Theory of Parallel Hardware March 28, 2004
Massachusetts Institute of Technology 6.896
Charles Leiserson, Michael Bender, Bradley Kuszmaul Handout 11

Solution Set 5

Due: In class on Wednesday, March 17. Starred problems are optional.

Problem 5-1. For a circuit $G = (V, E, d, w)$, define the ***size*** of G as

$$|G| = \sum_{e \in E} w(e) .$$

Show that the problem of determining a retiming $r : V \to \mathbb{Z}$ such that $|G_r|$ is minimized can be reduced to a linear-programming problem. What about minimizing $|G_r|$ such that $\Phi(G_r) \le c$ for a given $c > 0$?

Solution: In a general form, the linear programming problem is that given a matrix A and vectors b and c, we want to find the vector x that maximize $c \cdot x$ such that $Ax \le b$.

For a circuit $G = (V, E, d, w)$, we can reduce the problem of determining a retiming $r : V \to \mathbb{Z}$ such that $|G_r|$ is minimized to a linear programming problem. Our A matrix just contains the constraints that make r a valid retiming. Specifically, the matrix A contains $|V|$ columns—one for each vertex in V. For each edge in $u \xrightarrow{e} v \in E$, we need the constraint $r(u) - r(v) \le w(e)$. Thus, we add the row with a 1 in the column for u, a -1 in the column for v, and 0s in every other column. The corresponding row in the b vector receives $w(e)$. The x vector contains $|V|$ elements—the result is a retiming for each vertex in V.

Now, in addition to the constraints, we also want to minimize

$$|G_r| = \sum_{e \in E} w_r(e) = \sum_{e \in E} w(e) - r(u) + r(v) .$$

We need to be able to represent this minimization problem in terms of the vector c, which has size $|V|$. Since $w(e)$ is fixed no matter what retiming we use, we really want to minimize $\sum_{e \in E} r(v) - r(u)$. Notice that we subtract $r(u)$ once for each edge from u. Similarly, we add $r(v)$ once for each going to v. Thus, each element of the c vector is just the in-degree minus the out-degree of a vertex, and we have our linear programming problem. [1]

If we want to minimize $|G_r|$ such that $\Phi(G_r) \le c$ for a given $c > 0$, then we simply need to add a second set of contraints (as we did in class). For all $u, v \in V$ such that $D(u, v) > c$, we add the constraint that $r(u) - r(v) \le W(u, v) - 1$. As before, this means that we add a row to A with a 1 in the column for u, a -1 in the column for v, and we put $W(u, v) - 1$ in the corresponding row in the b vector. Our minimization vector c remains the same.

Problem 5-2. Recall that a ***c-slow*** circuit has c equivalence classes of computation that do not interact. For example, systolic conversion typically produces 2-slow systolic circuits. Describe an efficient algorithm that, given a circuit $G = (V, E, w)$, determines whether G is a c-slow version of some other circuit $G' = (V, E, w')$ for some $c > 1$, that is, that there exists a retiming $r : V \to \mathbb{Z}$ such that $G_r = cG'$. Give an efficient algorithm to produce a G' with maximum c. Analyze your algorithms.

Solution: Given a circuit $G = (V, E, w)$ and some value c with $c > 1$, we want to determine whether G is a c-slow version of some other circuit $G' = (V, E, w')$. Our algorithm determines whether there is some retiming G_r of G such that each edge weight $w_r(e) \equiv 0 \pmod{c}$.

