Set 2 Algoritmi

Cracco Andrea VR432685
Alberto Purrone VR433409
Litterini Simone VR439416

13 dicembre 2018

1 Exercise 1

Provide a linear time algorithm that takes in input a weighted directed graph $G = (V, E)$, with costs on theedges $c(e)$ for $e \in E$ (cost may be negative), a node $t \in V$ and, for each $v \in V$ a value $d(v)$,and decides whether for each $v \in V$ it is true that $d(v)$ is the cost of the path of minimum cost among all the paths from from v to t.

1.1 First Observation

We change the cost of each edge $e(v, w)$ according to the formula

$$c_{new}(e) = c(e) - d(v) + d(w)$$

- if $c_{new}(e) < 0$ the minimum cost didn't take in account the edge because $d(w) - d(v)$ is greater than the cost of the current edge so the cost are not correct;
- if $c_{new}(e) = 0$ the minimum path from v to t passes through this edge;
- if $c_{new}(e) > 0$ the edge is not part of the minimum path from v to t;

1.2 Second Observation

If there exist a path from v to t of cost c and we reverse all the edges, there should be a path from t to v with cost c;

1.3 Algorithm

1. Suppose that if t is not reachable from a vertex v, $d(v) = +\infty$;
2. For each edge $e(v, w)$, change its cost to $c_{new}(e)$;
3. Reverse the direction of all edges;

4. Starting from t, do a modified DFS in the following way:
5. For each edge $e(v, w)$ that is outgoing from the current vertex v:
   - If $c_{new}(e) = 0$: visit the node w
   - If $c_{new}(e) < 0$: return false
   - If $c_{new}(e) > 0$: ignore the edge
6. If there exist a node v not visited by the algorithm such that $d(v) < +\infty$: return false;
7. Return true;

2 Exercise 2

You are given a weighted directed graph $G = (V, E)$, with costs on the edges $c(e)$ for $e \in E$ (cost may be negative), a node $t \in V$ and, for each $v \in V$ a value $d(v)$ which is the cost of a path of minimum cost from v to t. Provide an $O(|E|log|V|)$ algorithm that given G, $d()$ and a node $t \neq t'$ computes, for each $v \in V$ the minimum cost of a path from v to t'.

Here is the algorithm:

1. Reverse the direction of each edge $e(v, w)$
2. For each edge, compute a new cost $\chi(e) = c(e) - d(v) + d(w)$
3. Do a Dijkstra starting from the node t and using the costs $\chi(e)$
4. Let $\gamma(v)$ be the minimum path costs computed at the previous step
5. The min cost of the path from a node v to t is then given by

$$d_{t'}(v) = d(v) + \gamma(v) - d(t')$$

6. Note that if t' is not reachable from v in the original graph, then $\gamma(v) = +\infty$ so $d_{t'}(v) = +\infty$

2.1

3 Exercise 3

Show that if we apply the algorithm seen in class to find the min-cost circulation for the network shown in the figure, there exist a sequence of choices of the cycle along which we improve that require $2 * 10^6$ iteration to solve the problem.

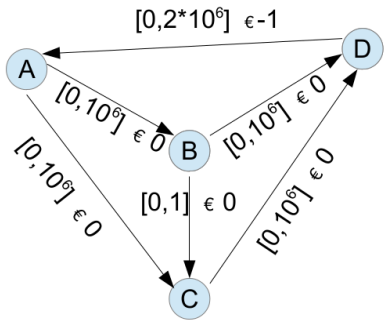


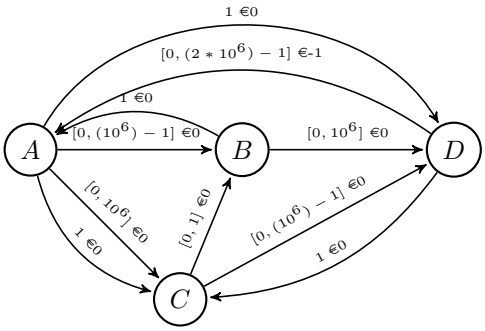
Figura 1: Flow network(N) of the problem

3.1 Algorithm

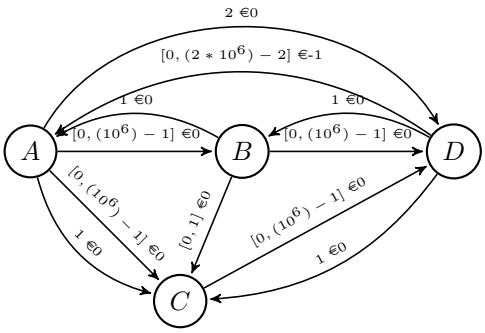
The steps of the algorithm saw in the class are:

1. Find a Feasible circulation Fc for N
2. While $R(Fc)$ has no negative cycle C, then $Fc = Sw(Fc, C)$

The fist step of the algorithm saw in the class is to produce the residual graph



the graph above is the result of the first iteration of the worst choice that the algoithm can do.



the second graph is the second iteration and if we continue to take the path from A to D passing each time from the edge between B and C (the path ABCD and ACBD) the result is a repeat of this two iterations above and using this choice we have that the algorithm takes $2 * 10^6$ iterations.

