Retiming with Interconnect and Gate Delay

Chris Chu[*], *Evangeline F. Y. Young*[†], *Dennis K. Y. Tong*[†], *and Sampath Dechu*[‡]

[*] Dept. of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011

[†] Dept. of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong

[‡] Physical Design Automation Group, Micron Technology, Inc., Boise, ID 83707

Abstract

In this paper, we study the problem of retiming of sequential circuits with both interconnect and gate delay. Most retiming algorithms have assumed ideal conditions for the non-logical portions of the data paths, which are not sufficiently accurate to be used in high performance circuits today. In our modeling, we assume that the delay of a wire is directly proportional to its length. This assumption is reasonable since the quadratic component of a wire delay is significantly smaller than its linear component when the more accurate Elmore delay model is used. A simple experiment is conducted to illustrate the validity of this assumption. We present two approaches to solve this problem, both of which have polynomial time complexity. The first one can compute the optimal clock period while the second one is an improvement over the first one in terms of practical applicability. The second approach gives solutions very close to the optimal (0.13% more than the optimal on average) but in a much shorter runtime. A circuit with more than 22K gates and 32K wires can be optimally retimed in 83.56 seconds by a PC with an 1.8GHz Intel Xeon processor.

1 Introduction

Retiming [1] is a useful and popular technique for performance optimization of sequential circuits. It relocates registers to reduce the cycle time while preserving the functionality of the circuit. Much effort has been made to apply this technique in different areas like power reduction [2, 3], testability [4, 5], logic resynthesis [6], circuit partitioning [7–9] and physical planning [10]. Some extended its applicability in large practical circuits efficiently [11–18]. However, most retiming algorithms have assumed ideal conditions for the non-logical portions of the data paths, specifically ignoring the interconnect delay. As process technology gets down to deep sub-micron, interconnect delay becomes a major factor of path delay. Without including this delay component, existing retiming algorithms are not sufficiently accurate to be used in practical high performance circuits.

The choice of an accurate interconnect delay model and an appropriate retiming algorithm are important. In some previous works [19, 20], interconnect delay was incorporated into the retiming process, but simplified assumptions were made such that the interconnect delay between adjacent registers on the same wire was neglected. Another approach to integrate retiming into detailed placement was presented in [21]. After an initial placement and routing, heuristics were used to estimate interconnect delay. Retiming and post-retiming placement were then performed to optimize the circuit performance. A recent paper [22] by Tabbara et al. applied retiming in the DSM domain and interconnect delay was considered. It was done by having a lower bound on the number of registers on each wire e_{uv}, while the delays at nodes were irrelevant. Registers could be retimed into a node that represented a component and affected the total area of the component. Retiming was performed to satisfy the constraint on the number of registers on each wire while minimizing the total area of the components. Another paper [13] by Deokar et al. used a combination of clock skew and retiming to find a retiming solution which was guaranteed to be at most one gate delay larger than the optimal clock period. In their work, a clock skew solution corresponding to an optimal clock period was converted into a retiming solution. However, their current approach to perform this conversion considered only gate delays.

In this paper, we study the problem of retiming with both interconnect and gate delay. In our modeling, the delay of a wire is assumed to be directly proportional to its length. When a wire is short, the quadratic component of the wire delay is significantly smaller than its linear component. For a long wire, buffer insertion can be performed to break the wire into short segments. A simple experiment is conducted to illustrate the validity of this assumption and the result is shown in Figure 1. In this experiment, the Elmore delay model is used and the parameters are based on the $0.07\mu m$ technology. This graph shows the relationship between wire delay (y-axis) and wire length (x-axis). If the wire is shorter than 1.46 *mm*, the error of using a linear approximation is at most 5.48%. If the wire is longer than 1.46 *mm*, the delay can be reduced by inserting a buffer and the error resulted is even less.

We present two approaches in this paper both of which have polynomial time complexity. The first one is extended from the MILP approach in the paper [1] and can solve the problem optimally, i.e., relocating the registers to give the smallest possible clock period. The second one transforms the problem into a single-source longest paths problem and then applies a technique to reduce the size of the graph for longest path computation. It is an improvement over the first one in terms of practical applicability. It gives solutions very close to the optimal (0.13% more than the optimal on average) but in a much shorter runtime. Experimental results showed that a circuit with more than 22K gates and 32K wires could be retimed in 83.56 seconds by a PC with an 1.8GHz Intel Xeon processor. These retiming techniques will also find applications in flip-flop dropping in placement by estimating the best possible register positions to optimize the circuit performance.

