconnect delays [3–6, 10, 12–14, 17, 20–22]. Recent projections of historical scaling trends by Saxena et al. [19] predict synthesis blocks to have 70% of their cell count dedicated to interconnect buffers within a few process generations. Consequently, there is an increasing demand for *efficient* buffer insertion approaches.

Van Ginneken [21] proposed a dynamic programming method to buffering in distributed RC-tree networks for minimal Elmore delay [8]. Shi and Li [20] presented an $O(n \log^2 n)$ algorithm for the optimal buffer insertion problem, where n is the number of legal buffer positions. Chen and Zhou [6] presented a flexible data structure to get a universal speed up in buffer insertion. However, all these approaches considered the buffer insertion problem only on a single net.

In reality, it is often required to insert buffers in a large network under a given timing constraint. It is unnecessary to optimally buffer each net to the minimal delay; nets on non-critical paths do not need to have minimal delays. Therefore, optimally buffering each net leads to over-buffering. In addition, if the given timing constraint is larger than the minimal delay that can be achieved by buffering, assignment of various timing budgets on the critical paths would give multiple buffering solutions, among which we need to select the one with minimal buffers. We thus need to insert buffers considering a global view instead of a local view. Liu et al. [16] presented a Lagrangian relaxation based algorithm to solve the buffer insertion problem in large networks. It was extended to consider multiple buffer types and feasible buffer locations in [15]. The objective function of the Lagrangian relaxation based algorithm is $\alpha \sum_{e \in E} K_e$, where K_e is the number of the buffers on edge e, and α is a specified weight. However, the weight is sensitive and may greatly influence the final results. To the best of our knowledge, there is no good weight selection method to obtain the best solution, and our experiments show that the results obtained using a Lagrangian relaxation based technique is significantly sub-optimal.

In this paper, we relate the timing constrained buffer insertion problem to the network flow problem, and present two efficient timing constrained minimum buffer insertion algorithms based on network flow theory. We assume that the buffers are of a fixed size. The forbidden areas for buffer insertion are handled smoothly in this paper. The basic idea is to solve the delay distribution among the wires via an iterative separable convex network flow optimization. Experimental results show that the number of buffers inserted by our algorithms is always less than the number of buffers inserted by [16] under the same timing constraint, and that our algorithms are efficient.

2 Problem formulation

The input to our problem is a placed and routed netlist of modules with drivers and loads. Our objective is to insert buffers into wires in the general combinational circuit such that the timing constraint is met and the number of buffers is minimized.

As in [16], we use a directed acyclic graph (DAG) $G(V, E)$ to represent the circuit, of which the vertices correspond to the primary inputs, the primary outputs, tree junctions and module inputs/outputs. Two dummy nodes s and t are introduced: s is connected to all the primary inputs, and t is connected from all the primary outputs. The edge set E includes two disjoint sets of edges E^P and E^F, corresponding to the buffer allowable wires and the buffer forbidden edges (wires or modules), respectively.

The notations used in this paper are given in Table 1. The problem is formulated as:

$$Minimize \quad \sum_{(i,j) \in E^P} K_{ij} \tag{1}$$

$$\text{s.t.} \quad a_i + d_{ij} \le a_j \qquad \forall (i,j) \in E \tag{2}$$

$$a_t - a_s \le REQ \tag{3}$$

where a_i is the arrival time at vertex i, and REQ is the timing constraint.

In the buffer insertion problem, the delay change of one component influences the delays of many other components. As shown in Figure 1, a net is composed by wires a, b and c. If a buffer B is inserted into wire c, the delay of c is changed, and according to Elmore delay model, the delay of a also changes, so the delay from s_0 to s_2 also changes. Actually, since the delay of the driver gate is related with the load capacitance, it may change too. This kind of relations between component delays make the buffer insertion problem very difficult.

We use the Elmore delay model for wires, modules and buffers as in [16]. Given a wire with buffers inserted, we can easily compute the delays. In this paper, we only consider the single buffer type, then from [16], we know that the wire

*This work was supported by NSF under CCR-0238484.