4 Exercise 4

Assume that the following matrix has to be tested for the C1P. Using the algorithm seen in class provide

a The overlap graph

b The containment graph

If the matrix has the C1P, provide 3 distict permutations of the columns that leave the 1s consecutive in each row (if there are fewer than 3 possible such permutations, provide them all). If the matrix does not have the C1P then apply the gap minimization algorithm considering only the last 5 columns.

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

4.1 overlap graph

for the overlap graph we have to compute Si that is the set of the columns that the value is 1; For the matrix A:

- r1 -> S1={1,2,3,5,6,8,9}
- r2 -> S2={1,2,3,4,5,6,9}
- r3 -> S3={2,3,5}

- r4 -> S4={2,6,9}
- r5 -> S5={1,2,3,5,6,7,8}
- r6 -> S6={2}
- r7 -> S7={5}

then:

- r1 overlap with r2
- r1 overlap with r5
- r2 overlap with r5
- r3 overlap with r4
- r4 overlap with r5

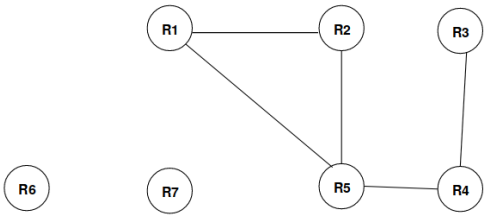


Figura 2: overlap graph

4.2 containment graph

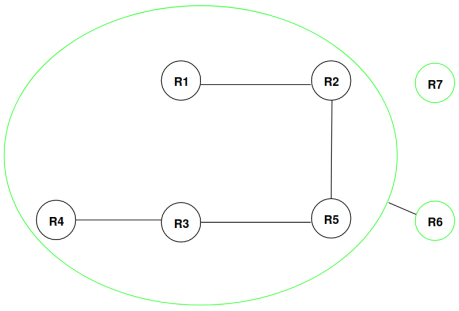


Figura 3: containment graph

4.3 Permutation of A

The only 2 permutations of the matrix A(c1,c2,c3,c4,c5,c6,c7,c8) in C1P are the following:

- c4 c9 c6 c2 c3 c5 c1 c8 c7

$$perm_1(A) = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

- c4 c9 c6 c2 c5 c3 c1 c8 c7

$$perm_2(A) = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

5 Exercise 5

A lab is seriously facing shortage of primary storage space for its files/data. The lab has to buying/rent secondary remote storage facilities. Preliminary analyses have indicated this as a better solution than in-house expansion of the primary storage. The different available options have both limitations in the maximum amount of space offered and costs related to the access of the information, once stored there. We are interested in determining the optimal policy to choose how to distribute the files not anymore fitting in-house on different secondary remote storage facilities in order to limit the cost incurred by taking into account the different expected usage rate of the different type of information we are relocating. Assume there are n remote facilities where to relocate the exceeding files. Let α_j be the maximum amount of information on the remote facility j and χ_j be the cost to access one unit of information from this facility. We assume that the information we need to store remotely is divided into m different categories, each one of which is accessed with some rate. Let γ_i be the amount of information units from the category i, and let ρ_i be the rate (how many times per unit time) a unit of information from category i will need to be retrieved. We aim at storing/distributing the information in the different remote storage places in order to minimize the overall expected cost of retrieval.

a) Build a circulation network in the following way:

- Add a node C_i for each category i
- Add a node S_j for each storage facility j
- Add an edge from each C_i to each S_j with cost $\rho_i * \chi_j$, capacity $+\infty$ and demand 0
- Add two nodes B, E that act as source and sink
- For each C_i add an edge $e(B, C_i)$ with cost 0, capacity and demand equal to γ_i.
- For each S_j add an edge $e(C_i, E)$ with cost 0, capacity α_j and demand 0.
- add an edge $x(E, B)$ with capacity $+\infty$, demand and cost 0.

b) If we assign the category with maximum rate to the minimum cost storage

- Suppose that we take two categories a, b and two facilities x, y such that $\rho_a >= \rho_b$ and $\chi_x <= \chi_y$.
- Suppose also that a and x are the choices of the greedy algorithm at step i and that b and y are respectively another category and another facility that a different algorithm could take.
- If we prove that

  $$\rho_a * \chi_x + \rho_b * \chi_y <= \rho_a * \chi_y + \rho_b * \chi_x$$

  then a and x are the globally best choices to match at step i.
- Rearranging terms:

  $$\rho_a * \chi_x - \rho_a * \chi_y <= \rho_b * \chi_y - \rho_b * \chi_x$$

- Recollecting ρ_a and ρ_b:

  $$\rho_a * (\chi_x - \chi_y) <= \rho_b * (\chi_x - \chi_y)$$

- If $\chi_x = \chi_y$ the expression is true
- We know that $z = (\chi_x - \chi_y) < 0$ by assumption
- If $\chi_x < \chi_y$ we have that $\rho_a * z <= \rho_b * z$ so it's true only if $\rho_a >= \rho_b$ but this condition is also an initial assumption, so the expression is also true.