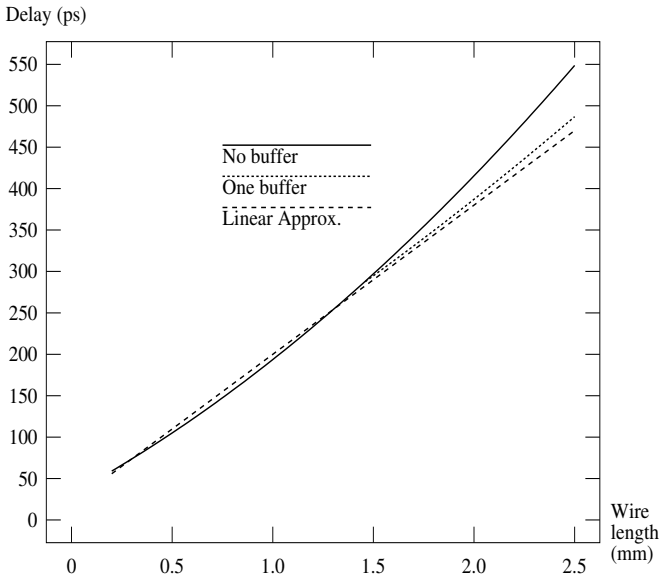


Figure 1: A simple experiment to illustrate the relationship between wire delay and wire length.

The original placement solution will be modified to relocate the registers according to the retiming solution. However the effect will be minor if the original solution is not very densely placed. This is a reasonable assumption today as area is not a major concern while routability and congestion are the important factors for circuit performance. Register relocations can then be done by making use of the empty space or by shifting the placed cells a little bit.

The remainder of this paper is organized as follows. We present the problem statement in Section 2. The optimal approach and the fast approach are presented in Section 3 and Section 4, respectively. Experimental results are shown and discussed in Section 5. A conclusion follows in Section 6.

2 Problem Formulation

A sequential circuit can be represented by a directed graph $G(V,E)$, where each node v corresponds to a combinational gate, and each directed edge e_{uv} represents a connection from the output of gate u to the input of gate v, through zero or more registers. Without loss of generality, we assume that G is strongly connected. If not, we can add a source node s and connect it to all primary inputs, add a target node t and connect all primary outputs to it, and connect t to s. Then the resulting graph is strongly connected. If we set the delay of s, t and all the added edges to zero, and set the number of registers on e_{ts} to one and that on the other added edges to zero, a retiming solution S of the modified graph will also be a valid retiming solution of the original graph as long as e_{ts} still has one register in S. Let w_{uv} be the number of registers of edge e_{uv}. Let d_{uv} be the interconnect delay of edge e_{uv} if all the registers are removed. Note that the delay of an interconnect segment is assumed to be proportional to the length of the segment. Let d_u be the gate delay of node u.

Traditionally, interconnect delay is ignored during retiming.

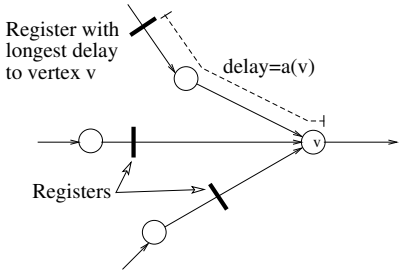


Figure 2: An example to illustrate the meaning of $a(v)$.

A retiming solution can be viewed as a labeling of the nodes $r : V \rightarrow Z$, where Z is the set of integers [1]. The retiming label $r(v)$ for a node v represents the number of registers moved from its outputs toward its inputs. After retiming, the number of registers $\hat{w}_{uv}$ on an edge e_{uv} is given by $\hat{w}_{uv} = r(v) + w_{uv} - r(u)$.

As interconnect delay is dominating in the VDSM technology, the exact position of each register will affect the clock period. A retiming solution should specify both the retiming label $r(v)$ for each node v and the exact positions of the $\hat{w}_{uv}$ registers on each edge e_{uv}. Retiming should be formulated as a problem of determining a feasible retiming solution, i.e., a solution in which the number of registers $\hat{w}_{uv}$ on each edge e_{uv} is non-negative, such that the clock period of the retimed circuit is minimized. In the following, we show how to check whether a particular clock period T can be achieved by a feasible retiming solution. The minimum achievable clock period T_{opt} can then be found by binary search.

