Clustering for Processing Rate Optimization

Chuan Lin, Jia Wang, and Hai Zhou, *Senior Member, IEEE*

***Abstract*—Clustering (or partitioning) is a crucial step between logic synthesis and physical design in the layout of a large scale design. A design verified at the logic synthesis level may have timing closure problems at post-layout stages due to the emergence of multiple-clock-period interconnects. Consequently, a tradeoff between clock frequency and throughput may be needed to meet the design requirements. In this paper, we find that the processing rate, defined as the product of frequency and throughput, of a sequential system is upper bounded by the reciprocal of its maximum cycle ratio, which is only dependent on the clustering. We formulate the problem of processing rate optimization as seeking an optimal clustering with the minimal maximum-cycle-ratio in a general graph, and present an iterative algorithm to solve it. Experimental results validate the efficiency of our algorithm.**

***Index Terms*—Algorithms, circuit optimization, clustering methods, design automation, integrated circuit interconnections.**

I. Introduction

CIRCUIT clustering (or partitioning) is often employed between logic synthesis and physical design to decompose a large circuit into parts. Each part will be implemented as a separate cluster that satisfies certain design constraints, such as the size of a cluster. Clustering helps to provide the first-order information about interconnect delays as it classifies interconnects into two categories: intra-cluster ones are local interconnects due to their spatial proximity while inter-cluster ones may become global interconnects after floorplan/placement and routing (also known as circuit layout).

Due to aggressive technology scaling and increasing operating frequencies, interconnect delay has become the main performance limiting factor in large scale designs. Industry data shows that even with interconnect optimization techniques such as buffer insertion, the delay of a global interconnect may still be longer than one clock period, and multiple clock periods are generally required to communicate such a global signal. Since global interconnects are not visible at logic synthesis when the functionality of the implementation is the major concern, a design that is correct at the logic synthesis level may have timing closure problems after layout due to the emergence of multiple-clock-period interconnects.

This gap has motivated recent research to tackle the problem from different aspects of view. Some of them resort to retiming [15], which is a traditional sequential optimization technique that moves flip-flops within a circuit without destroying its functionality. It was used in [5], [16], [17], [21], and [22] to pipeline global interconnects so as to reduce the clock period. Although retiming helps to relieve the criticality of global interconnects, there is a lower bound of the clock periods that can be achieved because retiming cannot change the latency of either a (topological) cycle or an input-to-output path in the circuit. In case the lower bound does not meet the frequency requirement, redesign and resynthesis may have to be carried out.

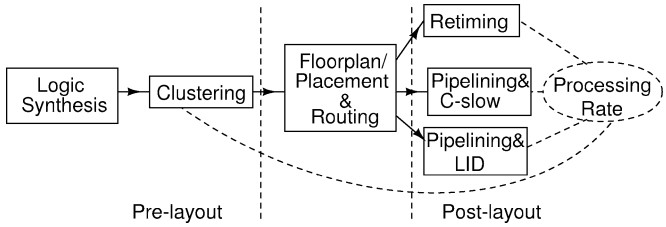


Fig. 1. Logical and physical design flow.

One way to avoid redesign is to insert extra wire pipelining units like flip-flops to pipeline long interconnects, as done within Intel [6] and IBM [14]. It can be shown that if the period lower bound is determined by an input-to-output path, pipelining can reduce the lower bound without affecting the functionality. However, if the period lower bound is given by a cycle, inserting extra flip-flops in it will change its functionality.

C-slow transformation [15] is a technique that slows down the input issue rate[1] of the circuit to accommodate higher frequencies. It was, thus, used in [19] to retain the functionality when extra flip-flops were inserted in cycles. The slowdown C is dictated by the slowest cycle in the circuit where the ratio between the extra flip-flops and the original flip-flops is the maximum. Extra flip-flops are inserted in other cycles to match the slowdown. As a result, throughput is sacrificed (becomes $1/C$) to meet the frequency requirement.

Instead of slowing down the throughput uniformly over the whole circuit, latency insensitive design (LID) [1], [2], on the other hand, employs a protocol that slows down the throughput of a part of the circuit only when it is needed. As a result, LID can guarantee minimal throughput reduction while satisfying the frequency requirement.

We show in Section II that the aforementioned three approaches (retiming, pipelining with C-slow, and pipelining with LID) can be unified under the same objective function of maximizing the *processing rate*, defined as the product of frequency and throughput, as illustrated in Fig. 1. In addition, the processing rate of a sequential system is upper bounded by the reciprocal of the maximum cycle ratio of the system, which

Manuscript received December 30, 2005; revised March 24, 2006. This work was supported by the National Science Foundation under Grant CCR-0238484.

C. Lin is with the Logic Synthesis Group, Magma Design Automation Inc., Santa Clara, CA 95054 USA.

J. Wang and H. Zhou are with the Electrical Engineering and Computer Science Department, Northwestern University, Evanston, IL 60208 USA (e-mail: haizhou@eecs.northwestern.edu).

Digital Object Identifier 10.1109/TVLSI.2006.886399

[1]The issue rate is defined as the number of clock periods between successive input changes. An issue rate of 1 indicates that the inputs can change every clock period.

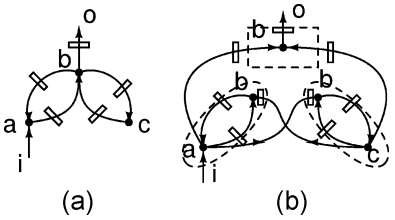


Fig. 2. (a) Example circuit. (b) Clustering with three replicas of gate b.

is only dependent on the clustering. Therefore, we propose an optimal algorithm that finds a clustering with the minimal maximum-cycle-ratio under the single-value inter-cluster delay model.

The rest of this paper is organized as follows. Section II presents the problem formulation. Two previous works are reviewed in Section III. Section IV defines the notations and constraints used in this paper. Following an overview in Section V, our algorithm is elaborated in Section VI and VII. Section VIII presents the speed-up techniques. Section IX reviews the techniques in [20] for cluster and replication reduction. We present some experimental results in Section X. Conclusions are given in Section XI.

II. Problem Formulation

We consider clustering subject to a size limit for clusters. More specifically, each gate has a specified size, as well as each interconnect. We require that the size of each cluster, defined as the sum of the sizes of the gates and the interconnects in the cluster, should be no larger than a given constant A. Replication of gates is allowed, i.e., a gate may be assigned to more than one cluster in the layout. When a gate is replicated, its incident interconnects are also replicated so that the clustered circuit is logically equivalent to the original circuit. Fig. 2 (taken from [20]) shows an example of gate replication in a clustering.

Given a particular clustering c, we treat the replicas of gates and the original ones distinctly and denote them all as V_c. We use E_c to denote the set of interconnects among V_c. The clustered circuit is represented as $G_c = (V_c, E_c)$. In order for the circuit to operate at a specified clock period λ, additional wire-pipelining flip-flops are inserted. For all cycle o_c in G_c, let $d(o_c)$ denote the cycle delay, $w(o_c)$ and $w_\lambda(o_c)$ denote the number of flip-flops in o_c before and after additional pipelining flip-flops are inserted, respectively. Assuming $w(o_c) > 0$, the cycle ratio of o_c is defined as $\phi(o_c) = d(o_c)/w(o_c)$. Note that $\phi(o_c)$ is defined using $w(o_c)$, not $w_\lambda(o_c)$. The *maximum cycle ratio* over all the cycles in G_c is denoted as $\phi_c = \max_{o_c \in G_c} \phi(o_c)$.

We define *processing rate* as follows.

Definition 1: For a sequential system, processing rate is defined as the length of processed input sequence per unit time. In particular, it is the product of frequency and throughput in a synchronous system.

The larger the processing rate, the better the sequential system. Given the previous definition, the approach of retiming actually maximizes the processing rate by minimizing the period while keeping the throughput. It is interesting to notice that the approach of pipelining with C-slow transformation also maximizes the processing rate for a specified period by computing the least slowdown of the issue rate, which is transformed into throughput reduction. As an alternative, LID helps the clustered circuit reach the maximum throughput for a specified period. Therefore, all three approaches can be unified under the same objective function of maximizing the processing rate.

It was shown in [3] and [4] that the maximum throughput ρ_λ of a LID for a specified period λ can be computed as

$$\rho_\lambda = \min_{\text{cycle} o_c \in G_c} \frac{w(o_c)}{w_\lambda(o_c)}.$$

On the other hand, the fact that the circuit can operate at the specified period λ after the insertion of additional flip-flops implies that $w_\lambda(o_c)\lambda \geq d(o_c)$, i.e., $1/w_\lambda(o_c) \leq 1/d(o_c)/\lambda$, $\forall$cycle $o_c \in G_c$. Substitute this into the formula of ρ_λ to get

$$\rho_\lambda \leq \min_{o_c \in G_c} \frac{w(o_c)}{d(o_c)/\lambda} = \min_{o_c \in G_c} \frac{\lambda}{\phi(o_c)} = \frac{\lambda}{\phi_c}.$$

It follows that the maximum processing rate of a LID is upper bounded by $1/\phi_c$ since

$$\text{max processing rate} = \frac{1}{\lambda} \cdot \rho_\lambda \leq \frac{1}{\lambda} \cdot \frac{\lambda}{\phi_c} = \frac{1}{\phi_c}.$$

It is also an upper bound of the maximum processing rate obtained by the approach of retiming, as shown in [22]. In other words, all three approaches share the same upper bound of their common objective.

To maximize the processing rate, one can either maximize the upper bound or try to achieve the upper bound. They are equally important. However, since achieving the upper bound requires further knowledge on physical design, such as buffer and flip-flop allowable regions [9], [22], while the upper bound itself is only dependent on the maximum cycle ratio of the clustered circuit, we will consider how to optimally cluster the circuit such that the upper bound is maximized, or equivalently, the maximum cycle ratio is minimized.

