第 2 章迭代边界

郭一隆

March 13, 2018

1 整体思路

1.1 LPM 算法实现

利用 `MATLAB` 实现 LPM 算法，以 `d`（延迟节点数目）和 $L^{(1)}$ 作为输入，详见附录 `LPM.m`。
注：将 LPM 算法中 -1 的定义进一步修改为 $-\infty$，方便 MATLAB 运算。

1.2 $L^{(1)}$ 求解

- 对于第 1 题、第 3 题，DFG 结构较简单，$L^{(1)}$ 可直接得到；
- 对于第 5 题，DFG 结构复杂，用递归的动态规划算法求解任意两个延迟节点之间的最长路径（不经过其他延迟节点），从而获得 $L^{(1)}$，详见附录 `p5_graph.m`。

1.3 L 序列求解

利用 $L^{(m)}$ 与 $L^{(\lfloor m/2 \rfloor)}$、$L^{(\lceil m/2 \rceil)}$ 的关系进行递归求解，降低复杂度。

2 问题求解

2.1 PROBLEM 1

输入：

- $d = 3$

- $L^{(1)} = \begin{bmatrix} -\infty & 0 & -\infty \\ 7 & -\infty & 3 \\ 3 & -\infty & -\infty \end{bmatrix}$

运行 `iteration_bound.m` 可得输出:

- `iteration_bound = 3.5`

2.2 Problem 3

输入:

- $d = 6$
- $L^{(1)} = \begin{bmatrix} 4 & 4 & 4 & -\infty & 4 & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & 0 & -\infty & -\infty \\ 4 & 4 & 4 & -\infty & 4 & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty & 0 \\ -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \end{bmatrix}$

运行 `iteration_bound.m` 可得输出:

- `iteration_bound = 4`

2.3 Problem 5

运行 `p5_graph.m` 可得输入:

- $d = 7$
- $L^{(1)} = \begin{bmatrix} 16 & 16 & -\infty & -\infty & 16 & -\infty & -\infty \\ -\infty & 4 & -\infty & -\infty & -\infty & -\infty & -\infty \\ 15 & 15 & -\infty & -\infty & 15 & -\infty & -\infty \\ 14 & 14 & -\infty & -\infty & 14 & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & 4 & -\infty & -\infty \\ 14 & 14 & -\infty & -\infty & 14 & -\infty & -\infty \\ 14 & 14 & -\infty & -\infty & 14 & -\infty & -\infty \end{bmatrix}$

  (按图中自左到右顺序依次将 D 节点编号为 D_1 到 D_7)

运行 `iteration_bound.m` 可得输出:

- `iteration_bound = 16`

A. 附录

所有源码已上传至 `https://github.com/Nuullll/iteration-bound`

ITERATION_BOUND.M

```matlab
% Iteration bound
clear all;

%% Define the input
%%%%%%%%%%%%%%%% problem 1 %%%%%%%%%%%%%%%
d = 3;  % 3 delay nodes
L1 = [-inf, 0, -inf;
      7, -inf, 3;
      3, -inf, -inf];
%%%%%%%%%%%%%%%% problem 3 %%%%%%%%%%%%%%%
d = 6;
L1 = [4, 4, 4, -inf, 4, -inf;
      -inf, -inf, -inf, -inf, -inf, -inf;
      -inf, -inf, -inf, 0, -inf, -inf;
      4, 4, 4, -inf, 4, -inf;
      -inf, -inf, -inf, -inf, -inf, 0;
      -inf, -inf, -inf, -inf, -inf, -inf];
%%%%%%%%%%%%%%%% problem 5 %%%%%%%%%%%%%%%
p5_graph;   % run script to get L1 for problem 5

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
memo{1,d} = [];     % define memo cell
memo{1} = L1;

%% Calculate all Ls
for k = 1:d
    [~,memo] = LPM(memo,k);
end

%% Get iteration bound
bound = getIterationBound(memo);

display(bound);
```

