Lab09-Network Flow

CS214-Algorithm and Complexity, Xiaofeng Gao & Lei Wang, Spring 2021.

∗ If there is any problem, please contact TA Yihao Xie.
∗ Name: Log Creative Student ID: Email: logcreative-lzl@sjtu.edu.cn

1. Consider there is a network consists n computers. For some pairs of computers, a wire i exists in the pair, which means these two computers can communicate with each other. When a signal passes through the wires, the noise in the signal will be amplified. If you know the magnification rate of noise $m_{i,j}$ of each wire (which must be greater than 1). Design an algorithm to find the route for each other computer to send signals to the computer v with the minimum total magnification rate of noise and analyze the time complexity.

Solution. To solve the problem, the model has to be first simpilified.

Lemma 1 (No loop)**.** *There is no loop on the optimized route.*

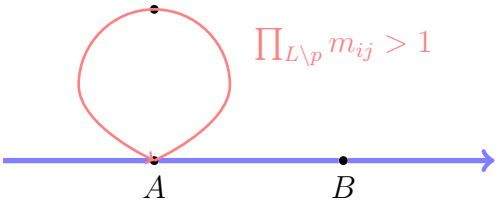

Figure 1: A loop.

Proof of Lemma 1. Prove by contradiction. Suppose there is a loop on the optimized route. Because the maginification on each wire $m_{ij} > 1$, shown in Figure 1, if their is a loop L between node A to node B, a better way is to just choose the path p directly from A to B,

$$\prod_{m_{ij}\notin L} m_{ij} \prod_{m_{ij}\in L} > \prod_{m_{ij}\notin L} m_{ij} \prod_{m_{ij}\in p} m_{ij} \cdot 1$$

This will make the following lemma available. □

Lemma 2 (Computer to computer)**.** *It is equivilent to consider the minimized maginification path between computer and the other computer.*

Proof of Lemma 2. Suppose there are many wires between the two computers A and B, then it is always the optimum to choose the minimum one for A and B. The bigger the wire maginification is, the bigger the total maginification.

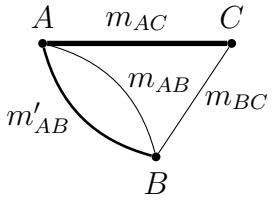

Figure 2: A better wire.

And now consider three computers A, B, and C, shown in Figure 2. A bigger path between A and B will not be selected as the part of the minimized path between A and C. Because a better wire could be used between A and B to get a lower minimized maginification.

$$m'_{AB} \cdot m_{BC} > m_{AB} \cdot m_{BC}$$

Hence, the network of wires and nodes could be simpilified to a network with only computers and their minimum maginification inbetween. If there is no direct wire between two computers, it will be marked as ∞ temporarily. □

The graph will be described as the adjecent matrix G, where

$$G[i][j] = \begin{cases} \min m_{ij}, & \text{the minimum maginification between computer } i \text{ and } j, \\ \infty, & \text{if there is no direct connection between } i \text{ and } j \end{cases}$$

Now, a similar Dijkstra's algorithm will be applied, shown in Algorithm 1.

Algorithm 1: Find optimized route in a network

Input: Graph adjecent matrix G, the ending point v
Output: Route array for every other computer.

```
1  M = {m_i}_{i=1}^{|V|} ;                                  /* Initialize maginification */
2  Π = {π_i}_{i=1}^{|V|};
3  for i ← 1 to |V| do
4      M[i] ← ∞,Π[i] ←NULL;
5  M[v] ← 1;
6  Q ← ∅ ;                                                  /* Initialize the min-heap */
7  for i ← 1 to |V| do
8      Q.INSERT(⟨i, M[i]⟩);
9  while Q ≠ ∅ do
10     ⟨u, M[u]⟩ ← Q.EXTRACT-MIN();
11     for j ← 1 to n do
12         if G[u][j] = ∞ then continue;
13         if M[u] × G[u][j] < M[j] then
14             M[j] ← M[u] × G[u][j] ;                      /* Relaxation */
15             Π[j] ← u;
16             if Q.FIND(j) then
17                 Q.UPDATE(j, M[j]);
18 for i ← 1 to |V| do
19     R[i] ← i ;                                           /* Print routes */
20     p ← i;
21     while p ≠ v do
22         p ← Π[p];
23         if p =NIL then
24             R[i] ←NIL ;                                  /* No route */
25             break;
26         R[i] ← (R[i] → p);
```

The time complexity of the algorithm is $O(V^2 + E \log V)$:

(a) Initialize G takes $O(V^2)$.

(b) Relaxation takes $O(E \log V)$.

(c) Print route takes $O(V^2)$.

Because the relaxation process statisfies the triangle inequality of

$$M[j] \leq M[u] \times G[u][j]$$

with the destination given for calculating M. Thus, it is the same proof as Dijkstra's algorithm. And Lemma 1 will make sure that the printing will eventually meet an end with no loop. □

2. Suppose that we wish to maintain the transitive closure of a directed graph $G = (V, E)$ as we insert edges into E. That is, after each edge has been inserted, we want to update the transitive closure of the edges inserted so far. Assume that the graph G has no edges initially and that we represent the transitive closure as a boolean matrix.