In order to compute the maximum cycle ratio, we need to know how to compute the delay of a cycle during clustering. Although local interconnect delays can be obtained using some delay models at synthesis, the delays of global interconnects are not available until layout. Therefore, during clustering, we assume that each global interconnect induces an extra constant delay D, as proposed in [20]. More specifically, if an interconnect (u, v) with delay $d(u, v)$ is assigned to be inter-cluster, then its delay becomes $d(u, v) + D$.

The single-value inter-cluster delay model is the best approximation to distinguish potential global interconnects from local ones. In floorplanning, critical global interconnects are made short by placing the relevant modules closer. On the other hand, the values of cluster size A and inter-cluster delay D can be chosen deliberately to make this model practical. The intra-cluster interconnects will be very long if large A is selected. On the other hand, A shall not be too small otherwise there will be too many clusters and the inter-cluster delays will be similar to the intra-cluster delays after floorplanning. By carefully choosing D, the single-value model can fulfill the need of integrating inter-cluster delay information in clustering.

Since we want to minimize the maximum cycle ratio, the path delays from primary inputs (PIs) to primary outputs (POs) can be ignored since they can be mitigated by pipelining. This motivates us to formulate the problem in a strongly connected graph.

Problem 1 (Optimal Clustering Problem): Given a directed, strongly connected graph $G = (V, E)$, where each vertex $v \in V$ has a delay $d(v)$ and a specified size, and each edge $(u, v) \in E$ has a delay $d(u, v)$, a specified size, and a weight $w(u, v)$ (representing the number of flip-flops on it), find a clustering of vertices with possible vertex replication such that: 1) the size of each cluster is no larger than a given constant A; 2) each global interconnect induces an extra constant delay D; and 3) the maximum cycle ratio of the clustered circuit is minimized.

We assume that all delays are integral[2] and, thus, all cycle ratios are rational. In addition, we assume that each gate has unit size and the size of each interconnect is zero. Our proposed algorithm can be easily extended to handle various size scenarios.

III. Previous Work

Pan *et al.* [20] proposed to optimally cluster a sequential circuit such that the lower bound of the period of the clustered circuit was minimized with retiming. However, the period lower bound may not come from a cycle ratio. In addition, their algorithm needs to start from PIs, thus, cannot be used to solve our problem in a strongly connected graph. Applying their algorithm with arbitrary PI selection may lead to a suboptimal solution. Consider an example circuit consisting of two gates u and v, connected in a ring with $w(u, v) = 0$, $w(v, u) = 4$, $D = 10$, $A = 1$, and zero gate and interconnect delays. If we pick u as PI and v as PO, their algorithm will return 10, but the optimal solution is 5. In this sense, they solved a different problem, even though it looks similar to ours.

Their problem was solved by binary search, using a test for feasibility as a subroutine. For each target period, they used a procedure called *labeling computation* to check the feasibility. The procedure starts with label assignment 0 for PIs and $-\infty$ for the other vertices, and repeatedly increases the label values until they all converge or the label value of some PO exceeds the target period, for which the target period is considered infeasible. For each vertex, the amount of increase in its label is computed using another binary search that basically selects the minimum from a candidate set. Because of the nested binary searches, their algorithm is relatively slow. In addition, the algorithm requires $O(|V|^2)$ space to store a precomputed all-pair longest-path matrix, which is impractical for large designs. Cong *et al.* [10] improved the algorithm by tightening the candidate set to speed up the labeling computation, and by reducing the space complexity to linear dependency. But the improved algorithm still needs the nested binary searches.

Besides the difference in problem formulation, our algorithm differs from theirs in two algorithmic aspects. First, our algorithm focuses on cycles, thus, can work on any general graph. Second, no binary search is employed in our algorithm. As a result, our algorithm is efficient and essentially incremental. Like [10], our algorithm does not need precomputed information on paths either.

[2]This assumption is not really restrictive in practice because computer works with rational numbers which we can convert to integers by multiplying by a suitably large number.

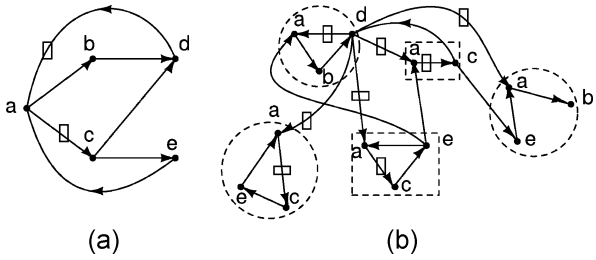


Fig. 3. Example of clustering representation.

Except for these differences, [20] revealed some important results on clustering, which we summarize in the following to simplify our notations.

- Each cluster has only one output, which is called the *root* of the cluster. If there is a cluster with more than one output, we can replicate the cluster enough times so that each copy of the cluster has only one output.
- For each vertex in V, there is exactly one cluster rooted at it and no cluster rooted at its replicas. Its arrival time (defined in Section IV) is no larger than the arrival times of its replicas.
- If $u \in V$ is an input of the cluster rooted at $v \in V$, then the cluster rooted at v must not contain a replica of u.

IV. Notations and Constraints

First of all, we define notations with respect to G and G_c (the clustered circuit of a particular clustering c), respectively.

We use p to denote a path in G. Let $w(p)$ be the number of flip-flops on p, which is the sum of the weights of p's constituent edges. Let $d(p)$ be the delay along p, which is the sum of the delays of p's constituent edges and vertices, except for $d(u)$. When a path actually forms a cycle o, $w(o)$ includes the weight of each edge in the cycle only once. Similarly, $d(o)$ includes the delay of each edge and vertex in the cycle only once. We assume, in this paper, that $w(o) > 0$ for all cycle $o \in G$.

Each of the previous notations is appended with a subscript c when it is referred to with respect to G_c. More specifically, a path in G_c is denoted as p_c with $w(p_c)$ flip-flops and $d(p_c)$ delays. Note that the delay of an inter-cluster edge $(u, v) \in G_c$ is $d(u, v) + D$. A cycle in G_c is denoted as o_c with $w(o_c)$ flip-flops and $d(o_c)$ delays. We have $w(o_c) > 0$ for all cycle $o_c \in G_c$. We use $\phi(o_c)$ to denote the cycle ratio of o_c, and ϕ_c to denote the maximum cycle ratio of G_c.

Since we only need to consider clusters rooted at the vertices in V, exactly one for each vertex, we use c_v to refer to the set of vertices that are included in the cluster rooted at $v \in V$. Let $i_v \subset V$ be the set of inputs of c_v. In the remainder of this paper, when we say $u \in c_v$ $(u \neq v)$, we mean that the cluster rooted at $v \in V$ contains a replica of $u \in V$. For example, Fig. 3(a) shows a circuit before clustering. There are five vertices (a–e) and seven edges. Fig. 3(b) illustrates a clustering of the circuit with size limit $A = 3$, where dashed circles represent clusters. For each cluster, the vertex whose index is outside the cluster indicates the root. For example, c_a contains replicas of vertices

c and e with the input set $i_a = \{d\}$. Note that, a is the output of c_a, even though c_a has no outgoing edges.

We use a label $t : V \to \mathbb{R}$ to denote the arrival time of the vertex. To ease the presentation, we will extend the domain of t to V_c to represent the arrival times of the replicas of the vertices. Based on this, a clustering that satisfies the cluster size requirement and has a maximum cycle ratio no larger than a given rational value ϕ can be characterized as follows:

$$t(v) \geq 0 \qquad \forall v \in V \tag{1}$$

$$t(v) \geq t(u) + d(u,v) + d(v) - w(u,v)\phi \quad \forall \text{ intra-cluster}(u,v) \in E_c \tag{2}$$

$$t(v) \geq t(u) + d(u,v) + D + d(v) - w(u,v)\phi, \quad \forall \text{ inter-cluster}(u,v) \in E_c \tag{3}$$

$$|c_v| \leq A \qquad \forall v \in V \tag{4}$$

where (1)–(3) guarantee that the arrival times are all achievable, and (4) is the cluster size requirement.

Following the convention, $(u,v) \in E_c$ is a *critical edge* under ϕ iff it is intra-cluster with $t(v) = t(u) + d(u,v) + d(v) - w(u,v)\phi$, or it is inter-cluster with $t(v) = t(u) + d(u,v) + D + d(v) - w(u,v)\phi$. A *critical path* under ϕ refers to a path whose constituent edges are all critical under ϕ. Vertex $u \in V$ is a *critical input* of c_v under ϕ iff $u \in i_v$ and v can be reached by u through a critical path $p = u \to x \rightsquigarrow v$ under ϕ where the subpath $x \rightsquigarrow v$ is in c_v. When a critical path actually forms a cycle, it is then called a *critical cycle*. Cycle o_c is critical under ϕ iff $d(o_c) = w(o_c)\phi$.

A *legal clustering* must satisfy (4). When the arrival times of a legal clustering satisfy (1)–(3) under ϕ, it is called a *feasible clustering* under ϕ. When a critical cycle is present in a feasible clustering under ϕ, it is called a *critical clustering* under ϕ. A given ϕ is *feasible* iff there exists a feasible clustering under ϕ. We must note that for a legal clustering, its maximum cycle ratio is feasible. In fact, any value larger than the maximum cycle ratio of a legal clustering is also feasible.

Given a feasible clustering c under ϕ, consider an edge $(u,v) \in E\ \forall v \in V$. It is either in E_c with $u \in i_v$, or there is an edge (u',v) such that u' is a replica of u. In either case, the following inequality is true because of (2), (3), and the fact that $t(u) \leq t(u')$ from [20]

$$t(v) \geq t(u) + d(u,v) + d(v) - w(u,v)\phi \quad \forall (u,v) \in E. \tag{5}$$

The following lemma provides a lower bound for ϕ.

Lemma 1: A feasible ϕ is no smaller than the maximum cycle ratio of G, denoted as ϕ_{lb}.

Proof: Since ϕ is feasible, then, by definition, there exists a clustering c satisfying (1)–(4) under ϕ. Therefore, (5) is true, and we have $d(o) \leq w(o)\phi$, for all cycle $o \in G$, i.e., $\phi \geq \phi_{\text{lb}}$.■

Define

$$\Delta(u,v,\phi) \triangleq \max_{p \in u \rightsquigarrow v \text{ in } G} (d(p) - w(p)\phi) \qquad \forall u,v \in V.$$

Lemma 1 ensures that $\Delta(u,v,\phi)$ is well-defined on feasible ϕ's and can be obtained by longest path computation in $O(|E| + |V|\log|V|)$ time [11].