3 An Optimal Approach

This approach is extended from the mixed integer linear programming (MILP) approach in [1]. In the original formulation, only gate delay is considered and there is thus no difference between having one or more than one registers on a wire. Their technique can be extended to solve the problem with both gate and interconnect delay optimally by modifying some of the constraint formulation. In order to formulate the problem as an MILP, for each gate v, we need to define a term $a(v)$ that represents the maximum arrival time at the output of gate v. An example to illustrate this definition is shown in Figure 2. We can then formulate the problem as the following MILP:

$$d_v \leq a(v) \quad \forall v \in V \quad (1)$$

$$a(v) \leq T \quad \forall v \in V \quad (2)$$

$$r(v) + w_{uv} - r(u) \geq 0 \quad \forall e_{uv} \in E \quad (3)$$

$$a(v) \geq a(u) + d_{uv} + d_v - T(r(v) + w_{uv} - r(u)) \quad \forall e_{uv} \in E \quad (4)$$

where T is the clock period that we want to check whether it is achievable. Since $a(v)$ is the longest delay to the output of gate v from a register connected directly to an input of v, this delay must be at least the delay of gate v, so $d_v \leq a(v)$ as stated in (1). Besides, this delay cannot exceed the clock period T as required in (2). Constraint (3) is needed for a feasible retiming solution. Constraint (4) is to ensure that enough registers are on each edge e_{uv} to achieve a clock cycle T. As the largest possible delay between two adjacent registers is T, the right-hand side of constraint (4) is reduced by T for each register on

edge e_{uv}. Note that this constraint also captures the scenario when there is no registers on edge e_{uv}. In that case, the arrival time at node u contributes directly to the arrival time at node v.

By introducing a variable $R(v)$ at each node v that is defined as $a(v)/T + r(v)$, the above set of constraints (1)–(4) can be rewritten as a set of difference constraints as follows:

$$R(v) - r(v) \geq \frac{d_v}{T} \quad \forall v \in V \tag{5}$$

$$R(v) - r(v) \leq 1 \quad \forall v \in V \tag{6}$$

$$r(u) - r(v) \leq w_{uv} \quad \forall e_{uv} \in E \tag{7}$$

$$R(v) - R(u) \geq \frac{d_{uv}}{T} + \frac{d_v}{T} - w_{uv} \quad \forall e_{uv} \in E \tag{8}$$

Notice that (5)–(8) is a set of difference constraints involving both integer and real variables. There are $|V|$ real variables $R(v)$, $|V|$ integer variables $r(v)$, and $2|V| + 2|E|$ constraints. This can be solved in polynomial time of $O(|V||E| \lg |V| + |V|^2 \lg^2 |V|)$ if Fibonacci heap is used as the data structure [23].

If the above set of constraints is solvable, the values of $r(v)$ and $a(v)$ for all $v \in V$ are known. We can then find the exact position of each register on a wire one by one as follows. For each edge e_{uv}, if there are registers retimed on it, i.e., $r(v) + w_{uv} - r(u) > 0$, the first register on this edge will be placed at a distance of delay $T - a(u)$ from the output of gate u. Other registers are then placed as far from each other as possible, i.e., at a distance of delay T from the previous one, until reaching the gate v. All the remaining registers on this edge are then placed right before v.

4 A Fast Near-Optimal Approach

In this approach, we first replace each gate by a wire of the same delay and then solve the problem with only interconnect delay optimally and efficiently. Those registers retimed "into" a gate are moved either to the input or the output wires of the gate. The exact positions of the registers on the wires are then determined by a linear program to minimize the clock period. The solution obtained by this approach is very close to the optimal on average as shown by the experimental results. In the following, we first show how the retiming problem with interconnect delay only can be solved optimally. Then we describe in details how gate delay can be handled simultaneously.

4.1 Retiming with Interconnect Delay Only

In this subsection, we assume $d_v = 0$ for all $v \in V$. We first show that the clock period feasibility problem can be reduced to a single-source longest paths problem. We then present a fast algorithm to solve the longest paths problem.

4.1.1 Reduction to Single-Source Longest Paths Problem

We solve the set of constraints (5)–(8) with the help of the following lemma.

Lemma 1 *Given $R(v)$ for all $v \in V$ satisfying constraint (8), we can obtain a solution to constraints (5)–(8) by setting $r(v) = \lfloor R(v) \rfloor$ for all $v \in V$.*

Proof: It is clear that $0 \leq R(v) - \lfloor R(v) \rfloor < 1$ for all $v \in V$. Therefore, (5) and (6) are satisfied. For any $e_{uv} \in E$,

$$\begin{aligned} r(u) - r(v) &\leq R(u) - r(v) \quad \text{as } r(u) \leq R(u) \\ &\leq (\frac{d_{uv}}{T} + R(u)) - r(v) \quad \text{as } \frac{d_{uv}}{T} > 0 \\ &\leq (w_{uv} + R(v)) - r(v) \quad \text{by constraint (8)} \\ &< w_{uv} + 1 \quad \text{as } R(v) - r(v) < 1 \end{aligned}$$

As $r(u) - r(v)$ is an integer, it must be less than or equal to w_{uv}. Hence, constraint (7) is also satisfied. □

Lemma 1 implies that we can first solve constraint (8) to find $R(v)$ and it is then easy to find $r(v)$ to satisfy the other three constraints. Notice that if $d_v \neq 0$ for some $v \in V$, Lemma 1 does not hold as constraint (5) is not satisfied. In other words, this idea cannot be applied to the retiming problem with both interconnect and gate delay discussed in Section 3.