Table 1: Notations

R_b	output resistance of a buffer
C_b	input capacitance of a buffer
t_b	intrinsic delay of a buffer
K_e	number of buffers on wire e
L_e	length of wire e
p_e	wire length from the input of wire e to the first buffer of wire e
q_e	wire length from the last buffer of e to the output of wire e
m_e	wire length between two adjacent buffers in wire e
$\hat{R}_e$	upstream resistance at the input of wire e
$\hat{C}_e$	downstream capacitance at the output of wire e
d_e	delay of the wire e
S_e	timing slack of the wire e
R	resistance of a unit-length wire
C	capacitance of a unit-length wire
S_{th}	timing slack threshold
δ_e	max delay change of wire e w/ one more buffer
c_e	capacity of wire e
f_e	flow through wire e

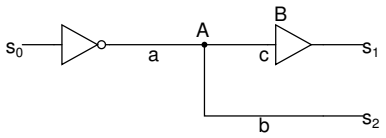

Figure 1: The influence of an inserted buffer.

lengths between two consecutive buffers on one wire are equal. When $K_e > 0$, the delay of wire e is equal to

$$\begin{aligned} d_e &= Rp_e(Cp_e/2 + C_b) \\ &\quad +(K_e - 1)(R_b(C_b + Cm_e) + t_b + Rm_e(C_b + Cm_e/2)) \\ &\quad +t_b + R_b(\hat{C}_e + Cq_e) + Rq_e(\hat{C}_e + Cq_e/2), \end{aligned} \tag{4}$$

We need to consider the contribution of the capacitance of e to the delay of the fanin edge of (e), so let

$$F_e = d_e + \hat{R}_e(Cp_e + C_b). \tag{5}$$

Then let $\frac{\partial F_e}{\partial p_e} = 0$, and $\frac{\partial F_e}{\partial q_e} = 0$. We can compute the optimal values of p_e and q_e for a given K_e such that F_e is minimized. The optimal values of p_e and q_e are given in [16]. When we get the optimal p_e and q_e for a given K_e, the delay of wire e for K_e is easily computed according to Eq. 4. Based on this, the maximal delay change of wire e when a new buffer is inserted into it, denoted as δ_e, can be computed. Since the wire delay is expected to decrease when a new buffer is inserted, we have $\delta_e < 0$. The *delay sensitivity* of component e is defined as $-1/\delta_e$.

In the reality, there exist some forbidden areas for buffer insertion. The buffering of the components in forbidden areas will not influence the delays of those components, which means $\delta_e = -0$ for any component e in the forbidden areas, and its delay sensitivity is infinity. Modules are forbidden areas for buffer insertion, so their delay sensitivities are infinity.

3 Network flow based buffering

3.1 Min-cost flow based buffering

The buffer insertion problem can be viewed as a wire substitution problem. As shown in Figure 2, a wire has different delays for different number of inserted buffers. The objective is to select a point in the configuration of each wire such that the total number of buffers is minimized while the timing constraints are satisfied. The advantage of this idea is that during the buffer insertion, the positions of inserted buffers can change such that the required delay objective is achieved.

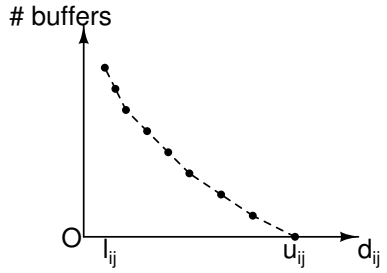

Figure 2: Number of buffers as a function of the wire delays.

Firstly, we consider a simplified buffer insertion problem: the module delays are fixed, and all the nets have only two pins, so there are no relations between component delays.

Let $K_e = \mathcal{C}_e(d_e)$, where $\mathcal{C}_e$ is the reverse function that gives the number of buffers for a given delay d_e. In this paper, we only consider the buffer insertion in combinational circuits. We introduce one edge from t to s with weight $-REQ$. This new graph is called a *constraint graph*.