V. Overview

The optimal clustering problem asks for a legal clustering with the minimal maximum-cycle-ratio. Since $A > 0$, the clustering with each vertex being a cluster is certainly legal. Starting from it, we will iteratively improve the clustering by reducing its maximum cycle ratio until the optimality is certified.

First of all, the maximum cycle ratio of a legal clustering is feasible and can be efficiently computed using Howard's algorithm [7], [12]. Given a feasible ϕ, we show that, unless ϕ is already the optimal solution, a particular legal clustering can be constructed whose maximum cycle ratio is smaller than ϕ. The smaller ϕ can be obtained by applying Howard's algorithm on the constructed clustering. Therefore, we alternate between applying Howard's algorithm and constructing a better clustering until the minimal ϕ is reached.

VI. Clustering Under a Given $\phi > \phi_{\text{lb}}$

Given $\phi > \phi_{\text{lb}}$, if ϕ is feasible, we show, in this section, how to construct a feasible clustering under ϕ, i.e., a clustering satisfying (1)–(5) under ϕ, whose maximum cycle ratio is no larger than ϕ.

A. Choosing (1) and (5) as Invariant

We choose to first satisfy (1) and (5) because they are independent on clustering, and iteratively update $t(v)$ and c_v to satisfy (2)–(4) while keeping (1) and (5).

Let T denote the arrival time vector, i.e.,

$$\mathrm{T} = (t(1), t(2), \ldots, t(|V|)).$$

A partial order $(\leq)$ can be defined between two arrival time vectors T and T' as follows:

$$T \leq T' \triangleq t(v) \leq t'(v) \qquad \forall v \in V.$$

According to the lattice theory [13], if we treat assignment $t(v) = 0\ \forall v \in V$ as the bottom element $(\perp)$ and assignment $t(v) = \infty\ \forall v \in V$ as the top element $(\top)$, then the arrival time vector space $\mathbb{R}^{|V|}$ becomes a *complete partially ordered set*, that is, for all $T \in \mathbb{R}^{|V|}$, $\perp \leq T \leq \top$.

To satisfy (1), we set $t(v) = 0\ \forall v \in V$. Then we apply a modified Bellman–Ford's algorithm, defined as $\mathrm{MBF}(T,\phi)$, on E to satisfy (5) under ϕ. The modified Bellman–Ford's algorithm is the same as Bellman–Ford's algorithm [11] except that it takes two inputs: a given arrival time vector T and a value of ϕ. The value of ϕ is used to specify (5) so that we can perform relaxation starting from the given arrival time vector T. The relaxation is guaranteed to converge when $\phi \geq \phi_{\text{lb}}$.

Given $\perp$ under ϕ, the resulting arrival time vector of MBF is denoted as

$$T_0 = \mathrm{MBF}(\perp, \phi).$$

In fact, T_0 is the *least* vector satisfying (1) and (5) under ϕ, as stated in the following lemma.

Lemma 2: $T_0 \leq T$, for all T satisfying (1) and (5) under ϕ.

Proof: Suppose we have a T satisfying (1) and (5) under ϕ with $t(v) < t_0(v)$ for some $v \in V$. It follows that $t_0(v) > 0$ since $t(v) \geq 0$ by (1). The modified Bellman-Ford's algorithm guarantees that there exists a path $p = u \rightsquigarrow v$ in G such that $t_0(u) = 0$ and $t_0(v) = t_0(u) + d(p) - w(p)\phi$. Since T satisfies (5) under ϕ, we have $t(v) \geq t(u) + d(p) - w(p)\phi = t(u) + t_0(v) \geq t_0(v)$, which contradicts $t(v) < t_0(v)$. Therefore, such a T does not exist and the lemma is true. ∎

B. Transformation $\mathcal{L}_v$ to Satisfy (2)–(4) Under ϕ

In order to satisfy (2)–(4) while keeping (1) and (5) under ϕ, we define transformation $\mathcal{L}_v : (\mathbb{R}^{|V|}, \mathbb{R}) \rightarrow \mathbb{R}\ \forall v \in V$, as follows.

For all $v \in V$, we will construct a new cluster c'_v, as opposed to the current c_v. The procedure starts with $c'_v = \{v\}$ and grows c'_v progressively by including one critical input at a time. Let $t'(v)$ denote the arrival time of v in c'_v, as opposed to $t(v)$ in c_v. Note that when a vertex is put in c'_v, its preceding vertices that are outside c'_v become inputs of c'_v. Therefore, $t'(v)$ varies every time c'_v grows. The procedure will stop only when either $|c'_v| = A$ or $t'(v) \leq t(v)$. When it stops, we compare $t'(v)$ with $t(v)$. If $t'(v) < t(v)$, we keep $t(v)$ and c_v unchanged; otherwise, we update $t(v)$ and c_v with $t'(v)$ and c'_v, respectively. The resulting arrival time of v is defined as $\mathcal{L}_v(T, \phi)$. The next lemma helps to identify the critical input to be included at each time.

Lemma 3: Given that (1) and (5) are satisfied under ϕ, we have $t'(v) \geq t(x) + D + \Delta(x, v, \phi)\ \forall x \notin c'_v$. In particular, if u is a critical input of c'_v, then $t'(v) = t(u) + D + \Delta(u, v, \phi)$.

Proof: For all $x \notin c'_v$, let p be the path from x to v in G such that $\Delta(x, v, \phi) = d(p) - w(p)\phi$. Since $x \notin c'_v$, there exists a vertex $y \in i'_v$ on p which divides p into two subpaths: p_1 from x to y, and p_2 from y to v. In addition, p_2 has the form of $y \rightarrow z \rightsquigarrow v$, where $z \rightsquigarrow v$ is in c'_v. By (3), we have $t'(v) \geq t(y) + D + d(p_2) - w(p_2)\phi$. In addition, $t(y) \geq t(x) + d(p_1) - w(p_1)\phi$ by (5). Therefore, $t'(v) \geq t(x) + D + d(p) - w(p)\phi = t(x) + D + \Delta(x, v, \phi)$.

If c'_v has a critical input u, then, by definition, there exists a path $p = u \rightsquigarrow v$ in G such that $t(u) + D + d(p) - w(p)\phi = t'(v)$. Since $u \notin c'_v$, we have $t'(v) \geq t(u) + D + \Delta(u, v, \phi)$, thus, $d(p) - w(p)\phi \geq \Delta(u, v, \phi)$. On the other hand, $d(p) - w(p)\phi \leq \Delta(u, v, \phi)$ since $\Delta(u, v, \phi)$ is the largest among all paths from u to v in G. Therefore, $d(p) - w(p)\phi = \Delta(u, v, \phi)$, which concludes our proof. ∎

Lemma 3 implies that when a vertex is put in c'_v, its preceding vertices that are already in c'_v can be ignored for the computation of $t'(v)$.

C. Implementation of $\mathcal{L}_v$

Our implementation of $\mathcal{L}_v$ is similar to the label computation in [10]. To characterize critical inputs, we introduce another label $\delta(u)\ \forall u \in V$. Before the construction of c'_v, we assign $\delta(u)$ with $-\infty$ for all $u \neq v$ in V while $\delta(v)$ with 0. At each time, the vertex $u \in i'_v$ with the largest $t(u) + \delta(u)$ is identified. If $t(u) + D + \delta(u) \leq t(v)$, the construction is completed. Otherwise, we put it in c'_v and update $\delta(x)$ with $\max(\delta(x), \delta(u) + d(u) + d(x, u) - w(x, u)\phi)\ \forall (x, u) \in E$. This procedure will iterate until either $|c'_v| = A$ or the last vertex u identified has $t(u) + D + \delta(u) \leq t(v)$.

To validate the previous procedure, we need to show that it is equivalent to $\mathcal{L}_v(T, \phi)$, or equivalently, to show that it can always identify the critical input of c'_v. This is fulfilled by the next lemma and corollary.

Lemma 4: Given that (1) and (5) are satisfied under ϕ, we have $\delta(u) = \Delta(u, v, \phi)\ \forall u \in c'_v$.

Proof: We prove it by induction on the size of c'_v. At the beginning, $c'_v = \{v\}$ and $\delta(v) = 0 = \Delta(v, v, \phi)$. Suppose that the lemma is true for $|c'_v| \leq k < A$, we need to show that the lemma is also true for $|c'_v| = k + 1$. Therefore, we start with $|c'_v| = k$ and let $u \in i'_v$ be the vertex with the largest $t(u) + D + \delta(u) > t(v)$.

We use p to denote the path where $d(p) - w(p)\phi = \Delta(u, v, \phi)$. Since $u \notin c'_v$, there exists a vertex $x \in i'_v$ on p such that the subpath from x to v has the form of $x \rightarrow y \rightsquigarrow v$, where $y \rightsquigarrow v$ is in c'_v. Let p_1 and p_2 be the subpath from u to x and from y to v, respectively. Since $d(p) - w(p)\phi = \Delta(u, v, \phi)$, we have $d(p_1) - w(p_1)\phi = \Delta(u, x, \phi)$ and $d(p_2) - w(p_2)\phi = \Delta(y, v, \phi)$.