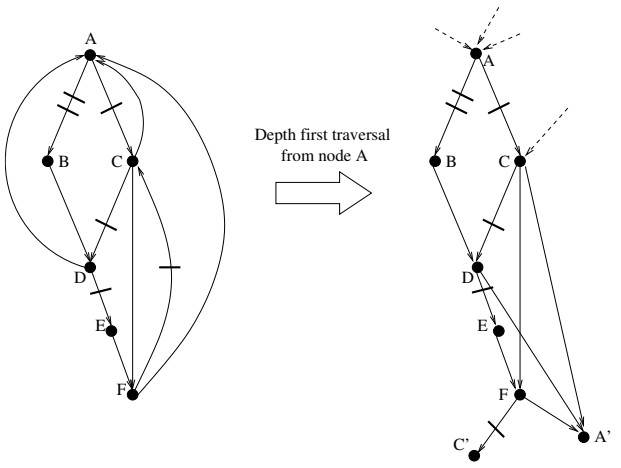
The problem of finding $R(v)$ for all $v \in V$ to satisfy constraint (8) can be viewed as a single-source longest paths problem on G with length l_{uv} equals $d_{uv}/T - w_{uv}$ for each $e_{uv} \in E$. As G is strongly connected, we can pick an arbitrary node as the source node s.[1] Note that edge lengths can be positive. If G has a positive cycle, the set of constraints has no solutions. It means that the clock period T is infeasible. The solution to this problem is presented in the following subsection.

4.1.2 Fast Single-Source Longest Paths Algorithm

The single-source longest paths problem in Section 4.1.1 can be solved by the Bellman-Ford algorithm [24]. The time complexity is $O(|V||E|)$, which is at least a factor of $\Theta(\lg |V|)$ faster than the optimal algorithm in Section 3. In practice, it is a factor of $\Theta(\lg^2 |V|)$ faster as $|E| = O(|V|)$. However, this algorithm may still be slow in practice. In this section, we present a single-source longest paths algorithm which is faster in practice. The basic idea is to reduce the size of G by compacting some paths into edges before the Bellman-Ford algorithm is applied. The details are given below.

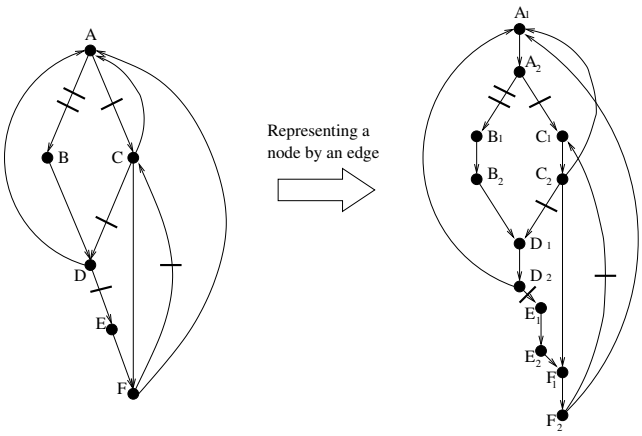
We first transform the graph $G(V,E)$ into a directed acyclic graph (DAG) $G'(V',E')$ by performing a depth-first traversal [24] starting from the source node s. The depth-first traversal defines a tree in G. Those non-tree edges running from a node u to an ancestor v of u are called back edges. If we point all incoming back edges of a node v to an extra node v', the resulting graph will be a DAG because every simple cycle in G involves exactly one back edge. Formally, we use E_b to denote the set of back edges and V_b to denote the set of nodes with an incoming back edge. For each node v in V_b, we introduce an extra node v'. The back edge e_{uv} is removed from the graph and the edge $e_{uv'}$ is added. The resulting DAG is $G'(V',E')$ where $V' = V \cup \{v' | v \in V_b\}$ and $E' = (E - E_b) \cup \{e_{u,v'} | e_{u,v} \in E_b\}$. We set the length $l_{uv'}$ of the edge $e_{uv'}$ to l_{uv}. To illustrate the transformation, consider the graph G in Figure 3(a) with source node A. Suppose the depth-first traversal visits the nodes in the order $ACDEFB$. Then $E_b = \{e_{DA}, e_{CA}, e_{FC}, e_{FA}\}$ and $V_b = \{A, C\}$. We introduce two extra nodes A' and C', and replace the four edges e_{CA}, e_{DA}, e_{FA} and e_{FC} with the edges

[1] If the original circuit is not strongly connected, a source node s has already been added.