As shown in Figure 2, we can also see that, on the same wire, with more and more buffers inserted, the amount of delay reduction by adding a buffer is getting smaller and smaller, and will finally become zero when the minimal delay is reached. Such a property of decreasing delay reduction with the increasing number of buffers is similar to the concept of convexity. Actually, the constraint graph excluding the edge from t to s is a DAG, the $\mathcal{C}_{ij}$ has only non-negative integer values, and d_{ij}s can only be discrete values. To the best of our knowledge, currently no algorithms can solve this problem optimally. We connect each node to its nearest nodes as in Figure 2 to get a piece-wise linear convex function $\mathcal{C}_{ij}$. For any edge $(i,j) \in E^P$, let l_{ij} be the minimal delay of edge (i,j) that can be achieved by buffering, and u_{ij} be the delay of edge (i,j) without buffers. Let $l_{ts} = u_{ts} = -REQ$, and $l_{ij} = u_{ij}$ for any edge $(i,j) \in E^F$. Let $\mathcal{C}_{ij}(d_{ij})$ be equal to ∞ when $d_{ij} < l_{ij} \vee d_{ij} > u_{ij}$. It is obvious that if there is an optimal solution for the buffering problem, $l_{ij} \leq d_{ij} \leq u_{ij}$.

Then the timing constrained buffering problem can be formulated as: Given a constraint graph $G(V,E)$,

$$\begin{aligned} Minimize \quad & \textstyle\sum_{(i,j)\in E} \mathcal{C}_{ij}(d_{ij}) \\ \text{s.t.} \quad & t_j \geq t_i + d_{ij} \qquad \forall (i,j) \in E \end{aligned}$$

If the objective function is a summation of convex functions of each d_{ij}, the buffering problem is the dual problem of a min-cost flow problem where the objective function is nonlinear [11], and is solved by Karzanov et al. [11]. When the objective function becomes a summation of convex functions of each d_{ij} and the variables are integers, Ahuja, Hochbaum, and Orlin [1] proposed a modified cost-scaling algorithm called *convex cost-scaling algorithm* to solve it efficiently.

When the relations between component delays are considered, the problem is more complicated: for a given d_{ij}, $\mathcal{C}_{ij}(d_{ij})$ may get different values for different loadings, so it is impossible to get closed form formulas for $\mathcal{C}_{ij}$s. In addition, $\mathcal{C}_{ij}$ has a discrete domain with an integer range. The algorithms in [1, 11] cannot be used here.

Now we design a heuristic to directly handle the buffering problem in combinational circuits. The KKT optimality

Algorithm MinCost-Buffering
$d_{ij} \leftarrow u_{ij}\ \forall (i,j) \in E$
$c_{(i,j)} \leftarrow -\mathcal{C}^-_{ij}(d_{ij})\ \forall (i,j) \in E \wedge S_{ij} < 0$
$c_{(i,j)} \leftarrow 0\ \forall (i,j) \in E \wedge S_{ij} \geq 0$
while there exist positive cycles in G
  Augment maximal flows using s as the source and t as the sink;
  Select a min-cut M;
  Insert one buffer into $(i,j)\ \forall (i,j) \in M$;
  UpdateTimingSlack(G);
  $\mathbf{c_{(i,j)} \leftarrow -\mathcal{C}^-_{ij}(d_{ij}) - f_{ij}\ \forall (i,j) \in E \wedge S_{ij} < 0}$
  $c_{(i,j)} \leftarrow 0\ \forall (i,j) \in E \wedge S_{ij} \geq 0$

Figure 3: Min-cost flow based buffering algorithm

conditions [18] become [11]:

$$t_j \geq t_i + d_{ij} \quad \forall (i,j) \in E \tag{6}$$

$$-\mathcal{C}^+_{ij}(d_{ij}) \leq x_{ij} \leq -\mathcal{C}^-_{ij}(d_{ij}) \quad \forall (i,j) \in E \tag{7}$$

$$\sum_{(i,j)\in E} x_{ij} - \sum_{(j,i)\in E} x_{ji} = 0 \quad \forall i \in V \tag{8}$$

$$x_{ij} \geq 0 \quad \forall (i,j) \in E \tag{9}$$

$$x_{ij}(t_j - t_i - d_{ij}) = 0 \quad \forall (i,j) \in E \tag{10}$$

According to Eq. (10), if there exists a non-zero flow on an edge (i,j), $t_j = t_i + d_{ij}$, which is difficult to satisfy. We discard condition (10), and design a heuristic based on min-cost flow to satisfy all the other conditions. The pseudo-code of this algorithm is shown in Figure 3. The capacity of any wire (i,j) is a function of its delay, and is set to be $-\mathcal{C}^-_{ij}(d_{ij})$. One thing we need to note is that when the flow flows through an edge, we do not introduce a residual backward edge, that is, the flows always flow through the forward edges. After each iteration, conditions (7)-(9) are always true. When the algorithm terminates, condition (6) is also true. We thus avoid considering the residual backward edges.