Given that $u \in i'_v$ is the vertex with the largest $t(u) + \delta(u)$, we know that $t(u) + \delta(u) \geq t(x) + \delta(x)$. (5) ensures that $t(x) \geq t(u) + d(p_1) - w(p_1)\phi$. On the other hand, we have $\delta(x) \geq \delta(y) + d(x, y) - w(x, y)\phi$ since we updated $\delta(x)$ to be no less than $\delta(y) + d(x, y) - w(x, y)\phi$ when y was put in c'_v. Further, $\delta(y) = \Delta(y, v, \phi)$ by the inductive hypothesis. Consequently

$$\begin{aligned}
t(u) + \delta(u) &\geq t(x) + \delta(x) \\
&\geq (t(u) + d(p_1) - w(p_1)\phi) + \delta(x) \\
&\geq (t(u) + d(p_1) - w(p_1)\phi) \\
&\quad + (\delta(y) + d(x, y) - w(x, y)\phi) \\
&= (t(u) + d(p_1) - w(p_1)\phi) \\
&\quad + (\Delta(y, v, \phi) + d(x, y) - w(x, y)\phi) \\
&= (t(u) + d(p_1) - w(p_1)\phi) \\
&\quad + ((d(p_2) - w(p_2)\phi) + d(x, y) - w(x, y)\phi) \\
&= t(u) + d(p) - w(p)\phi \\
&= t(u) + \Delta(u, v, \phi)
\end{aligned}$$

or $\delta(u) \geq \Delta(u, v, \phi)$. However, $\delta(u) \leq \Delta(u, v, \phi)$ since $\Delta(u, v, \phi)$ is the largest among all paths from u to v. Therefore, $\delta(u) = \Delta(u, v, \phi)$. ∎

Corollary 4.1: Given that (1) and (5) are satisfied under ϕ, the vertex $u \in i'_v$ with the largest $t(u) + D + \delta(u) \geq t(v)$ is the critical input of c'_v.

Proof: Suppose u is not a critical input. Let p denote the path where $d(p) - w(p)\phi = \delta(u)$. Since p is not critical, we have $t'(v) > t(u) + D + d(p) - w(p)\phi = t(u) + D + \delta(u) \geq t(v)$. It implies that c'_v has a critical input x, otherwise the vertices that have critical paths to v are all inside c'_v and we have $t'(v) = t_0(v) \leq t(v)$, which is a contradiction. Let p' be a critical path $x \rightarrow y \rightsquigarrow v$ from x to v whose subpath $p'_1 : y \rightsquigarrow v$ is in c'_v and $t'(v) = t(x) + D + d(p') - w(p')\phi$. In addition, $t'(v) = t(x) + D + \Delta(x, v, \phi)$ by Lemma 3. Thus, $\Delta(x, v, \phi) = d(p') - w(p')\phi$, which implies that $\Delta(y, v, \phi) = d(p'_1) - w(p'_1)\phi$.

On the other hand, since $y \in c'_v$, we have $\delta(y) = \Delta(y, v, \phi)$ by Lemma 4. Hence, $\delta(x) \geq \delta(y) + d(x, y) - w(x, y)\phi =$

Input: $G=(V,E)$, A, D, T, ϕ, $v \in V$.
Output: $t'(v)$, c'_v.

$\delta(u) \leftarrow -\infty$, $\forall u \in V$; $\delta(v) \leftarrow 0$;
$Q \leftarrow \{v\}$; $c'_v \leftarrow \emptyset$;
While $\big((Q \neq \emptyset) \wedge (|c_v| < A)\big)$
  $u \leftarrow$ extract from Q with max $t(u)+\delta(u)$;
  $t'(v) \leftarrow t(u)+D+\delta(u)$;
  $c'_v \leftarrow c'_v \cup \{u\}$;
  For $e=(x,u) \in E$ with $x \notin c'_v$
    If $\big(\delta(x) < \delta(u)+d(u)+d(x,u)-w(x,u)\phi\big)$
      $\delta(x) \leftarrow \delta(u)+d(u)+d(x,u)-w(x,u)\phi$;
    If $\Big((x \notin Q) \wedge \big(t(x)+D+\delta(x) > t(v)\big)\Big)$
      $Q \leftarrow Q \cup \{x\}$;
If $(Q \neq \emptyset)$
  $t'(v) \leftarrow$ max $t(u)+D+\delta(u)$ in Q;
Else $t'(v) \leftarrow t(v)$;
Return $t'(v)$ and c'_v;

Fig. 4. Pseudocode of $\mathcal{L}_v(T,\phi)$.

$d(p')-w(p')\phi = \Delta(x,v,\phi)$. Since $\Delta(x,v,\phi)$ is the largest among all paths from x to v, we have $\delta(x)=\Delta(x,v,\phi)$. It follows that $t(x)+D+\delta(x)=t'(v) > t(u)+D+\delta(u)$, which contradicts that u is the input with the largest $t(u)+\delta(u)$. Therefore, the lemma is true. ■

The pseudocode for computing $\mathcal{L}_v(T,\phi)$ is given in Fig. 4. It employs a heap Q for bookkeeping the vertices $u \in i'_v$ whose $t(u)+D+\delta(u) > t(v)$. At each iteration, it puts in c'_v the input $u \in Q$ with the largest $t(u)+D+\delta(u)$ and updates $\delta(x)$ for each fanin of u that becomes an input in i'_v. In our implementation, we choose Fibonacci heap [11] for Q.

The complexity of $\mathcal{L}_v(T,\phi)$ in Fig. 4 is given in the following lemma.

Lemma 5: The procedure in Fig. 4 terminates in $O(|E|\log|V|)$ time.

Proof: At each iteration, the complexity of extracting the vertex u with the maximum $t(u)+\delta(u)$ is $O(\log|V|)$ by Fibonacci heap [11]. For each $(x,u) \in E$, updating $\delta(x)$ takes $O(\log|V|)$ time. On the other hand, since a vertex cannot be put in Q more than once, the total number of edges processed by the inner for-loop is $O(|E|)$. Therefore, the complexity the procedure is $O(A\log|V|+|E|\log|V|)$, or $O(|E|\log|V|)$. ■

D. Clustering Under ϕ as a Fixpoint Computation

We define $\mathcal{L}(T,\phi)$ as the arrival time vector when all the $\mathcal{L}_v(T,\phi)$'s $\forall v \in V$, are applied once, followed by the modified Bellman-Ford's algorithm to ensure (5), expressed as

$$\mathcal{L}(\mathrm{T},\phi) \triangleq \mathrm{MBF}\left(\left(\mathcal{L}_1(\mathrm{T},\phi),\mathcal{L}_2(\mathrm{T},\phi),\ldots,\mathcal{L}_{|V|}(\mathrm{T},\phi)\right),\phi\right).$$

The following lemma shows that $\mathcal{L}$ is an order-preserving transformation.

Lemma 6: For any T and $\bar{T}$ satisfying (1) and (5) under ϕ, if $T \leq \bar{T}$, then $\mathcal{L}(T,\phi) \leq \mathcal{L}(\bar{T},\phi)$.

Proof: We first show that $\mathcal{L}_v(T,\phi) \leq \mathcal{L}_v(\bar{T},\phi)\ \forall v \in V$.

For the sake of contradiction, we assume that $\mathcal{L}_v(T,\phi) > \mathcal{L}_v(\bar{T},\phi)$ for some $v \in V$. The procedure of $\mathcal{L}_v$ guarantees that $\mathcal{L}_v(T,\phi)=\max(t'(v),t(v))$ and $\mathcal{L}_v(\bar{T},\phi)=\max(\bar{t}'(v),\bar{t}(v))$. Since $t(v) \leq \bar{t}(v)$ by $T \leq \bar{T}$, we have $t'(v) > t(v)$, otherwise, $\mathcal{L}_v(T,\phi)=t(v) \leq \bar{t}(v) \leq \mathcal{L}_v(\bar{T},\phi)$, which contradicts the assumption that $\mathcal{L}_v(T,\phi) > \mathcal{L}_v(\bar{T},\phi)$. In addition, since T satisfies (1) and (5) under ϕ, Lemma 2 ensures that $t(v) \geq t_0(v)$, where $t_0(v)$ is the arrival time of v in vector $T_0=\mathrm{MBF}(\perp,\phi)$. Thus, $t'(v) > t_0(v)$, which implies that cluster c'_v has a critical input u, otherwise, the vertices that have critical paths to v are all inside c'_v and we have $t'(v)=t_0(v)$, which is a contradiction. The existence of a critical input u implies that $|c'_v|=A$, otherwise, u should have been put in c'_v since $t'(v) > t(v)$. Let p be the path where $\Delta(u,v,\phi)=d(p)-w(p)\phi$. Fig. 5(a) shows an example of c'_v.

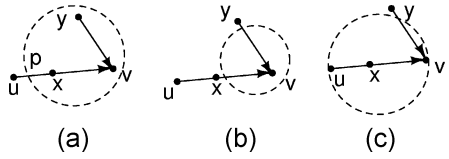


Fig. 5. (a) Cluster c'_v. (b) and (c) Two cases of cluster $\bar{c}'_v$.

Now consider cluster $\bar{c}'_v$, there are two cases. First, $u \notin \bar{c}'_v$. Thus, there exists a vertex $x \in p$ such that $x \in \bar{i}'_v$, as illustrated in Fig. 5(b). x divides p into two subpaths: p_1 from u to x, and p_2 from x to v. We know that $\Delta(x,v,\phi)=d(p_2)-w(p_2)\phi$, otherwise, p cannot subsume p_2. Together with $t(x) \geq t(u)+d(p_1)-w(p_1)\phi$ by (5), we have $t(x)+\Delta(x,v,\phi) \geq t(u)+\Delta(u,v,\phi)$. On the other hand, since $x \in \bar{i}'_v$, we have $\bar{t}'(v) \geq \bar{t}(x)+D+\Delta(x,v,\phi)$ by Lemma 3. Given that $t(x) \leq \bar{t}(x)$ (since $T \leq \bar{T}$), we have $\bar{t}'(v) \geq t(u)+D+\Delta(u,v,\phi)=t'(v)$.

Second, $u \in \bar{c}'_v$. Given that $|c'_v|=A$ and u is not in c'_v, we know that there exists a vertex $y \in c'_v$ such that $y \in \bar{i}'_v$, as shown in Fig. 5(c). By the same argument, we can show that $\bar{t}'(v) \geq \bar{t}(y)+D+\Delta(x,v,\phi) \geq t(y)+D+\Delta(x,v,\phi) \geq t(u)+D+\Delta(u,v,\phi)=t'(v)$.