(a) Original graph G (b) Directed acyclic graph G'

Figure 3: An example to illustrate the transformation to a DAG.



(a) Original graph G (b) Transformed graph $\tilde{G}$

Figure 4: Representation of gates by wires.

$e_{CA'}$, $e_{DA'}$, $e_{FA'}$ and $e_{FC'}$, respectively. The resulting DAG is shown in Figure 3(b).

We then construct a graph H with node set V_b. The edge set E_H contains an edge e_{uv} for $u, v \in V_b$ if there exists a path in G with either no back edge or one back edge at the end from u to v. The length l_{uv}^H of the edge e_{uv} is the longest path distance among those paths. Note that the longest path distance in G with no back edge (respectively, with one back edge at the end of the path) from u to v equals the longest path distance in G' from u to v (respectively, from u to v'). Hence l_{uv}^H for all $u, v \in V_b$ can be computed by solving $|V_b|$ single-source longest paths problems in G' for different source nodes in V_b. As G' is a DAG, each single-source longest paths problem can be solved in linear time by visiting the nodes in topological order. The time complexity to construct H is therefore $O(|V_b||E|)$.

It is obvious that every path in H corresponds to at least one path in G of the same length. Therefore if H contains a positive cycle, G will also contain a positive cycle. On the other hand, if G contains a positive cycle, the cycle can be broken up into a set of paths $p_1, p_2, \ldots, p_k$ such that both endpoints of each path p_i are in V_b. Notice that each path p_i corresponds to an edge in H of at least the same length. So H must also contain a positive cycle. Therefore we can solve the positive cycle detection problem in H instead of in G. If H has no positive cycles, $R(v)$ for all $v \in V_b$ can be found from H. $R(v)$ for all $v \in V - V_b$ can then be found in linear time by propagating $R(v)$ for all $v \in V_b$ through G' in topological order.

4.1.3 The Retiming Algorithm and Time Complexity

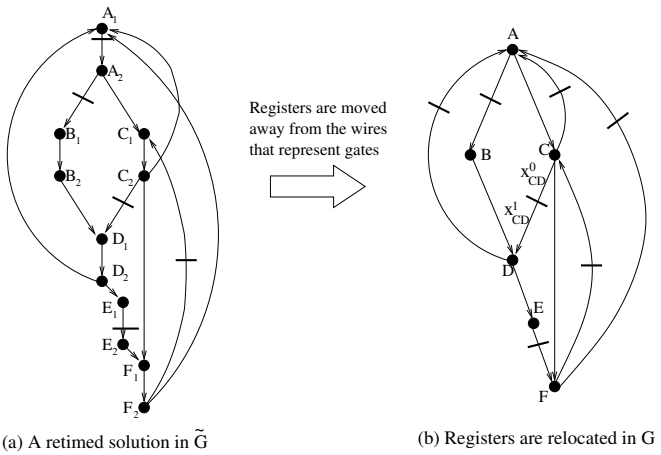
The complete retiming algorithm *I-Retiming()* is summarized below. The most time consuming steps are step 7 and step 8 inside the binary search loop. Step 7 can be done in $O(|V_b||E|)$ time as discussed above. Step 8 can be done in $O(|V_b||E_H|)$ time by the Bellman-Ford algorithm. As V_b contains much fewer nodes than V and E_H usually contains comparable or fewer edges than E, this technique is usually much more efficient than applying the Bellman-Ford algorithm to G directly. The total time complexity is $O(|V_b| \max\{|E|, |E_H|\} \lg \frac{K}{\varepsilon T_{opt}})$, where ε is the error bound for the binary search, K is the difference between the upper and lower bounds of the clock period initially, and T_{opt} is the optimal clock period.

Algorithm *I-Retiming()*
Input: A sequential circuit C with interconnect delay only
Output: An optimally retimed circuit of C
1. Build graph $G(V,E)$ from C
2. Build DAG G' by DFS(G)
3. C_{up} = a feasible clock, C_{low} = an infeasible clock
4. Do
5. $\quad T = (C_{up} + C_{low})/2$
6. $\quad$ Update edge lengths of G' according to T
7. $\quad$ Build graph $H(V_b, E_H)$ with $E_H = \{e_{uv} | u \in anc(v) \cup anc(v')\}$ by finding single-source longest paths in G'
8. $\quad$ If H does not have any positive cycle then
9. $\quad\quad C_{up} = T$
10. $\quad$ Else
11. $\quad\quad C_{low} = T$
12. while $(C_{up} - C_{low})/C_{up} > \varepsilon$
13. $T = C_{up}$ // C_{up} is always a feasible clock period
14. Compute $R(v)$ and $r(v)$ for each node $v \in V$
15. Compute the exact position of each register on a wire