Theorem 1 *The solution generated by MinCost-Buffering satisfies conditions (6)-(9).*

3.2 Min-cut based buffering

Algorithm MinCut-Buffering
maxdelay $\leftarrow$ ComputeTimingAndSlack(G);
$d_{ij} \leftarrow u_{ij}\ \forall (i,j) \in E$
$c_{(i,j)} \leftarrow -\mathcal{C}^-_{ij}(d_{ij})\ \forall (i,j) \in E$;
while maxdelay$> REQ$
  Find a min-cut M of G;
  Insert one buffer into $(i,j)\ \forall (i,j) \in M$;
  maxdelay $\leftarrow$ UpdateTimingSlack(G);
  $\mathbf{c_{(i,j)} \leftarrow -\mathcal{C}^-_{ij}(d_{ij})\ \forall (i,j) \in E \wedge S_{ij} < 0}$
  $c_{(i,j)} \leftarrow 0\ \forall (i,j) \in E \wedge S_{ij} \geq 0$

Figure 4: Min-cut based buffering algorithm

MinCost-Buffering computes the flows incrementally, so the flow through an edge before one iteration is the sum of the flows through this edge in the previous iterations. Intuitively, if we do not consider the historical flows, that is, the capacity of any edge (i,j) is set to be -$\mathcal{C}^-_{ij}(d_{ij})$ instead of the residual capacity, the number of inserted buffers might not be large, either. Based on this, we design a *greedy* algorithm based on min-cut, called MinCut-Buffering. The pseudo-code is shown in Figure 4. The major difference between MinCut-Buffering and MinCost-Buffering is that the historical flows are not subtracted from the capacities of the edges after each iteration in MinCut-Buffering, so the capacities are *not* the *residual* capacities. Although MinCut-Buffering uses the minimal cost in every step, the solution may not satisfy the conditions (7)-(9).

3.3 Implementation issues

The previous two subsections do not consider the relations between component delays. In this subsection, we consider them, so we need to decouple the branches. The general framework of network flow based buffering algorithms, called NetworkBIN, is shown in Figure 5. At the beginning, there are no buffers in the circuit. Firstly, set the required time of the sink t to be the timing constraint, and compute the slack of each wire. In the capacity setting step, the delay sensitivities of wires are computed. The capacity of any edge e (wire or module) satisfying $S_e < 0$ is set to be the delay sensitivity $-1/\delta_e$, and the capacities of all the other edges are set to be 0. Then we use Ford-Fulkerson algorithm [9] to compute the maximal flow from s to t, and simultaneously get the min-cut. Then one buffer is inserted into every wire in the min-cut.

Algorithm NetworkBIN
maxdelay $\leftarrow$ ComputeTimingAndSlack(G);
SetCapacity(G);
while maxdelay$> REQ$
  Find a min-cut of G;
  for each wire (u,v) in the found min-cut
    Insert one buffer into (u,v);
    Decouple the other wires that connect from u;
  maxdelay $\leftarrow$ UpdateTimingSlack(G);
  UpdateCapacity(G);

Figure 5: Network flow based buffer insertion algorithm

When the relations between component delays are considered, the sensitivity computation step becomes difficult. As shown in Figure 6(a), the net has four pins, and the slacks on the wires (b,c), (b,d) and (b,e) are much different, while these slacks are all negative. If NetworkBIN finds a min-cut through (b,c), (b,d) and (b,e), it will insert one buffer into each wire. After that, their slacks might still be negative, so more buffers might be required to be inserted. This solution is shown in Figure 6(b). But we may find a better solution by decreasing the delay of the wire (a,b) through decoupling some critical branches: since the slacks of the branches are different, we may only need to insert one buffer into the most critical branch, and insert buffers into other branches to decouple them. This solution is shown in Figure 6(c). The reason that NetworkBIN generates worse solutions is that the sensitivity computation is based on the assumption that no buffers will be inserted into all the other wires, but actually, the sensitivities of wires might change when new buffers are inserted into other wires. So when the relations between component delays are considered, the exact sensitivity computation becomes difficult. An approach to avoid this problem is to enforce that the min-cut contains at most one wire for each net, and the capacity setting accomplishes this. Thus, in each iteration, for each net, we set the capacity of the most critical branch to be its delay sensitivity, and set the capacities of other branches to be 0. In the example of Figure 6(a), we first insert buffers into the most critical branch $b \rightarrow e$, and

decouple the branches $b \rightarrow c$ and $b \rightarrow d$, so we get the solution as shown in Figure 6(c).