P5_GRAPH.M

```matlab
% convert graph of problem 5 into a matrix
%% Define the graph
Nlogic = 34;    % 34 logic nodes
times = ones(1,34);     % computing time of each logic node
times([5,10,14,21,22,27,31,34]) = 2;    % multiply nodes

```

```
A = zeros(Nlogic);  % adjacent matrix
edges = sub2ind(size(A),...
    [1,2,2,2,3,5,6,6,6,7,7,7,8,9,10,11,11,12,14,15,15,15,16,17,17,17,18,18,...
    18,20,21,22,23,24,24,24,25,25,25,26,27,28,28,29,31,32,34],...
    [4,3,6,7,5,1,3,4,9,8,15,16,10,11,6,13,14,13,12,8,9,20,17,20,21,22,19,23,...
    26,19,15,18,27,17,18,23,26,32,33,28,25,30,31,30,29,34,33]);
A(edges) = 1;

%% Find longest path for any given two nodes
% preprocess dp_path_map
dp_path_map = diag(times);          % 0 is the flag of NOT-CALCULATED,
                                    % path from node to itself is its computing
                                    % time

%% Calculate L1 matrix for problem 5
d = 7;
L1 = -inf*ones(d);

start_nodes_cell = {[2],[11,12],[7],[16],[28,29],[24],[24,25,32]};
end_nodes_cell = {[4],[12],[13],[19],[29],[30],[33]};

for i = 1:d
    for j = 1:d
        for start_node = start_nodes_cell{i}
            for end_node = end_nodes_cell{j}
                [path_length, dp_path_map] = findLongestPath(start_node, end_node,...
                    dp_path_map, A, times);
                L1(i,j) = max(L1(i,j), path_length);
            end
        end
    end
end
```

findLongestPath.m

```
function [ path_length, dp_path_map ] = findLongestPath( src, dest, dp_path_map, adjacent, node_times )
% Find the longest path from node src to node dest
% Using dynamic programming, which is NOT the best algorithm,
% but it's best for programmers.
%
% src   [int]   source node
% dest  [int]   destination node
% dp_path_map   [2-d matrix]    stores calculated path
% adjacent      [2-d matrix]    adjacent matrix of the graph
```

```
% node_times      [vector]          computing time of each node

path_length = dp_path_map(src,dest);

if path_length ~= 0
    % return calculated value
    return
end

s_adjs = find(adjacent(src,:)==1);
if isempty(s_adjs)
    % source node has no adjacent nodes
    path_length = -inf;
    dp_path_map(src,:) = -inf;
    return
end

path_length = -inf;
for v = s_adjs
    [sub_length, dp_path_map] = findLongestPath(v,dest,dp_path_map,
        adjacent,node_times);
    path_length = max(path_length, node_times(src) + sub_length);
end
dp_path_map(src,dest) = path_length;

end
```

LPM.M

```
function [ Lm, memo ] = LPM( memo, m )
% Calculate L^(m) and update to memo cell
% memo   [cell]   stores the calculated Ls
% m      [int]    order

Lm = memo{m};

if ~isempty(Lm)
    % return result in memo directly
    return
end

[Lm1, memo] = LPM(memo, floor(m/2));
[Lm2, memo] = LPM(memo, ceil(m/2));

d = size(Lm1,1);
Lm = zeros(d);
for i = 1:d
    for j = 1:d
        Lm(i,j) = max(Lm1(i,:) + Lm2(:,j).');
```

```
        end
end

% update to the memo cell
memo{m} = Lm;

end
```

getIterationBound.m

```
function [ iteration_bound ] = getIterationBound( memo )
% Calculate iteration bound from a list of Ls

d = length(memo);
bounds = zeros(d);

for k = 1:d
        L = memo{k};
        bounds(:,k) = diag(L)/k;
end

iteration_bound = max(max(bounds));

end
```