   (a) Show how to update the transitive closure of a graph $G = (V, E)$ in $O(V^2)$ time when a new edge is added to G.

   Solution. Assume that (a, b) is inserted to G. The updated closure item will be $v_x \rightsquigarrow v_a \to v_b \rightsquigarrow v_y$ for every pair (v_x, v_y). Before updating, the closure has already been set properly, thus the updated closure can be described as

   $$t'_{xy} = t_{xy} \vee (t_{xa} \wedge t_{by})$$

   It could be summarized as Algorithm 2, which has the time complexity of $O(V^2)$. □

Algorithm 2: Update Transitive-Closure

Input: Previous Closure $T = (t_{ij})_{|V|\times|V|}$ and newly added edge (a, b)
Output: Updated Closure $T' = (t'_{ij})_{|V|\times|V|}$

```
1 T' ← (t'_ij)_{|V|×|V|};
2 for i ← 1 to |V| do
3     foreach j ← 1 to |V| do
4         t'_xy ← t_xy ∨ (t_xa ∧ t_by);
5 return T';
```

   (b) Give an example of a graph G and an edge e such that $\Omega(V^2)$ time is required to update the transitive closure after the insertion of e into G, no matter what algorithm is used.

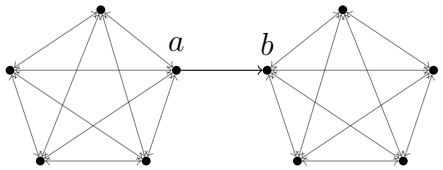

Figure 3: An example requires $\Omega(|V|^2)$

   Solution. An example comes to G is consisted with two unconnected strong connected components. With the bridge (a, b) between the two is applied, all the closure state from the first SCC to the second SCC should be changed. The amount of changes is $\frac{|V|}{2} \times \frac{|V|}{2} = O(|V|^2)$, shown in Figure 3. □

(c) Describe an efficient algorithm for updating the transitive closure as edges are inserted into the graph. For any sequence of m insertions, your algorithm should run in total time $\sum_{i=1}^{m} t_i = O(V^3)$, where t_i is the time to update the transitive closure upon inserting the ith edge. Prove that your algorithm attains this time bound.

Solution. One by one insertion.

The following algorithm also has the time complexity of $O(V^3)$ and it is not nessesary to consider adding edges one by one. The transitive closure $G^* = (V, E^*)$ is constructed by putting edge (i, j) into E^* if and only if $t_{ij}^{(n)} = 1$.

$$t_{ij}^{(0)} = \begin{cases} 0, & \text{if } i \neq j \text{ and } (i,j) \notin E \\ 1, & \text{if } i = j \text{ or } (i,j) \in E \end{cases}$$

For $k \geq 1$,

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)} \right)$$

This process will be terminated when $k > |V|$. And like Folyd-Warshall's Algorithm, all the path with the length under or equal to $|V|$ (assume all the path is weighing 1) is accessed. The proof is similar by replacing min as $\vee$ and $+$ as $\wedge$.

Algorithm 3: TRANSITIVE-CLOSURE

Input: Graph $G = (V, E)$, E has already contained m newly added edges
Output: Transitive Closure of G

```
1  T^(0) ← (t_ij^(0))_{|V|×|V|};
2  for i ← 1 to |V| do
3      for j ← 1 to |V| do
4          if i = j or (i, j) ∈ E then
5              t_ij^(0) = 1;
6          else t_ij^(0) = 0 ;
7  for k ← 1 to |V| do
8      T^(k) ← (t_ij^(k))_{|V|×|V|};
9      for i ← 1 to |V| do
10         for j ← 1 to |V| do
11             t_ij^(k) ← t_ij^(k−1) ∨ (t_ik^(k−1) ∧ t_kj^(k−1));
12 return T^(|V|);
```

This algorithm 3 has time bound of $\Theta(V^3)$.

□

3. An $n \times n$ grid is an undirected graph consisting of n rows and n columns of vertices, as shown in Figure 4. We denote the vertex in the ith row and the jth column by (i, j). All vertices in a grid have exactly four neighbors, except for the boundary vertices, which are the points (i, j) for which $i = 1, i = n, j = 1$, or $j = n$. Given $m \leqslant n^2$ starting points $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$ in the grid, the escape problem is to determine whether or not there are m vertex-disjoint paths from the starting points to any m different points on the boundary such that every vertex in V is included in at most one of the m paths. For example, the grid in Figure 4(a) has an escape, but the grid in 4(b) does not.

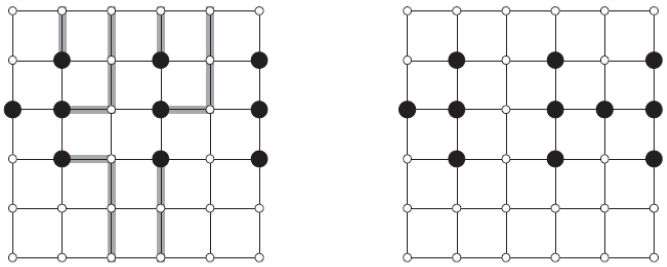

Figure 4: Grids for the escape problem. Starting points are black, and other grid vertices are white. (a) A grid with an escape, shown by shaded paths. (b) A grid with no escape.

