Algorithm Design - Homework 2015/2016

Travagliati, Lorenzo
mat. 1495035

Simic, Djordje
mat. 1685456

December 2016

1 Theory Problem

Your grandfather loves riddles they publish in the Sunday newspaper. The last week riddle was as follows. You are given a grid $n \times n$ with n rows and n columns. For every row i you are given number r_i and for every column j you are given number c_j. To solve the riddle you need to mark some points on the grid, in the following way:

1. the number of marked points in every row is at most r_i,
2. the number of marked points in every column is at most c_j,
3. you mark the maximum number of points that satisfy 1. and 2.

The task is actually really difficult, so no wonder your grandfather has problems solving it. Your grandfather solved every riddle that the newspaper published in the last 44 years, so your heart is full of pain seeing him struggling with this week's one, and doubting his riddle-solving skills. You need to solve the riddle, but the solution you come up with has to be simple - you obviously cannot tell your grandfather that he needs to run a max-flow algorithm.

1.1 Task

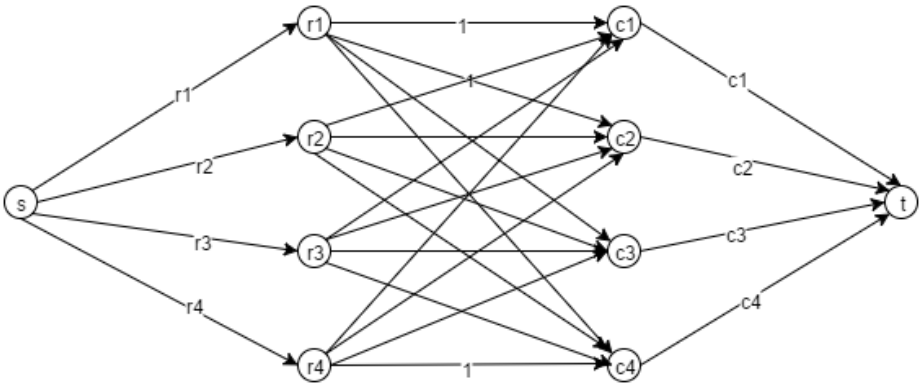
Give an algorithm that will take time linear or near-linear in n. Near linear means that you can have additional logarithmic factors in the complexity, so $O(n \cdot lg^7(n))$ is still fine, while $O(n^2)$ is unacceptable. State an algorithm in the pseudocode, prove its correctness and near-linear time complexity. No implementation is required.

2 Subset of eligible Min Cuts

The problem we are facing can be lead back to a *Maximum Flow Problem*: we can build a bipartite graph G composed by $(2 \times n + 2)$ nodes, where the $2 \times n$ nodes represents rows $r_1 \geq r_2 \geq \cdots \geq r_n$ of the given matrix and columns

$c_1 \geq c_2 \geq \cdots \geq c_n$ of the given matrix, both sequences sorted in a descending order. The remaining two nodes are the *source* and the *sink*.

Each node representing the rows is reached by a directed edge of capacity r_i from the *source*. Similarly directed edges leave the nodes representing the columns with capacity c_i and connect to the *sink* as shown below. Finally, each node r_i is connected to each node c_i by a unitary weight edge.



By runnning a *max flow algorithm* on this graph, we will find the optimal solution to the problem, so we will use this as a starting point in order to prove the correctness of our solution.

Let's say then that an upper bound to our problem can be easily found by considering the minimum between the sum of the row vector and the sum of the column vector, in particular

$$OPT \leq min(\sum_{i=1}^{n} r_i, \sum_{i=1}^{n} c_i)$$

Since we can't use any of the *max flow algorithms* because their time complexity exceeds the given constraint, we need to concentrate more on its dual problem, known as the *Minimum-Cut Problem*.

2.1 Minimum Cut

The nature of the problem (bipartite graph) let us define the *minimum cut* in the following way;

$$MinCut = min(\sum_{r_i \in \bar{N_R}} r_i + |N_R| \cdot |\bar{N_C}| + \sum_{c_i \in S_C} c_i)$$

where $\bar{N_R}$ represents the set of nodes r_i that doesn't belong to the cut, $\bar{N_R}$ represents the set of nodes r_i that belong to the cut and similiarly , $\bar{N_C}$ represents the set of nodes c_i that doesn't belong to the cut and N_C is the set of nodes that belong to the cut. What we want to do is to find a subset of all the cut that

are eligible to be *min cuts* that we know it to be the optimal solution because of the *Max Flow - Min Cut Theorem*.

