```
clear; clc;
n = 5;
E = [0 1 0 0 0; 0 0 1 0 0; 0 0 0 0 1; 1 1 1 0 0; 0 0 0 1 0];
W = [0 0 0 2 0; 4 0 0 2 0; 4 3 0 2 0; 2 1 0 0 0; 4 3 2 2 0];
D = [6 11 15 25 19; 25 5 9 19 13; 20 19 4 14 8; 12 11 10 6 14; 16 15 14 10 4];
R = zeros(0,n+2);
d = zeros(1,0);
aseen = 0;

% sort candidate alphas
for sorted = sort(sort(D,2,"ascend"),1,"ascend")
    d = [d; sorted];
end

% iterate over only unique alphas
for a = transpose(d)
    if aseen == a
        continue
    end
    aseen = a;

    A = zeros(0,n+1);
    b = zeros(0);
    for r = 1:n+1
        for c = 1:n
            if r == n+1
                row = zeros(1,n+1);
                row(c) = 1;
                row(n+1) = -1;
                A = [A; row];
                b = [b; 0];
            else
                if E(r,c) == 1
                    row = zeros(1,n+1);
                    row(r) = 1;
                    row(c) = -1;
                    A = [A; row];
                    b = [b; W(r,c)];
                end
                if D(r,c) > a
                    row = zeros(1,n+1);
                    row(r) = 1;
                    row(c) = -1;
                    A = [A; row];
                    b = [b; W(r,c)-1];
                end
            end
        end
    end
```

```
    [dist, pred, error] = bellman_ford(A, b);

    % only consider legal retiming solution
    if ~error
        R = [R; [a, dist]];
    end
end
fprintf('First column is legal alphas\nSecond column shows number of FF\nLast n ↙
columns are r values for nodes 1 to n\n');

num_regs = zeros(0,1);
for e = 1:size(R,1)
    REG = E.*W;
    for u = 1:n
        for v = 1:n
            REG(u,v) = REG(u,v) - R(e,u+1) + R(e,v+1);
        end
    end
    num_regs = [num_regs sum(REG.*E,"all")];
end

[R(:,1), transpose(num_regs), R(:,2:(size(R,2)-1))]

fprintf("Min alpha solution: %d with %d flip flops\n", R(1,1), num_regs(1));

function [dist, pred, error] = bellman_ford(graph, w)
    % graph is constraint graph including start
    % w is edge weights of constraing graph
    num_edges = size(graph,1);
    num_nodes = size(graph,2);
    dist = inf(1, num_nodes);
    pred = zeros(1, num_nodes);
    dist(num_nodes) = 0;
    error = false;

    for i = 1:num_nodes-1
        for edge = 1:num_edges
            for u = 1:num_nodes
                for v = 1:num_nodes
                    if graph(edge,v) == 1 && graph(edge,u) == -1 ...
                       && dist(u) + w(edge) < dist(v)
                        dist(v) = dist(u) + w(edge);
                        pred(v) = u;
                    end
                end
            end
        end
    end
```

```
    % Check for negative cycles
    for edge = 1:num_edges
        for u = 1:num_nodes
            for v = 1:num_nodes
                if graph(edge,v) == 1 && graph(edge,u) == -1 ...
                   && dist(u) + w(edge) < dist(v)
                    % error('Graph contains a negative cycle');
                    error = true;
                end
            end
        end
    end
end
```