(a) Consider a flow network in which vertices, as well as edges, have capacities. That is, the total positive flow entering any given vertex is subject to a capacity constraint. Show that determining the maximum flow in a network with edge and vertex capacities can be reduced to an ordinary maximum-flow problem on a flow network of comparable size. That is, the sizes of the two graph are in the same order of magnitude.

Solution. Split the capacity-constraint vertex into two vertices with edges inbetween. The capacity of the edge is the capacity c of the original vertex. And the splited vertices are v_{in} and v_{out}, connecting the in-coming edges and out-going edges seperately, shown in Figure 5.

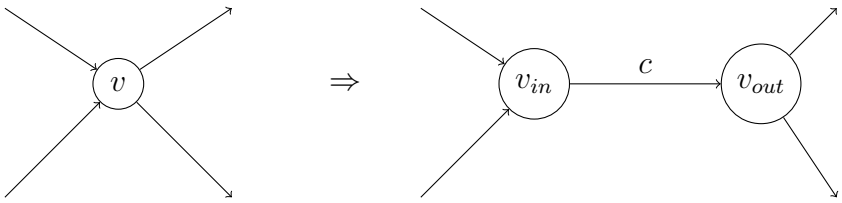

Figure 5: Vertex split in a directed graph

As for the undirected graph, the vertices will be splitted any way. But it will be transformed into directed graph by adding more edges, shown in Figure 6. The constraint between v_{in} and v_{out} will be the bottleneck even though there are a lot of flow coming into v_{in} and out from v_{out}.

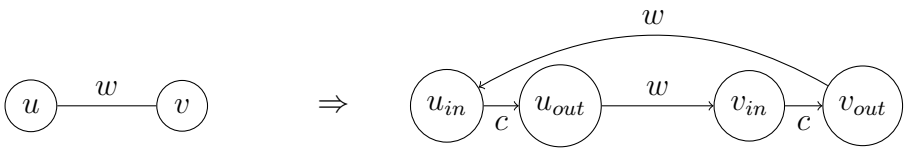

Figure 6: Vertex split in an undirected graph

Thus, the vertex magnitude of the reduced graph is $2|V|$, which is the same level as $O(|V|)$. The edge magnitude is at most

~~$2|E| + 2|E|$~~

. Because the split on the vertex is only trigged when there is an edge between the two vertices. Then, the edge has the same level as $O(|E|)$.

□

(b) Describe an efficient algorithm to solve the escape problem, and analyze its running time.

Solution. The spliting thought could be implemeted in this problem as to plug in two artifical vertices into the graph to constraint the capacity.

Algorithm 4: Abstracted Solution to Escape Problem

1 Source vertices now share the same source vertex s to get the flow in, shown in Figure 7;
2 Replace each edge (u, v) in the original graph with edges from u_{out} to v_{in} and from v_{out} to u_{in} of capacity 1, which is similar to Figure 6;
3 Target vertices v_{out} now share the same target vertex v to get the flow out;
4 Apply Ford-Fulkerson's algorithm on this problem to find the maximum flow from s to t;
5 Determine whether the maximum flow is m;

$O(mn^2)$

~~The time complexity of Algorithm 4 is $O(2|V|\cdot 4|E|\cdot 1) = O((2n)^2 \cdot 4 \times 2n(n+1)) = O(n^4)$.~~

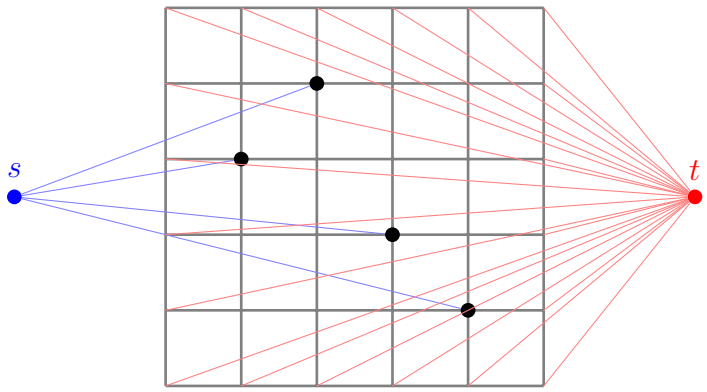

Figure 7: With artificial vertices added

Firstly, the maximum flow could not exceed m because there are only m edges connecting to s. Then, if the maximum flow is less than m, then there is no such solution to the original escape problem, because the escape problem requires m total flow.

If the maximum flow is m, then their is a solution to the escape problem. And the solution is the flow except those artifical edges. Because every argumented path will be a unit flow since all the edges share the same constraint of 1. And the capacity constraint on splitted vertices will effect the the edge coming to the vertex. And every flow start from starting point will end up in the border with no vertice on two paths based on the vertex constraint flow network. And that is the solution. □

Remark: Please include your .pdf, .tex files for uploading with standard file names.