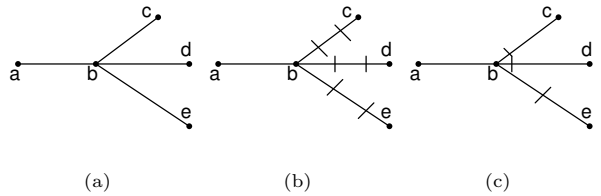

Figure 6: The sensitivity computation problem.

Although the general flows of the min-cost flow based and the min-cut based algorithms are the same, they have some important differences.

We first present our min-cost flow based buffering algorithm called CostBIN. The algorithm works as follows. First, we compute the slacks and timing information in ComputeTimingAndSlack. We use the timing constraint as the required time at sink t to compute slacks. Then we set the capacities of edges. For any edge e, if $S_e \geq 0$, it should be excluded from the network, so we set its capacity 0. As mentioned before, the found min-cut contains at most one wire for each net in each iteration, and we need to consider the influence of the decoupling buffers on the delay of the fanin edge of wire e. So when we compute the delay sensitivity of wire (u, v), we assume that all the other wire branches connecting from u are decoupled. Let $output(e)$ represent the set containing all the fanout edges of e. Then the maximal change of the sum of the delays of e and its fanin edge e' is

$$T_e = \delta_e + \hat{R}_{e'}(number(output(e'))C_b + Cp_e - \hat{C}_{e'}),$$

where function $number(S)$ represents the number of elements in set S. If $T_e < 0$, the capacity of wire e with $S_e < 0$ is set $-1/T_e - f_e$; otherwise, it is set infinity. Since the δ_e is -0 for any edge $e \in E^F$, its capacity is equal to infinity. The procedures followed are the same with the corresponding procedures in NetworkBIN.

We next present our min-cut based buffering algorithm called CutBIN. When we perform maximal flow algorithm to find a min-cut in MinCut-Buffering, the sub-network that the flow might flow through contains *all* the parts with negative slacks in a circuit, and this sub-network might be very large. An approach to speed up the algorithm is to restrict the flow at a sub-network that contains only the most critical paths. We use the current maximal delay from s to t as the required time at sink t, and compute the slacks of components. A subgraph containing only the components with slacks less than S_{th} is called the *critical subgraph* of the circuit. Besides the speeding up effect, S_{th} can also avoid the sensitivity computation problem in MinCut-Buffering. We use the following adaptive determination approach to select S_{th}: for each component, the difference between the slacks of the most critical and the second most critical fanout edges is computed, then select the minimal difference for all the components as S_{th}; if S_{th} is less than a specified lower bound, it is set to be the lower bound. The lower bound is used to speed up the algorithm. This step is performed once before the capacity setting step in each iteration, so the S_{th} may change for different iterations.

The CutBIN works as follows. First, we compute the slacks and timing information in ComputeTimingAndSlack. We use the current maximal delay from s to t as the required time at sink t to compute slacks, so all the slacks are nonnegative. Then we set the capacities of edges. For any edge e, if $S_e \geq S_{th}$, it should be excluded from the critical subgraph, so we set its capacity 0. As mentioned before, we need to consider the influence of decoupling buffers on the wire delay, so if $T_e < 0$, the capacity of wire e with $S_e < S_{th}$ is set $-1/T_e$, otherwise, it is set infinity. Since the δ_e is equal to -0 for any edge $e \in E^F$, its capacity is equal to infinity. The procedures followed are the same with the corresponding procedures in NetworkBIN.

4 Experimental results

We have implemented the CostBIN and CutBIN algorithms in C. We use the parameters from 100-nanometer technology [7]. We got four test cases from Liu, and generated additional four cases using the case generator in [16]. All experiments are run on a Linux PC with a 2.4 GHz Xeon CPU and 2.0 GB memory.