In either case, we have $\max(t(v),t'(v)) \leq \max(\bar{t}(v),\bar{t}'(v))$, i.e., $\mathcal{L}_v(T,\phi) \leq \mathcal{L}_v(\bar{T},\phi)$, which is a contradiction. Therefore, the assumption is wrong and $\mathcal{L}_v(T,\phi) \leq \mathcal{L}_v(\bar{T},\phi)$ is true $\forall v \in V$. It is easy to verify that $\mathcal{L}(T,\phi) \leq \mathcal{L}(\bar{T},\phi)$ after applying the modified Bellman–Ford's algorithm. ■

We say that T is a *fixpoint of* $\mathcal{L}$ under ϕ if and only if $T=\mathcal{L}(T,\phi)$. The following theorem bridges the existence of a fixpoint and the feasibility of ϕ.

Theorem 1: ϕ is feasible if and only if $\mathcal{L}$ has a fixpoint under ϕ.

Proof: ($\rightarrow$): If ϕ is feasible, then, by definition, there exists a legal clustering c whose arrival time vector T satisfies (1)–(3) and (5) under ϕ. We claim that $\mathcal{L}_v(T,\phi) \leq t(v)\ \forall v \in V$. Otherwise, $t'(v) > t(v) \geq t_0(v)$ for some $v \in V$, and c'_v has a critical input u, as shown in Fig. 5(a). We can conduct a similar case study as Fig. 5(b) and (5c) to show that $t(v) \geq t'(v)$, which is a contradiction. On the other hand, $t(v) \leq \mathcal{L}_v(T,\phi)$ by the procedure of $\mathcal{L}_v(T,\phi)$. Therefore, $\mathcal{L}_v(T,\phi)=t(v)\ \forall v \in V$. Given that T satisfies (5), applying the modified Bellman-Ford's algorithm gives $T=\mathcal{L}(T,\phi)$, i.e., T is a fixpoint of $\mathcal{L}$ under ϕ.

($\leftarrow$): If $\mathcal{L}$ has a fixpoint under ϕ, then, by the definition of $\mathcal{L}$, the constructed clustering is legal and the arrival time vector satisfies (1)–(3) and (5) under ϕ. Therefore, ϕ is feasible. ■

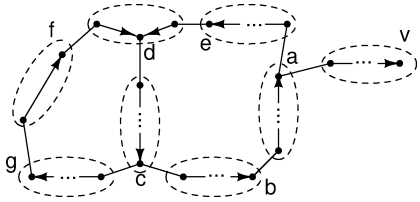


Fig. 6. Vertices that have critical paths to v.

In fact, according to the lattice theory [13], if $\mathcal{L}$, defined on a complete partially ordered set, has a fixpoint under ϕ, then it has a *least fixpoint* T^ϕ, defined as

$$\left(T^\phi = \mathcal{L}(T^\phi, \phi)\right) \wedge \left(\forall\, T : T = \mathcal{L}(T, \phi) : T^\phi \leq T\right).$$

We use c^ϕ to denote the clustering constructed by $\mathcal{L}(T^\phi, \phi)$. In fact, if $t^\phi(v) > t_0(v)$, then there is a critical path from $u \in V$ to v with $t^\phi(u) = t_0(u)$. This is made precise in the following lemma.

Lemma 7: If $t^\phi(v) > t_0(v)$, then there exists a sequence of vertices $x_i \in V, i = 0, 1, \ldots, k-1$ such that $t_0(x_0) = t^\phi(x_0)$, $x_k = v$, and x_i is a critical input of cluster $c^\phi_{x_{i+1}}$.

Proof: Since $t^\phi(v) > t_0(v)$, we know that cluster c^ϕ_v has critical inputs, otherwise, the vertices that have critical paths to v are all inside c^ϕ_v and we have $t_0(v) = t^\phi(v)$, which is a contradiction.

Suppose, otherwise, that such a sequence does not exist, namely, all the critical paths terminating at v are actually critical cycles, where each constituent vertex $u \in V$ has $t^\phi(u) > t_0(u) \geq 0$. Choose the one that contains all of them, denoted as O. For the example in Fig. 6, we will choose O to be $a \rightsquigarrow e \rightsquigarrow d \rightsquigarrow c \rightsquigarrow g \rightsquigarrow f \rightsquigarrow d \rightsquigarrow c \rightsquigarrow b \rightsquigarrow a$. Now consider any incoming edge of O (from a vertex outside of O to a vertex in O), it must be noncritical, otherwise, we can trace back from this edge and find another critical cycle that is not in O, which is a contradiction. Since the arrival times of the vertices in O are all greater than zero, we can decrease them simultaneously while keeping the arrival times of other vertices unchanged until some incoming edge of O becomes critical or the arrival time of some vertex in O is reduced to zero. For either case, we obtain a fixpoint less than T^ϕ, which is a contradiction. Therefore, the lemma is true. ∎

To reach a fixpoint, iterative method can be used on $\mathcal{L}$. It starts with T_0 as the initial vector, iteratively computes new vectors from previous ones $T_1 = \mathcal{L}(T_0, \phi), T_2 = \mathcal{L}(T_1, \phi), \ldots$, until it finds a T_n such that $T_n = T_{n-1}$. The following lemma states that applying iterative method on $\mathcal{L}$ will converge to its least fixpoint in a finite number of iterations.

Lemma 8: If ϕ is feasible, applying iterative method on $\mathcal{L}$ will converge to T^ϕ in a finite number of iterations.

Proof: Since we start with $T = T_0 \leq T^\phi$, Lemma 6 ensures that $T \leq T^\phi$ at each iteration. Therefore, if $\mathcal{L}$ converges, the fixpoint has to be the least fixpoint. What remains is to show that $\mathcal{L}$ is finitely convergent.

By Lemma 3 and 7, if $t^\phi(v) > t_0(v)$, then $t^\phi(v)$ can be written as

$$t^\phi(v) = t_0(x_0) + \sum_{0 \leq i \leq k-1} (D + \Delta(x_i, x_{i+1}, \phi))$$

where $x_i \in V$ and $x_k = v$. Given that each vertex in V has exactly one cluster rooted at it, we know that $k \leq |V|$, thus, $t^\phi(v) \leq U$, where $U = (|V|-1)D + |V| \max_{u,v \in V} \Delta(u, v, \phi)$.

On the other hand, since ϕ is a rational number, it can be expressed as r/q, where r and q are integers and $q \neq 0$. If $t(v)$ is increased during the iteration, the amount of increase will be at least $1/q$. Therefore, if $\mathcal{L}$ does not converge after $|V|Uq$ iterations, then there exists a vertex $v \in V$ whose $t(v) > U \geq t^\phi(v)$, which contradicts $T \leq T^\phi$, which concludes our proof. ∎

The next result is a corollary of Lemma 6–8.

Lemma 9: $(\forall v \in V : t(v) > t_0(v))$ implies that ϕ is infeasible.

Proof: Suppose, otherwise, that ϕ is feasible. Then, by Lemma 6 and 8, when T^ϕ is reached, we have $(\forall v \in V : t^\phi(v) > t_0(v))$, which contradicts Lemma 7. Therefore, ϕ is infeasible. ∎

VII. Optimal Clustering Algorithm

Given a legal clustering c, its maximum cycle ratio ϕ_c is feasible. If ϕ_c is not optimal, then we can find a feasible $\phi_{c'} < \phi_c$, which is specified in the following lemma.

Lemma 10: Given that ϕ_c is the maximum cycle ratio of a legal clustering c, if ϕ_c is not optimal, then $\phi_c - 1/(|V|N_{\text{ff}})^2$ is also feasible, where N_{ff} is the total number of flip-flops in G.

Proof: Let o_c denote the cycle with the maximum cycle ratio, that is, $\phi_c = d(o_c)/w(o_c)$. If ϕ_c is not optimal, it means that there exists another legal clustering c' whose maximum cycle ratio $\phi_{c'}$ is smaller than ϕ_c. Let $o_{c'}$ be the cycle with $\phi_{c'} = d(o_{c'})/w(o_{c'})$. The difference between $\phi_{c'}$ and ϕ_c can be written as

$$\begin{aligned}\phi_c - \phi_{c'} &= \frac{d(o_c)}{w(o_c)} - \frac{d(o_{c'})}{w(o_{c'})} = \frac{d(o_c)w(o_{c'}) - d(o_{c'})w(o_c)}{w(o_c)w(o_{c'})} \\ &\geq \frac{1}{w(o_c)w(o_{c'})}\end{aligned}$$

since all delays are integers. In addition, since each vertex in V has exactly one cluster rooted at it, both $w(o)$ and $w(o')$ can pass at most $|V|$ clusters. Thus, neither $w(o)$ nor $w(o')$ will be larger than $|V|N_{\text{ff}}$. Therefore, $\phi_c - \phi_{c'} \geq 1/(|V|N_{\text{ff}})^2$. In other words, $\phi_c - 1/(|V|N_{\text{ff}})^2$ is also feasible. ∎

It implies that we can certify the optimality of ϕ_c by checking the feasibility of $\phi_c - 1/(|V|N_{\text{ff}})^2$. The algorithm for finding the optimal ϕ is presented in Fig. 7. It first computes a feasible ϕ by treating each vertex as a cluster, and computes a lower bound of ϕ_{lb} by Lemma 1. After that, it checks the feasibility of $\phi - 1/(|V|N_{\text{ff}})^2$ by iterative method on $\mathcal{L}$. If $\mathcal{L}$ converges, it means that we find a better clustering whose maximum cycle ratio is at most $\phi - 1/(|V|N_{\text{ff}})^2$ and can be computed by Howard's algorithm. The evidence of $(\forall v \in V : t(v) > t_0(v))$ or the fact that ϕ is reduced below $\phi_{\text{lb}} + 1/(|V|N_{\text{ff}})^2$ immediately certifies the optimality of the current feasible ϕ.

We prove the correctness of the algorithm by showing that it returns the optimal ϕ when it terminates.

Theorem 2: The algorithm in Fig. 7 will return a clustering with the optimal ϕ when it terminates.