4.2 Retiming with Interconnect and Gate Delay

In this section, we discuss how to consider interconnect and gate delay simultaneously based on the above algorithm for interconnect delay only. To consider gate delay, we first represent a gate v with delay d_v by a wire $e_{v_1 v_2}$ with delay $d_{v_1 v_2} = d_v$. This transformation for the circuit in Figure 3(a) is shown in Figure 4(b). We can then obtain an optimal retiming on this transformed circuit $\tilde{G}$ using the algorithm in Section 4.1. However the retiming solution obtained on $\tilde{G}$ may not be feasible for the original circuit G because some registers may be retimed into a wire that represents a gate. Therefore, we need to perform a post-processing step to get back a feasible retiming solution for G from the optimal retiming solution for $\tilde{G}$. This is done by linear programming.

First of all, we move the registers in a gate either backward to the input wires or forward to the output wires of the gate, depending on which direction has a shorter distance. An example showing the relocation of registers is given in Figure 5. After this relocation step, the number of registers $\hat{w}_{uv}$ on each edge e_{uv} is fixed. A linear program is used to determine the



(a) A retimed solution in $\tilde{G}$ (b) Registers are relocated in G

Figure 5: Relocation of registers retimed into a gate.

exact positions of the registers on the edges. The objective of the linear program is to minimize the clock period T subject to the constraints in register count on each edge. In the following, we use x^k_{uv} to denote the delay from the k^{th} register to the $k+1^{st}$ register of the wire from node u to node v in G for $k = 0, 1, \ldots, \hat{w}_{uv}$. Notice that when $\hat{w}_{uv} = 0$, x^0_{uv} is the delay of the whole wire, and when $k = 0$ and $k = \hat{w}_{uv} > 0$, x^k_{uv} are the delays of the wire from node u to the first register and from the last register to node v, respectively. The linear program is formulated as follows:

$$
\begin{array}{lll}
\text{Minimize} & \text{T} & \\
\text{Subject to} & \sum_{k=0}^{\hat{w}_{uv}} x^k_{uv} = d_{uv} \quad \forall e_{uv} \in E & \text{(A)} \\
& x^{\hat{w}_{uv}}_{uv} + d_v \le a(v) \quad \forall e_{uv} \in E \text{ s.t. } \hat{w}_{uv} > 0 & \text{(B)} \\
& a(u) + x^0_{uv} \le T \quad \forall e_{uv} \in E \text{ s.t. } \hat{w}_{uv} > 0 & \text{(C)} \\
& a(u) + d_{uv} \le a(v) \quad \forall e_{uv} \in E \text{ s.t. } \hat{w}_{uv} = 0 & \text{(D)}
\end{array}
$$

For the circuit in Figure 5(b), example constraints are $x^0_{CD} + x^1_{CD} = d_{CD}$ for type (A), $x^1_{CD} + d_D \le a(D)$ for type (B), $a(C) + x^0_{CD} \le T$ for type (C), and $a(B) + d_{BD} \le a(D)$ for type (D). We can solve this linear program to obtain the best possible clock period T^* under the register count constraint on each edge. The overall algorithm *IG-Retiming()* to handle both interconnect and gate delay is summarized as follows:

Algorithm *IG-Retiming()*
Input: A sequential circuit C with both interconnect and gate delay
Output: A retimed circuit of C
1. Build graph G from C
2. Build $\tilde{G}$ by replacing each gate in G by a wire of the same delay
3. Solve the retiming problem of $\tilde{G}$ by *I-Retiming()*
4. Move registers away from wires that represent gates
5. Set up a linear program based on the register count on each edge
6. Solve the linear program to obtain a feasible retiming solution and the smallest possible clock period T^*

5 Experimental Results

We implemented the two approaches in a 1.8GHz Intel Xeon PC with 512 KB cache and 512 MB RAM. We tested them with circuits from the ISCAS89 benchmark suite. In our experiments, we implement the circuits in a 0.25 μm process. We layout the circuits by Silicon Ensemble. Wire delays are then extracted according to the layout. In our current implementation, the lower and upper bounds of the binary search are set to 0 and 100ns respectively. In the near-optimal approach, we perform the procedure *I-Retiming()* with an error bound of 1%. After assigning the registers retimed into a gate to the appropriate wires, a linear program is set up to relocate the registers on the wires to get the smallest possible clock period T^*. In the optimal approach, binary search is performed until an error bound of 0.01% is obtained. We call the resulting clock period T_{opt}. Notice that we do not need to obtain a very accurate result from *I-Retiming()* because the solution is optimized by the linear program afterwards. On average, the number of binary search iterations is 9.6 for the near-optimal approach and 16.5 for the optimal approach.