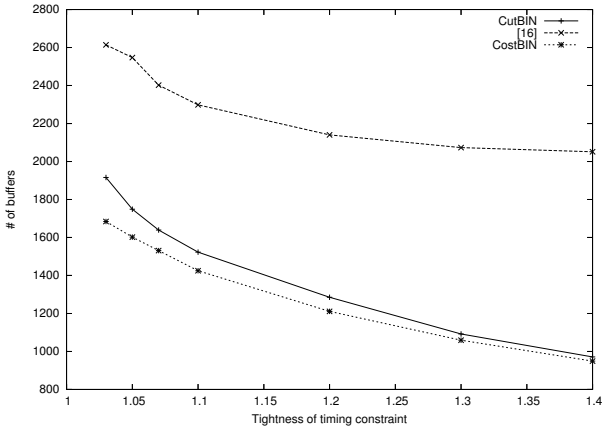

Figure 7: The influence of the tightness of timing constraint on the number of inserted buffers.

In order to test the benefit of the network flow based buffer insertion, we got the source code of the algorithm in [16] from Liu for comparison. The comparison results of CutBIN, CostBIN and [16] are shown in Table 2. We set the timing constraints to be 0.2 times larger than the minimal delays from s to t that can be achieved by buffering, which are computed by the min-delay buffering algorithm in [16]. The four subcolumns under "CutBIN" column show the number of buffers inserted by CutBIN, the running time of CutBIN, the reduction percentage of the number of buffers inserted by CutBIN compared with the number of buffers inserted by [16], and the speed-up of CutBIN over [16], respectively. The three sub-columns under "CostBIN" column show the number of inserted buffers by CostBIN, the running time of CostBIN, and the reduction percentage of the number of buffers inserted by CostBIN compared with the number of buffers inserted by [16], respectively.

We can see that CutBIN achieves 39% reduction of the number of inserted buffers on average compared with [16] and is much more efficient than [16], and CostBIN even achieves 45% reduction of the number of inserted buffers on average compared with [16]. We can also see that CostBIN always inserts less buffers than CutBIN. The reason behind this is that CostBIN considers the relations of different iterations by incremental flows, but CutBIN is only a greedy algorithm.

Using the case with 505 modules and 2,039 wires as an example, we test the influence of the tightness of timing constraint on the number of inserted buffers. As shown in Figure 7, both CostBIN and CutBIN save even more buffers than [16] when the timing constraints are looser, which means

Table 2: Comparison results of CutBIN, CostBIN and [16].

circuit		[16]		CutBIN				CostBIN		
# Module	# Wire	# Buffer	Time (s)	# Buffer	Time (s)	Reduction	Speed up	# Buffer	Time (s)	Reduction
22	98	172	0	153	0	11%	1×	127	0	35%
44	197	305	5	175	0.02	43%	250×	150	0.03	51%
81	398	606	16	311	0.05	49%	320×	286	0.10	53%
159	799	887	10	499	0.15	44%	67×	474	0.23	47%
258	1037	1096	28	773	0.29	29%	97×	718	0.88	34%
505	2039	2140	20	1285	1.11	40%	18×	1211	3.73	43%
2514	10039	10297	170	5860	21	43%	8×	5515	96	46%
5034	20038	21201	344	10620	67	50%	5.13×	9803	348	54%
Average						39%				45%

that they are more effective buffering approaches. Especially, CostBIN always inserts less buffers than CutBIN does under the same timing constraint, and when the timing constraint is tighter, CostBIN saves more buffers than CutBIN does. During our experiments, we observed that the results of [16] are very sensitive to the selection of the weights in the objective functions in the Lagrangian relaxation, and it is required to select the weights *manually* in order to get a good result.

5 Conclusions and future work

With the development of VLSI technology, buffering becomes an effective technique to the problem of growing interconnect delays in modern designs. Most previous researches were focusing on the problem of buffer insertion in a single net, which may introduce the over-buffering problem in a whole circuit. In this paper, we relate the timing constrained minimal buffer insertion problem to the min-cost flow dual problem, and propose two algorithms CostBIN and CutBIN based on min-cost flow and min-cut techniques, respectively, to solve the buffering problem in large combinational circuits. We compare our approaches to a Lagrangian relaxation based buffer insertion algorithm proposed by Liu et al. [16]. Experimental results demonstrate that our approaches are efficient and on the average, achieve 45% and 39% reduction, respectively, on the number of buffers inserted in comparison to the latter. The CostBIN algorithm always inserts less buffers than the CutBIN algorithm does, which implies the relations between iterations should be considered.