Proof: When the algorithm terminates, we have either $\phi < \phi_{\text{lb}} + 1/(|V|N_{\text{ff}})^2$, or $(\forall v \in V : t(v) > t_0(v))$ under $\phi -$

Algorithm Optimal clustering
Input: A directed graph $G=(V,E)$, A, D.
Output: A clustering $c^{\rm opt}$ with $\phi^{\rm opt}$.

$c_v^{\rm opt} \leftarrow c_v \leftarrow \{v\}$, $\forall v \in V$;
$\phi^{\rm opt} \leftarrow \phi \leftarrow$ maximum cycle ratio of G_c;
$\phi_{\rm lb} \leftarrow$ maximum cycle ratio of G;
While $\left(\phi \geq \phi_{\rm lb} + 1/(|V|N_{\rm ff})^2\right)$
  $\phi \leftarrow \phi - 1/(|V|N_{\rm ff})^2$;
  $\mathrm{T} \leftarrow \mathrm{T}_0 \leftarrow MBF(\perp, \phi)$;
  While $\Big(\left(\mathrm{T} \neq \mathcal{L}(\mathrm{T},\phi)\right) \wedge \left(\exists v \in V : t(v) = t_0(v)\right)\Big)$
    $\mathrm{T} \leftarrow \mathcal{L}(\mathrm{T},\phi)$;
  If $\left(\forall v \in V : t(v) > t_0(v)\right)$
    break;
  $\phi \leftarrow$ maximum cycle ratio of G_{c^ϕ};
  $c^{\rm opt} \leftarrow c^\phi$; $\phi^{\rm opt} \leftarrow \phi$;
Return $c^{\rm opt}$ and $\phi^{\rm opt}$;

Fig. 7. Pseudocode of optimal clustering algorithm

$1/(|V|N_{\rm ff})^2$. For the first case, Lemma 10 ensures that ϕ is optimal, otherwise, $\phi - 1/(|V|N_{\rm ff})^2$ is feasible, which contradicts Lemma 1. For the second case, $\phi - 1/(|V|N_{\rm ff})^2$ is infeasible by Lemma 9, which, by Lemma 10, implies that ϕ is optimal. The optimal ϕ and the corresponding clustering are recorded in $\phi^{\rm opt}$ and $c^{\rm opt}$. ■

We finally present the worst case complexity of the algorithm in the next theorem.

Theorem 3: The algorithm in Fig. 7 terminates in $O(|V|^9|E|N_{\rm ff}^5 D(D+\Delta_{\rm lb})\log|V|)$ time, where $\Delta_{\rm lb} = \max_{u,v\in V}\Delta(u,v,\phi_{\rm lb})$ and $N_{\rm ff}$ is the total number of flip-flops in G.

Proof: First of all, ϕ is reduced during the execution of the outer while-loop in Fig. 7. Since the amount of decrease in ϕ is at least $1/(|V|N_{\rm ff})^2$ after each loop, the algorithm will terminate in $(\phi_{\rm ub}-\phi_{\rm lb})|V|^2N_{\rm ff}^2$ loops, where $\phi_{\rm ub}$ is an upper bound of ϕ. Since $\phi_{\rm ub} \leq \phi_{\rm lb} + |V|D$, the number of loops can be bounded by $|V|^3N_{\rm ff}^2D$.

At each loop, it takes $O(|V||E|)$ to compute T_0 by the modified Bellman-Ford's algorithm. The complexity of maximum-cycle-ratio computation can be bounded by $O(|V_c|^2|E_c|)$ [12], or $O(A^2|V|^3|E|)$ since G_c consists of $|V|$ clusters and the size of each cluster is no larger than A.

We next analyze the complexity of checking the feasibility of ϕ. According to the proof of Lemma 8, if ϕ is feasible, iterative method will converge in $|V|Uq$ iterations, where $U = (|V|-1)D + |V|\max_{u,v\in V}\Delta(u,v,\phi)$ and q is an integer such that the product of ϕ and q is integral. Since $\phi \geq \phi_{\rm lb}$, we have $\Delta(u,v,\phi) \leq \Delta(u,v,\phi_{\rm lb})\ \forall u,v \in V$. On the other hand, since ϕ is the maximum cycle ratio of a legal clustering minus $1/(|V|N_{\rm ff})^2$, it is true that $q \leq |V|^3N_{\rm ff}^3$. As a result, the number of iterations can be bounded by $O(|V|^5N_{\rm ff}^3(D+\Delta_{\rm lb}))$, where $\Delta_{\rm lb} = \max_{u,v\in V}\Delta(u,v,\phi_{\rm lb})$. The complexity of each iteration can be computed as $O(|V||E|\log|V|)$ by Lemma 5.

Therefore, the computational complexity of each loop is $O(|V|^6|E|N_{\rm ff}^3(D+\Delta_{\rm lb})\log|V|)$ and the theorem is true. ■

Remark 1: The significance of Theorem 3 is not the actual formula of the bound, but showing that the optimal clustering problem has a pseudopolynomial time complexity. Furthermore, caution should be taken on this bound. The worst case complexity is based on a series of assumptions that are very unlikely to be attainable in reality. For example, although we do feasible checking on a value that is slightly smaller than a given feasible ϕ, the improvement at each loop is not small. This is because the new clustering will have a different structure whose maximum cycle ratio is usually much smaller than the given ϕ. This is confirmed by our experiments in Section X. Therefore, we believe that the worst case bound we obtained is just an upper bound of the actual running time. A tighter bound may exist but its mathematical analysis is so complex that we cannot deduce it so far. The efficiency of our algorithm is confirmed by the experiments.

VIII. Speed-Up Techniques

A. Variations of $\mathcal{L}$

In Section VI, $\mathcal{L}(T,\phi)$ is defined as applying all the $\mathcal{L}_v$'s $\forall v \in V$, once followed by the modified Bellman-Ford's algorithm. In our implementation, all the $\mathcal{L}_v(T,\phi)$'s are not computed at the same time. Intuitively, if previously computed $\mathcal{L}_v$'s can be taken into account in later computations of others, the convergence rate may be accelerated.

This motivates our study on a variation of $\mathcal{L}$, in which later computations of $\mathcal{L}_v$'s are based on previously computed ones, and each computation of $\mathcal{L}_v$ is followed by the modified Bellman-Ford's algorithm. Let $\mathcal{J}_v(T,\phi)$ denote the vector after $t(v)$ is updated with $\mathcal{L}_v(T,\phi)$, that is

$$\mathcal{J}_v(T,\phi) = (t(1),\ldots,t(v-1),\mathcal{L}_v(T,\phi),t(v+1),\ldots,t(|V|)).$$

Define

$$\bar{\mathcal{L}}(T,\phi) \triangleq \mathrm{MBF}\left(\mathcal{J}_{i_{|V|}}(\ldots\mathrm{MBF}(\mathcal{J}_{i_1}(T,\phi),\phi),\ldots,\phi),\phi\right)$$

where $i_1,\ldots,i_{|V|} \in V$. It can be shown that different evaluation orders of V give different $\bar{\mathcal{L}}$'s. However, they all satisfy the following relation.

Lemma 11: For any $T \leq T^\phi$ satisfying (1) and (5) under a feasible ϕ and any evaluation order of V, $\mathcal{L}(T,\phi) \leq \bar{\mathcal{L}}(T,\phi) \leq \bar{\mathcal{L}}(T^\phi,\phi) = T^\phi$.

Proof: Let $\bar{\mathcal{L}}_v(T,\phi)$ denote the arrival time of $v \in V$ in $\bar{\mathcal{L}}(T,\phi)$. Since T satisfies 1) and (5) under ϕ, the definition of $\bar{\mathcal{L}}(T,\phi)$ implies that $\mathcal{L}_v(T,\phi) \leq \bar{\mathcal{L}}_v(T,\phi)\ \forall v \in V$, independent of the evaluation order. It follows that $\mathcal{L}(T,\phi) \leq \bar{\mathcal{L}}(T,\phi)$. On the other hand, since $\mathcal{L}_v(T^\phi,\phi) = t^\phi(v)$, we have $\mathcal{J}_v(T^\phi,\phi) = T^\phi$, hence, $\bar{\mathcal{L}}(T^\phi,\phi) = T^\phi$. What remains to show is $\bar{\mathcal{L}}(T,\phi) \leq T^\phi$. To this aim, we observe that $\mathrm{MBF}(\mathcal{J}_v(T,\phi)) \leq T^\phi\ \forall v \in V$, provided that $T \leq T^\phi$. Based on this, we can show by induction that $\bar{\mathcal{L}}(T,\phi) \leq T^\phi$. Therefore, the lemma is true. ■

As a corollary, the next result ensures that we can apply iterative method on $\bar{\mathcal{L}}$ to reach T^ϕ.

Corollary 11.1: If ϕ is feasible, applying iterative method on $\bar{\mathcal{L}}$ will converge to T^ϕ in a finite number of iterations, independent of the evaluation order of V.

Proof: Since ϕ is feasible, $\mathcal{L}$ is finitely convergent by Lemma 8. Let K be the number of iterations such that $\mathcal{L}^K(T_0,\phi) = T^\phi$. The corollary can be proven

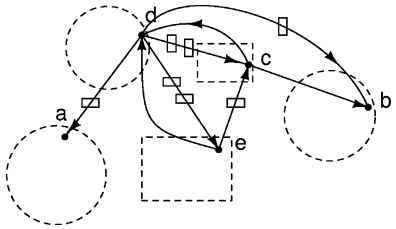


Fig. 8. Reduced clustering representation of Fig. 3(b).

if we can show that $\bar{\mathcal{L}}^K(T_0,\phi) = T^\phi$, or equivalently, $\mathcal{L}^K(T_0,\phi) \le \bar{\mathcal{L}}^K(T_0,\phi) \le \bar{\mathcal{L}}^K(T^\phi,\phi)$.