For our algorithm, we begin at any random vertex $s \in V$ (s would generally be the Host as there should be a path from the host to everything and everything to the host for output). We assign this vertex a lag of $r(s) = 0$. We add s to the set of vertices that we've visited (at the beginning, it's just s). Next, we choose a new edge e connecting a vertex u

[1]Note that we are trying to minimize $c \cdot x$, but the way i stated the general linear programming model, we want to maximize $c \cdot x$. We simply have to negate the c vector.

that we've visited to some vertex v. We could have visited v before, or v could be a new vertex.
Case 1. Suppose v is a new vertex. We add v to the logical set of vertices that we've visited. We want to assign v a lag such that we can make $w_r(e) \equiv 0 \pmod{c}$. Specifically, for convenience, we make $w_r(e) = 0$. If e goes from u to v, then we make $w(e) - r(u) + r(v) = 0$, or $r(v) = r(u) - w(e)$. Similarly, if e goes from v to u, then $w(e) - r(v) + r(u) = 0$.
Case 2. Suppose we have visited v before. Then we have already assigned v a lag $r(v)$, and we do not want to change it. Instead, we check whether this edge could work with the assigned lag.[2] That is, if e goes from u to v, then we check that $w_r(e) = w(e) - r(u) + r(v) \equiv 0 \pmod{c}$. Similarly, if e goes from v to u, then we check that $w(e) - r(v) + r(u) \equiv 0 \pmod{c}$. If $w_r(e) \equiv 0 \pmod{c}$, then we continue executing. Otherwise, we return an answer indicating that there G is not a c-slow version of any circuit G'.
Whether we fall into case 1 or case 2, we repeat by choosing another new edge (as before) until we've followed every edge.[3] If the algorithm runs to completion (there were no rejections in case 2), then we return that G is a c-slow version of some graph G'.

Before we get to correctness, let us look at running time of this algorithm. We follow each edge exactly once. We also have to assign each vertex a lag once. Thus, the running time is $\Theta(V + E)$. Since we assume that each vertex has a path back to the host, $E > V$, so the running time is $\Theta(E)$.

Now, we look at correctness. Notice that in our algorithm, we can introduce negative edge weights. We first show that we can map a solution with negative edge weights to a solution without negative edge weights.

Lemma 1 *Suppose that we have a retiming $r : V \to \mathbb{Z}$ of G such that each edge has a weight that is a multiple of c, with $c > 1$. Then the retiming $r' : V \to \mathbb{Z}$, defined by $r'(v) \equiv r(v) \pmod{c}$ for all $v \in V$, contains only non-negative edge weights that are multiples of c.*

Proof. First, we show that the edge weights given by $w_{r'}$ are multiples of c. By supposition, $w_r(e)$ is a multiple of c for all $e \in E$. That is, $w_r(e) = w(e) - r(u) + r(v) \equiv 0 \pmod{c}$ for $u \xrightarrow{e} v$. Since the mod operation is associative over addition and subtraction, it follows that $w_{r'}(e) = w(e) - (r(u) \pmod{c}) + (r(v) \pmod{c}) \equiv w(e) - r(u) + r(v) \pmod{c} \equiv 0 \pmod{c}$.

Next, we show that the retiming r' is valid—it contains only non-negative edge weights. The value $r(u) \pmod{c}$ must be between 0 and $c - 1$. Thus, $-c < -(r(u) \pmod{c}) + (r(v) \pmod{c}) = -r'(u) + r'(v) < c$. Since $w(e) \geq 0$, it follows that $w_{r'}(e) = w(e) - r'(u) + r'(v) > -c$. Given that $w_{r'}(e)$ is a multiple of c, we have $w_{r'}(e) \geq 0$.

Lemma 2 *Suppose that there are two paths, $u \overset{p_1}{\leadsto} v$ and $u \overset{p_2 2}{\leadsto} v$, between vertices u and v in the circuit G. Suppose also that the total weights of the paths, $w(p_1)$ and $w(p_2)$ are such that $w(p_1) \not\equiv w(p_2) \pmod{c}$. Then there is no retiming such that G has only edge weights that are multiples of c.*

Proof. Assume for the sake of contradiction that $w(p_1) \not\equiv w(p_2)$, but there is a retiming G_r of G with only edge weights that are multiples of c. The total weight of each retimed path is just $w_r(p_i) = w(p_i) - r(u) + r(v)$. Since all retimed edge weights are multiples of c, it follows that $w_r(p_i) \equiv 0 \pmod{c}$. So we have $w(p_i) - r(u) + r(v) \equiv 0 \pmod{c}$. Thus, $w(p_1) - r(u) + r(v) \equiv w(p_2) - r(u) + r(v) \pmod{c}$, which reduces to $w(p_1) \equiv w(p_2)$, contradicting the assumption.