The results are shown in Table 1. The second and third columns give the number of nodes and the number of edges in the graph G, respectively. Notice that all circuits are not strongly connected. The number of nodes and edges listed are those after the addition of the source node, the target node, and the associated edges. The fourth and fifth columns show the number of nodes and the number of edges in the reduced graph H, respectively. These two values are dependent on the node chosen as the root in the depth-first traversal. In our current implementation, we always pick the additional node s as the root. We notice that using other nodes as the root does not change the result significantly. The speedup of the Bellman-Ford algorithm by the graph reduction approach in Section 4.1.2 is $(|V||E|)/(|V_b||E_H|)$, which is given in the sixth column. The graph reduction approach is faster in all circuits except s38584. On average, it is faster by 30.61 times. However, the speedup is less (may even be less than one) for larger circuits. The reason is that $|E_H|$ is roughly quadratic in $|V_b|$. For the circuits in Table 1, the ratio of $|E_H|$ to $|V_b|^2$ is from 0.11 to 0.86 with an average of 0.41. Therefore, the graph reduction approach may not be useful for large circuits. We can avoid a slowdown of the Bellman-Ford algorithm by determining whether to use G or H based on the ratio $(|V||E|)/(|V_b||E_H|)$. $|V_b|$ and $|E_H|$ can be found in $O(|V_b||E|)$ time. Moreover, we only need to perform this checking once for each circuit. Hence, the runtime overhead is insignificant compared with the total runtime.

The seventh, eighth, and ninth columns show the runtime of the *I-Retiming()* procedure, the time taken to solve the linear program, and the total runtime, respectively. The tenth column shows the runtime for the optimal approach. We can see that the near-optimal approach is much more efficient than the optimal approach (especially for large circuits). The eleventh and twelfth columns show the clock period T^* and T_{opt} obtained by the near-optimal approach and the optimal approach, respectively. The last column is the percentage increase of T^* over T_{opt}. The clock period produced by the near-optimal approach is only 0.13% more than that by the optimal approach on average. The optimal clock period is found in seven out of thirteen circuits.

6 Conclusion

We have presented two elegant approaches to perform retiming on sequential circuits with both interconnect and gate delay. This is a pioneer work in solving this problem as far as we

Circuit	No. of Nodes in V	No. of Edges in E	No. of Nodes in V_b	No. of Edges in E_H	$\frac{\|V\|\|E\|}{\|V_b\|\|E_H\|}$	CPU Time				Clock Period		
						I-Retiming + LP = *IG-Retiming*			Optimal	T^*	T_{opt}	$\frac{T^*-T_{opt}}{T_{opt}}$
						(sec)	(sec)	(sec)	(sec)	(ns)	(ns)	(%)
s1488	655	1405	27	627	54.36	0.09	0.19	0.28	5.62	18.85	18.82	0.16
s1494	649	1411	30	749	40.75	0.09	0.16	0.25	4.37	20.78	20.78	0.00
s3271	1574	2707	112	3360	11.32	0.38	0.71	1.09	33.70	10.24	10.24	0.00
s3330	1791	2890	56	1200	77.02	0.13	0.37	0.50	43.14	27.05	27.05	0.00
s3384	1687	2782	98	2041	23.46	0.16	0.58	0.74	25.19	24.21	24.16	0.21
s4863	2344	4093	154	20413	3.05	2.13	0.99	3.12	87.75	23.58	23.58	0.00
s5378	2781	4261	66	2554	70.30	0.55	0.61	1.16	138.68	27.27	27.25	0.07
s6669	3082	5399	67	1876	132.38	0.36	1.55	1.91	177.59	23.07	22.96	1.00
s9234	5599	8005	325	26570	5.19	2.69	1.39	4.08	512.86	42.73	42.73	0.00
s13207	7953	11302	550	44825	3.65	6.45	1.66	8.11	1161.07	72.34	72.34	0.00
s15850	9774	13794	603	100738	2.22	21.42	2.60	24.02	1545.59	67.82	67.82	0.00
s35932	16067	28590	884	163945	3.17	54.59	6.66	61.25	8644.27	29.59	29.54	0.17
s38417	22181	32135	1657	308790	1.39	72.64	10.92	83.56	7680.79	36.53	36.52	0.03
s38584	19255	33010	1924	1115868	0.30	433.82	11.81	445.63	> 15000	94.26		

Table 1: The runtime of the algorithms and the clock periods obtained.

know. Most traditional retiming algorithms have neglected interconnect delay. Our first approach is extended from the MILP approach in the paper [1] and can solve the problem optimally. Our second approach is an improvement over the first one in terms of practical applicability. The main idea is to transform the problem into a single-source longest paths problem in a reduced graph. We have implemented both algorithms, and compared their performance on ISCAS89 benchmark circuits. Experimental results show that the second approach gives solutions that are only 0.13% larger than the optimal on average but in a much shorter runtime.