The former part can be derived from Lemma 11 because $\mathcal{L}^K(T_0,\phi) \le \bar{\mathcal{L}}(\mathcal{L}^{K-1}(T_0,\phi)) \le \ldots \le \bar{\mathcal{L}}^K(T_0,\phi)$. The latter part is also a consequence of Lemma 11 since $T_0 \le T^\phi$. ■

B. Reduced Clustering Representation

It was shown in [12] that Howard's algorithm was by far the fastest algorithm for maximum-cycle-ratio computation. Given a clustered circuit $G_c = (V_c, E_c)$ with edge delays and weights specified, Howard's algorithm finds the maximum cycle ratio in $O(N_c|E_c|)$ time, where N_c is the product of the out-degrees of all the vertices in V_c. Since vertex replication is allowed, N_c and $|E_c|$ could be $|V|N$ and $|V||E|$, respectively, where N is the product of the out-degrees of the vertices in V.

To reduce the complexity, we propose a reduced clustering representation for maximum-cycle-ratio computation. For each cluster, we use edges from its inputs to its output (root) to represent the paths between them such that the delay and weight of an edge correspond to the delay and weight of an acyclic input-to-output path. Fig. 8 shows the reduced representation of the clustered circuit in Fig. 3(b).

Let ϕ_c^r denote the maximum cycle ratio of the reduced representation for clustering c. The following lemma formulates the relation among ϕ_c, ϕ_c^r, and the lower bound ϕ_{lb} defined in Lemma 1.

Lemma 12: For any clustering c, $\phi_c = \max(\phi_c^r, \phi_{\text{lb}})$.

Proof: All the cycles in G_c can be classified into two groups according to whether they contain an inter-cluster edge or not. If a cycle contains only intra-cluster edges, its maximum cycle ratio is upper bounded by ϕ_{lb}. If a cycle contains inter-cluster edges, it is present in the reduced representation and, thus, is upper bounded by ϕ_c^r. ■

One benefit of the reduced clustering representation is that we can now represent the clustered circuit without explicit vertex replication, that is, using V instead of V_c. Let $G_c^r = (V, E_c^r)$ denote the reduced representation for clustering c. We call an edge in G_c^r *redundant* if its removal will not affect the maximum cycle ratio of G_c^r. The following lemma provides a criterion to prune the redundant edges so that Howard's algorithm can find the maximum cycle ratio of G_c^r more efficiently.

Lemma 13: Let c denote a feasible clustering under ϕ, E_c^r denote its reduced representation, e_1 and e_2 denote two edges from $u \in V$ to $v \in V$ in E_c^r, $d(e_1)$ and $d(e_2)$ denote their delays, respectively, and $w(e_1)$ and $w(e_2)$ denote their weights, respectively. If $w(e_1) \ge w(e_2)$ and $d(e_1) - w(e_1)\phi \ge d(e_2) - w(e_2)\phi$, then e_2 can be pruned.

Proof: Since c is feasible under ϕ, we know that $\phi \ge \phi_c \ge \phi_c^r$. If e_2 is not involved in any cycle in G_c^r, then e_2 can be safely pruned as it will not affect the computation of the maximum cycle ratio. Otherwise, let o_2 be a cycle in G_c^r involving e_2, and o_1 be the cycle in G_c^r such that $o_1 = \{e_1\} \cup o_2 - \{e_2\}$. What remains to show is that if o_2 is a critical cycle under ϕ_c^r, so is o_1.

By definition, if o_2 is a critical cycle under ϕ_c^r, then $d(o_2) - w(o_2)\phi_c^r = 0$. Since o_1 differs o_2 in e_1 only, we have

$$\begin{aligned} d(o_1) - w(o_1)\phi_c^r &= d(o_2) - d(e_2) + d(e_1) \\ &\quad - (w(o_2) - w(e_2) + w(e_1))\,\phi_c^r \\ &= d(e_1) - w(e_1)\phi_c^r - (d(e_2) - w(e_2)\phi_c^r) \\ &= d(e_1) - w(e_1)\phi - (d(e_2) - w(e_2)\phi) \\ &\quad + (w(e_1) - w(e_2))\,(\phi - \phi_c^r) \\ &\ge 0 \end{aligned}$$

provided that $d(e_1) - w(e_1)\phi \ge d(e_2) - w(e_2)\phi$ and $w(e_1) \ge w(e_2)$. On the other hand, $d(o_1) \le w(o_1)\phi_c^r$ since ϕ_c^r is the maximum cycle ratio. Therefore, $d(o_1) = w(o_1)\phi_c^r$, i.e., o_1 is also a critical cycle under ϕ_c^r. This implies that we can safely remove e_2 and the resulting representation has the same maximum cycle ratio as G_c^r. ■

The next result is a corollary of the previous lemma that provides an upper bound for the number of nonredundant edges in E_c^r.

Corollary 13.1: The number of nonredundant edges in E_c^r is $O(|V|^2 N_{\text{ff}})$, where N_{ff} is the total number of flip-flops in G.

Proof: First of all, for all $e \in E_c^r$, we know that $w(e) \le N_{\text{ff}}$. According to Lemma 13, there are at most N_{ff} nonredundant edges from $u \in V$ to $v \in V$. Therefore, the corollary is true. ■

In practice, the number of flip-flops on an input-to-output path in a cluster is much smaller than N_{ff}, which enables the efficiency of the reduced clustering representation.

In our implementation, we employ another two parameters $\text{pd} : V \to \{R\}$ and $\text{pw} : V \to \{Z\}$ to record the path delays and weights from the inputs of a cluster to its output, respectively. More specifically, we set $\text{pw}(u) = \text{pd}(u) = \emptyset\ \forall u \in V$, before $\mathcal{L}_v(T,\phi)$ is about to be carried out for some $v \in V$. After that, whenever a vertex u is put in c_v', we compute the pd and pw values of its preceding vertices based on $\text{pd}(u)$ and $\text{pw}(u)$, followed by pruning.

IX. Cluster and Replication Reduction

In this section, we briefly review the techniques that were used in [18] and [20] to reduce the number of clusters and vertex replication.

In Section III, we assume that each cluster has one output. If this assumption is relaxed, a post-processing step can be added to reduce the number of clusters. For example, if the arrival time of a vertex $v \in V$ in its own cluster is equal to the arrival time of a copy of v in another cluster, then the entire cluster at v can be removed, and replaced by the copy.

Replicated vertices can also be reduced as follows. If the arrival times of two copies of a vertex differ by an amount greater than or equal to the inter-cluster delay D, then the output of the copy with the smaller arrival time can replace the copy with the

TABLE I
SEQUENTIAL CIRCUITS FROM ISCAS-89

Circuit	$\|V\|$	$\|E\|$	N_{ff}	ϕ_{lb}
s208	104	183	40	10.00
s349	161	285	35	14.00
s420	218	385	84	12.00
s635	286	478	97	66.00
s838	446	789	172	16.00
s1196	529	1024	31	24.00
s1423	657	1170	239	53.00
s1512	780	1286	185	22.50
s3330	1789	2890	435	14.00
s4863	2342	4093	105	30.00
s5378	2779	4262	301	21.00
s9234	5597	6932	333	38.00
s35932	16065	28590	5815	27.00
s38584	19253	33061	7372	48.00

TABLE II
OPTIMAL MAXIMUM-CYCLE RATIO

Circuit	ϕ^{opt}		
	$A = 5\%\|V\|$	$A = 10\%\|V\|$	$A = 20\%\|V\|$
s208	13.00	11.00	10.00
s349	18.00	16.00	14.67
s420	14.00	13.00	12.00
s635	75.00	70.00	68.00
s838	17.00	16.00	16.00
s1196	26.00	25.00	24.00
s1423	55.00	53.00	53.00
s1512	23.78	22.50	22.50
s3330	14.33	14.00	14.00
s4863	30.25	30.00	30.00
s5378	21.00	21.00	21.00
s9234	38.00	38.00	38.00
s35932	27.00	27.00	27.00
s38584	48.00	48.00	48.00

larger arrival time. In addition, there are slacks available for vertices on noncritical paths and their arrival times need not be the least fixpoint. This property can also be used to further remove replicated vertices. Once this reduction of replicated vertices is carried out, there may be clusters that are not completely filled. We can merge some of the clusters, provided that the area bound is not exceeded. Using these techniques, the area overhead can be reduced to 14%, as shown in [10].

It is worthy to point out that it is both unnecessary and memory-wise prohibitive to keep all clusters during clustering under a given ϕ. The cluster rooted at each vertex $v \in V$ is dynamically built during the execution of procedure $\mathcal{L}_v(T, \phi)$ and released when the procedure finishes. The existence of such a cluster is implied by the updated arrival time of v. The whole clustered circuit needs to be built only when we want to compute its maximum cycle ratio, or when ϕ^{opt} is found. For the former case, reduced clustering representation helps to manage the storage requirement. For the latter case, a reasonable overhead can be obtained using the aforementioned techniques.

X. EXPERIMENTAL RESULTS

We implemented the algorithm in a PC with a 2.4-GHz Xeon CPU, 512-kB second-level cache memory, and 1-GB RAM. To compare with the algorithm in [20], we used the same test files, which were generated from the ISCAS-89 benchmark suite. For each test case, we introduced a flip-flop with directed edges from each PO to it and from it to each PI so that every PI-to-PO path became a cycle. As in [20], the size and delay of each gate was set to 1, intra-cluster delays were 0, and inter-cluster interconnects had delays $D = 2$. The circuits used are summarized in Table I. We also list the maximum cycle ratio of the circuit before clustering in column ϕ_{lb}, which provides a lower bound of the solution by Lemma 1.

Although theoretically the algorithm in Fig. 7 will reach the exact solution without being provided a precision, we have to consider the impact of floating point error introduced by practical finite precision arithmetic, due to the divisions involved in the maximum-cycle-ratio computation. In our experiments, we set the error to be 0.001. Since $1/(|V|N_{\text{ff}})$ is generally smaller than 0.001, we set the precision of ϕ^{opt} to be 0.01.

For each circuit, we tested three size bounds: A is 5%, 10%, and 20% of the number of gates. The optimal maximum-cycle-ratio for each circuit is shown in Table II.