Theorem 3 *Our algorithm works (it claims that G is a c-slow version of some circuit G' if and only if it is).*

[2] Note that u must have already been assigned a lag $r(u)$, as we chose the edge e connecting to a visited vertex u.
[3] If there are still edges remaining but none of them touch vertices we have visited, then we would start over by choosing a new random vertex and assigning it a lag of 0. In reality, this situation should not occur—it would mean that there is a part of the circuit that is in no way connected to another part of the circuit, which would be a very silly circuit design.

Proof. ($\Rightarrow$) Suppose our algorithm claims that G is a c-slow version of some circuit G'. We followed every edge at some point. Each time we followed an edge $u \xrightarrow{e} v$, we hit one of two cases:
Case 1. We had not visited v before. Thus, we assigned edge the weight $w(e) = 0$. Obviously $0 \equiv 0 \pmod{c}$.
Case 2. We had visited v before, but we did not reject the circuit. Then $w_r(e) = w(e) - r(u) + r(v) \equiv 0 \pmod{c}$ (otherwise, we would have rejected).
Thus, we computed a retiming such that every edge has a weight that is a (possibly negative) multiple of c. Applying Lemma 1, there is a valid (non-negative) retiming of G with edge weights that are multiples of c. Thus, G is a c-slow version of some circuit G'.

($\Leftarrow$) Suppose that our algorithm rejects the circuit (our algorithm claims that there is no G' such that G is a c-slow version of G'. Then let us consider the point at which we rejected. We must have been in case 2 for some edge $u \xrightarrow{e} v$. Notice that the $r(w)$ is only computed and set for a vertex w when following case 1. Thus, there must have been some path $s \overset{p_1}{\leadsto}$ by which we computed lags using only case 1s. Notice also that the lag we calculate is just the sum of the edge weights (or, rather, the negation of the sum). It follows that $w(p_1) = -r(v)$. Similarly, for the path $s \overset{p_2}{\leadsto} v = s \overset{p_2'}{\leadsto} u \xrightarrow{e} v$, the original weight of the path is $w(p_2) = -r(u) + w(e)$. If we reject at edge e, then $w(e) - r(u) + r(v) \not\equiv 0 \pmod{c}$. We know that $r(v) = w(p_1) \equiv 0 \pmod{c}$ and $r(u) \equiv 0 \pmod{c}$ (or we would have rejected earlier), so it follows that $w(e) \not\equiv 0 \pmod{c}$. Thus, $w(p_1) \not\equiv w(p_2) \pmod{c}$. Applying Lemma 2, there is no possible solution, so our algorithm was right to reject.

We can modify our algorithm slightly to produce a G' to maximize c. Notice that it is trivial to produce a G' with the original algorithm—we just need to store the lags at each vertex and use these lags to compute the edge weights.

The tricky part is finding a maximum c. We keep an extra value m that indicates the maximum value we can still get for c (in the beginning, this value is infinite). We update this m value every time we enter case 2. We can no longer reject the circuit by computing $w(e) - r(u) + r(v) \pmod{c}$, as we do not know c. Instead, we can reduce the range of feasible values for c by taking into account the constraint $w(e) - r(u) + r(v) \equiv 0 \pmod{c}$—$c$ must divide $w(e) - r(u) + r(v)$. We can now limit c to be the greatest common factor of $w(e) - r(u) + r(v)$ and m. If m ever drops to 1, we reject the circuit and say there are no c-slow circuits for $c > 1$. Otherwise, we end by setting $c = m$.

The correctness proofs are similar to the problem with a given c. Running time is also similar in that we need to traverse $\Theta(E)$ edges, but running time also depends on the running time of the greatest common divisor. Each $\gcd(a, b)$ computation takes something like $O(\max(a, b))$ time, and one number could be as large as the entire graph, so the total running time is something like $O(E \lg |G|)$.