References

[1] Charles E. Leiserson and James B. Saxe. Retiming Synchronous Circuitry. *Algorithmica*, 6:5–35, 1991.

[2] C. V. Schimpfle, Sven Simon, and Josef A. Nossek. Optimal Placement of Registers in Data Paths for Low Power Design. In *Proc. ISCAS*, pages 2160–2163, 1997.

[3] J. Monteiro, S. Devadas, and A. Ghosh. Retiming Sequential Circuits for Low Power. In *Proc. ICCAD*, pages 398–402, 1993.

[4] A. El-Maleh, T. E. Marchok, J. Rajski, and W. Maly. Behavior and Testability Preservation under the Retiming Transformation. *IEEE TCAD*, 16:528–542, 1997.

[5] S. Dey and S. Chakradhar. Retiming Sequential Circuits to Enhance Testability. In *Proc. IEEE VLSI Test Symposium*, pages 28–33, 1994.

[6] Rajeev K. Ranjan, Vigyan Singhal, Fabio Somenzi, and Robert K. Brayton. On the Optimization Power of Retiming and Resynthesis Transformation. In *Proc. ICCAD*, pages 402–407, 1998.

[7] Peichen Pan, Arvind K. Karandikar, and C. L. Liu. Optimal Clock Period Clustering for Sequential Circuits with Retiming. *IEEE TCAD*, 17(6):489–498, 1998.

[8] Jason Cong, Honching Li, and Chang Wu. Simultaneous Circuit Partitioning/Clustering with Retiming for Performance Optimization. In *Proc. DAC*, pages 460–465, 1999.

[9] Jason Cong, Sung Kyu Lim, and Chang Wu. Performance Driven Multi-level and Multiway Partitioning with Retiming. In *Proc. DAC*, pages 274–279, 2000.

[10] Jason Cong and Sung Kyu Lim. Physical Planning with Retiming. In *Proc. ICCAD*, pages 2–7, 2000.

[11] N. Shenoy and R. Rudell. Efficient Implementation of Retiming. In *Proc. ICCAD*, pages 226–233, 1994.

[12] N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli. Retiming of Circuits with Single Phase Transparent Latches. In *Proc. ICCAD*, pages 86–89, 1991.

[13] Rahul B. Deokar and Sachin S. Sapatnekar. A Fresh Look at Retiming via Clock Skew Optimization. In *Proc. DAC*, pages 310–315, 1995.

[14] Marios C. Papaefthymiou. Asymptotically Efficient Retiming under Setup and Hold Constraints. In *Proc. ICCAD*, pages 396–401, 1998.

[15] H. J. Touati and R. K. Brayton. Computing the Initial States of Retimed Circuits. *IEEE TCAD*, 12:157–162, 1993.

[16] I. Karkowski and R.H.J.M. Otten. Retiming Synchronous Circuitry with Imprecise Delay. In *Proc. DAC*, pages 322–326, 1995.

[17] Vigyan Singhal, Sharad Malik, and Robert K. Brayton. The Case for Retiming with Explicit Reset Circuitry. In *Proc. ICCAD*, pages 618–625, 1996.

[18] N. Maheshwari and S. S. Sapatnekar. An Improved Algorithm for Minimum-area Retiming. In *Proc. DAC*, pages 2–7, 1997.

[19] T. Soyata and E. G. Friedmann. Retiming with nonzero clock skew, variable register and interconnect delay. In *Proc. ICCAD*, pages 234–241, 1994.

[20] Kumar N. Lalgudi and Marios C. Papaefthymiou. DELAY: An Efficient Tool for Retiming with Realistic Delay Modeling. In *Proc. DAC*, pages 304–309, 1995.

[21] Tzu-Chieh Tien, Hsiao-Pin Su, and Yu-Wen Tsay. Integrating Logic Retiming and Register Placement. In *Proc. ICCAD*, pages 136–139, 1998.

[22] Abdallah Tabbara, Robert K. Brayton, and A. Richard Newton. Retiming for DSM with Area-Delay Trade-offs and Delay Constraints. In *Proc. DAC*, pages 725–730, 1999.

[23] C. E. Leiserson and James B. Saxe. A Mixed-Integer Programming Problem Which is Efficiently Solvable. *Journal of Algorithms*, 9:114–128, 1988.

[24] Thomas H. Cormen and Charles E. Leiserson and Ronald L. Rivest. *Introduction to Algorithms*. McGraw Hill, eighth edition, 1992.