Neither CostBIN nor CutBIN has considered residual backward edges in the network. We plan to consider them to improve the results. The introduction of backward edges will make condition (10) satisfied for the convex integer min-cost flow dual problem, and might also benefit our buffering algorithms. But it might also lead to more iterations. Another important future work is to improve the efficiency of the min-cost flow based algorithms.

References

[1] R. K. Ahuja, D. S. Hochbaum, and J. B. Orlin. Solving the convex cost integer dual network flow problem. *Management Science*, 49:950–964, 2003.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Application*. Prentice Hall, 1993.

[3] C. J. Alpert and A. Devgan. Wire segmenting for improved buffer insertion. In *DAC*, pages 588–593, 1997.

[4] C. J. Alpert, M. Hrkic, and S. T. Quay. A fast algorithm for identifying good buffer insertion candidate locations. In *ISPD*, pages 47–52, 2004.

[5] C. J. Alpert, J. Hu, S. S. Sapatnekar, and C. N. Sze. Acurate estimation of global buffer delay within a floorplan. In *ICCAD*, pages 706–711, 2004.

[6] R. Chen and H. Zhou. A flexible data structure for efficient buffer insertion. In *ICCD*, pages 216–221, 2004.

[7] J. Cong and Z. Pan. Interconnect performance estimation models for synthesis and design planning. In *Workshop Notes of Intl. Workshop on Logic Synthesis*, pages 427–433, 1998.

[8] W. C. Elmore. The transient response of damped linear networks with particular regard to wide-band amplifiers. *Journal of Applied Physics*, 19(1):55–63, January 1948.

[9] J. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[10] M. Kang, W. W.-M. Dai, T. Dillinger, and D. LaPotin. Delay bounded buffered tree construction for timing driven floorplanning. In *ICCAD*, pages 707–712, 1997.

[11] A. V. Karzanov and S. T. McCormick. Polynomial methods for separable convex optimization in unimodular linear spaces with applications. *SIAM*, 26:1245–1275, 1997.

[12] V. Khandelwal, A. Davoodi, A. Nanavati, and A. Srivastava. A probabilistic approach to buffer insertion. In *ICCAD*, pages 560–567, 2003.

[13] J. Lillis, C. K. Cheng, T. T. Lin, and C. Y. Ho. New performance driven routing techniques with explicit area/delay tradeoff and simultaneous wire sizing. In *DAC*, pages 395–400, 1996.

[14] J. Lillis, C. K. Cheng, and T. T. Y. Lin. Optimal wire sizing and buffer insertion for low power and a generalized delay model. *IEEE Trans. Solid-State Circuits*, 31:437–447, 1996.

[15] I.-M. Liu, A. Aziz, and D. F. Wong. Meeting delay constraints in DSM by minimal repeater insertion. In *DATE*, pages 436–440, 2000.

[16] I.-M. Liu, A. Aziz, D. F. Wong, and H. Zhou. An efficient buffer insertion algorithm for large networks based on lagrangian relaxation. In *ICCD*, pages 210–215, 1999.

[17] T. Okamoto and J. Cong. Buffered Steiner tree construction with wire sizing for interconnect layout optimization. In *ICCAD*, pages 44–49, 1996.

[18] R. L. Rardin. *Optimization in Operations Research*. Prentice Hall, 1998.

[19] P. Saxena, N. Menezes, P. Cocchini, and Desmond A. Kirkpatrick. The scaling challenge: Can correct-by-construction design help? In *ISPD*, pages 51–58, 2003.

[20] W. Shi and Z. Li. An $O(n \log n)$ time algorithm for optimal buffer insertion. In *DAC*, pages 580–585, 2003.

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

[22] H. Zhou, D. F. Wong, I-Min Liu, and Adnan Aziz. Simultaneous routing and buffer insertion with restrictions on buffer locations. DAC, 1999.