To illustrate the advantage of our incremental algorithm over binary search, we ran binary search to find the optimal maximum-cycle-ratio using the proposed iterative method as a subroutine for feasibility checking. The lower bound of the binary search was ϕ_{lb} and the upper bound was the maximum-cycle-ratio of the clustering where each vertex itself is a cluster. The binary search precision was also set to be 0.01. We report the results in Table III, where column "#step" lists the number of search steps and column "time(s)" lists the running time in seconds. "BS" refers to the binary search-based algorithm and "INC" refers to our incremental algorithm. Row "arith" ("geo") gives the arithmetic (geometric) mean of the running times. It can be shown that the incremental algorithm is more efficient.

To compare the running time in [20], where the optimal clock period is integral, we set the precision of ϕ^{opt} to be 1 and ran the algorithm again for $A = 5\%|V|$, $10\%|V|$, $20\%|V|$, respectively. The obtained ϕ^{opt} matches the result in [20] for all the scenarios of A. The only running time information given in [20] is the largest running time per step among the three scenarios, which we list in Table IV under column "[20]." We then compute ours in column "ours." Note that the running time from [20] was based on an UltraSPARC 2 workstation.

We observe that, for most of the circuits, our algorithm finds the optimal solution in just a few steps, which is generally less than the number of iterations conducted in a binary search, which are not given in [20].

TABLE III
IMPROVEMENT OVER BINARY SEARCH IN RUNNING TIME

Circuit	$A = 5\%\|V\|$				$A = 10\%\|V\|$				$A = 20\%\|V\|$			
	#step		time(s)		#step		time(s)		#step		time(s)	
	BS	INC	BS	INC	BS	INC	BS	INC	BS	INC	BS	INC
s208	12	4	0.98	0.20	12	4	0.14	0.09	12	3	0.01	0.04
s349	13	14	2.56	1.73	13	21	0.81	0.47	13	14	4.83	1.39
s420	13	3	1.95	0.33	13	3	0.94	0.34	13	2	0.01	0.00
s635	15	4	19.20	5.71	15	4	66.14	8.02	15	4	65.64	6.43
s838	13	3	2.18	0.69	13	2	0.05	0.01	13	2	0.05	0.01
s1196	14	3	8.05	2.86	14	3	21.19	3.40	14	2	0.08	0.02
s1423	15	3	6.68	2.77	15	2	0.43	0.07	15	2	0.47	0.06
s1512	14	15	76.08	166.36	14	8	1.37	0.79	14	8	0.32	0.68
s3330	13	11	16.15	11.34	13	10	0.79	1.82	13	10	0.79	1.82
s4863	14	8	1688.22	133.54	14	5	3.27	3.85	14	5	1.86	3.70
s5378	13	3	2.66	0.94	13	3	1.61	0.73	13	3	1.36	0.73
s9234	14	3	12.86	3.78	14	3	5.86	2.98	14	3	6.97	2.98
s35932	14	4	108.47	48.13	14	4	46.26	47.79	14	4	48.54	46.59
s38584	15	2	104.56	21.92	15	2	47.77	21.90	15	2	58.22	21.46
arith			3.70X	1			3.00X	1			3.09X	1
geo			2.87X	1			2.22X	1			1.78X	1

TABLE IV
RUNNING TIME COMPARISON WITH [20] (IN SECONDS)

Circuit	time/step (s)	
	[20]	ours
s208	0.05	0.00
s349	1.48	0.02
s420	0.25	0.00
s635	0.76	0.03
s838	1.06	0.01
s1196	0.65	0.02
s1423	5.20	0.04
s1512	9.75	0.06
s3330	8.52	0.17
s4863	658.37	1.04
s5378	60.62	0.31
s9234	n/a	1.20
s35932	n/a	11.46
s38584	n/a	11.07

XI. CONCLUSION

Processing rate, defined as the product of frequency and throughput, is identified as an important metric for sequential circuits. We show that the processing rate of a sequential circuit is upper bounded by the reciprocal of its maximum cycle ratio, which is only dependent on the clustering of the circuit. The problem of processing rate optimization is formulated as seeking an optimal clustering with minimal maximum-cycle-ratio in a general graph. An iterative algorithm is proposed that finds the minimal maximum-cycle-ratio under the single-value inter-cluster delay model. Since our algorithm avoids binary search and is essentially incremental, it has the potential to be combined with other optimization techniques, such as gate sizing, budgeting, etc., thus, can be used in incremental design methodologies [8]. In addition, since maximum cycle ratio is a fundamental metric, the proposed algorithm can be adapted to suit other traditional designs.

REFERENCES

[1] L. P. Carloni, K. L. McMillan, A. Saldanha, and A. L. Sangiovanni-Vincentelli, "A methodology for correct-by-construction latency insensitive design," in *Proc. Int. Conf. Comput.-Aided Des.*, 1999, pp. 309–315.

[2] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli, "Latency insensitive protocols," in *Proc. Comput. Aided Verification Conf.*, 1999, pp. 123–133.

[3] M. R. Casu and L. Macchiarulo, "Floorplanning for throughput," in *Int. Symp. Phys. Des.*, 2004, pp. 62–69.

[4] ——, "A new approach to latency insensitive design," in *Proc. Des Autom Conf.*, 2004, pp. 576–581.

[5] C. Chu, E. F. Y. Young, D. K. Y. Tong, and S. Dechu, "Retiming with interconnect and gate delay," in *Proc. Int. Conf. Comput.-Aided Des.*, 2003, pp. 221–226.

[6] P. Cocchini, "Concurrent flip-flop and repeater insertion for high performance integrated circuits," in *Proc. Int. Conf. Comput.-Aided Des.*, 2002, pp. 268–273.

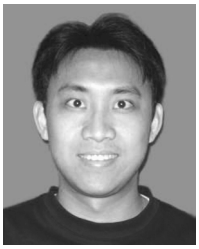
[7] J. Cochet-Terrasson, G. Cohen, S. Gaubert, M. McGettrick, and J.-P. Quadrat, "Numerical computation of spectral elements in max-plus algebra," in *Proc. IFAC Conf. Syst. Struct. Contr.*, 1998, pp. 699–706.

[8] J. Cong, O. Coudert, and M. Sarrafzadeh, "Incremental CAD," in *Proc. Int. Conf. Comput.-Aided Des.*, 2000, pp. 236–244.

[9] J. Cong, T. Kong, and D. Z. Pan, "Buffer block planning for interconnect-driven floorplanning," in *Proc. Int. Conf. Comput.-Aided Des.*, 1999, pp. 358–363.

[10] J. Cong, H. Li, and C. Wu, "Simultaneous circuit partitioning/clustering with retiming for performance optimization," in *Proc. Des. Autom. Conf.*, 1999, pp. 460–465.

[11] T. H. Cormen, C. E. Leiserson, and R. H. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1989.
[12] A. Dasdan, S. S. Irani, and R. K. Gupta, "Efficient algorithms for optimum cycle mean and optimum cost to time ratio," in *Proc. Des. Autom. Conf.*, 1999, pp. 37–42.
[13] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order*. Cambridge, U.K.: Cambridge Univ. Press, 1990.
[14] S. Hassoun and C. J. Alpert, "Optimal path routing in single and multiple clock domain systems," in *Proc. Int. Conf. Comput.-Aided Des.*, 2002, pp. 247–253.
[15] C. E. Leiserson, F. M. Rose, and J. B. Saxe, "Optimizing synchronous circuitry by retiming," in *Proc. Adv. Res. VLSI: Proc. 3rd Caltech Conf.*, 1983, pp. 86–116.
[16] C. Lin and H. Zhou, "Wire retiming as fixpoint computation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 12, pp. 1340–1348, Dec. 2005.
[17] ——, "Optimal wire retiming without binary search," *IEEE Trans. Comput.-Aided Des. Integr. Syst.*, vol. 25, no. 9, pp. 1577–1588, Sep. 2006.
[18] R. Murgai, R. Brayton, and A. Sangiovanni-Vincentelli, "On clustering for minimum delay/area," in *Proc. Int. Conf. Comput.-Aided Des.*, 1991, pp. 6–9.
[19] V. Nookala and S. S. Sapatnekar, "A method for correcting the functionality of a wire-pipelined circuit," in *Proc. Des. Autom. Conf.*, 2004, pp. 570–575.
[20] P. Pan, A. K. Karandikar, and C. L. Liu, "Optimal clock period clustering for sequential circuits with retiming," *IEEE Trans. Comput.-Aided Des.*, vol. 17, no. 6, pp. 489–498, Jun. 1998.
[21] D. K. Y. Tong and E. F. Y. Young, "Performance-driven register insertion in placement," in *Proc. Int. Symp. Phys. Des.*, 2004, pp. 53–60.
[22] H. Zhou and C. Lin, "Retiming for wire pipelining in system-on-chip," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 23, no. 9, pp. 1338–1345, Sep. 2004.



Chuan Lin received the B.S. degree in electrical engineering from Tsinghua University, Beijing, China, in 2002, and the Ph.D. degree in computer engineering from Northwestern University, Evanston, IL, in 2006.

He is a Member of the Technical Staff in Magma Design Automation Inc., Santa Clara, CA. His research interests include VLSI computer-aided design (CAD), especially deep submicrometer physical design.

Jia Wang received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 2002, and is currently pursuing the Ph.D. in computer engineering at Northwestern University, Evanston, IL.

His research interests include VLSI computer-aided design (CAD), especially algorithm design.

Hai Zhou (SM'04) received the B.S. and M.S. degrees in computer science and technology from Tsinghua University, Beijing, China, in 1992 and 1994, respectively, and the Ph.D. degree in computer sciences from The University of Texas at Austin, in 1999.

He is an Assistant Professor of Electrical and Computer Engineering at Northwestern University, Evanston, IL. Before he joined the faculty of Northwestern University, he was with the Advanced Technology Group, Synopsys Inc., Mountain View, CA. His research interests include VLSI computer-aided design (CAD), algorithm design, and formal methods.

Prof. Zhou served on the Technical Program Committees of the ACM International Symposium on Physical Design and the IEEE International Conference on Computer-Aided Design. He was a recipient of the CAREER Award from the National Science Foundation in 2003.