In the very beginning we start to discriminate which row nodes are the best to select in order to achieve the *min cut* in few step.

Lemma 2.1. *Consider a cut formed by the row nodes, and let's say that for a certain k we have the following cut:*

$$n \cdot k + \sum_{i=k+1}^{n} r_i$$

where $n \cdot k$ represents the capacity of the unitary weighted edges at a given iteration that leave the cut and the remaining part represents the capacity of the edges from the source to the r_i nodes not included in the cut. Among all the possible row nodes, the nodes that will be part of the min cut are being choosen in a non descending fashion and in a contiguos way.

Proof. Let's have another cut, formed by

$$n \cdot (k+1) + \sum_{i=k+2}^{n} r_i.$$

We have to prove that:

$$n \cdot k + \sum_{i=k+1}^{n} r_i \leq n \cdot (k+1) + \sum_{i=k+2}^{n} r_i.$$

And that is very easy since we can rewrite the formula in the following way:

$$n \cdot k + r_i + \sum_{i=k+2}^{n} r_i \leq n \cdot k + n + \sum_{i=k+2}^{n} r_i$$

and by subtracting $n \cdot k$ and the sum of the r_i we conclude that $r_i \leq n$ which is always true since r_i can be at most n. □

Now that we know which row nodes are we going to select, me must discuss which nodes belonging to the colum nodes we are going to pick in order to build our *min cut*.

Lemma 2.2. *Consider a cut, at a given iteration k formed by the most k significant r_i nodes (according to **Lemma 2.1**) and the c_{j+1} nodes such that $c_{j+1} < k$ so that we have*

$$\sum_{i=k+1}^{n} r_i + k \cdot j + \sum_{i=j+1}^{n} c_i$$

. Then that cut is eligible to be a min cut.

Proof. Let's have a cut

$$P_1 = \{s, R_1, R_2, \cdots, R_n, C_{j+1}, C_{j+2}, \cdots, C_n\}$$

where $R_1, R_2, \cdots, R_n$ belongs to the row nodes and $C_{j+1}, C_{j+2}, \cdots, C_n$ belong to the column nodes. Similarly let's have

$$P_2 = \{s, R_1, R_2, \cdots, R_n, C_u, C_{u+1}, \cdots, C_n\}$$

and

$$P_3 = \{s, R_1, R_2, \cdots, R_n, C_v, C_{v+1}, \cdots, C_n\}$$

such that $j \in [0, n-1]$ is the first index so that $c_{j+1} < k, 0 \leq u < (j+1)$ and $(j+1) < v \leq n$.

What we are going to prove is that $cap(P_1) \leq cap(P_2)$ and $cap(P_1) \leq cap(P_3)$. Consider P_2 and P_3 with $u = j$ and $v = j+2$, we can do that because the columns are ordered in a descending fashion. The capacity of P_1 is:

$$cap(P_1) = \sum_{i=k+1}^{n} r_i + k \cdot j + \sum_{i=j+1}^{n} c_i$$

and in the same way

$$cap(P_2) = \sum_{i=k+1}^{n} r_i + k \cdot (j-1) + \sum_{i=j}^{n} c_i$$

and that's equivalent as

$$\sum_{i=k+1}^{n} r_i + k \cdot (j-1) + \sum_{i=j+1}^{n} c_i + c_j$$

By construction we know that j is the first index such that $c_{j+1} < k$. Without loss of generality, we can rewrite cj in this way: $c_j = k + \epsilon$ with $\epsilon \geq 0$.

Therefore, if we put c_j in the $cap(P_2)$ formula as shown below

$$cap(P_2) = \sum_{i=k+1}^{n} r_i + k \cdot (j-1) + \sum_{i=j+1}^{n} c_i + k + \epsilon$$

so we can rewrite as follows:

$$cap(P_2) = \sum_{i=k+1}^{n} r_i + k \cdot j + \sum_{i=j+1}^{n} c_i + \epsilon$$

which is obviously greater or equal to the capacity of P_1, such that

$$\sum_{i=k+1}^{n} r_i + k \cdot j + \sum_{i=j+1}^{n} c_i \leq \sum_{i=k+1}^{n} r_i + k \cdot j + \sum_{i=j+1}^{n} c_i + \epsilon$$

and $0 \leq \epsilon$ which is true.

The proof for $cap(P_1) \leq cap(P_3)$ is similiar to what we have previously done. In fact, we define P_3 to be

$$\sum_{i=k+1}^{n} r_i + k \cdot (j+1) + \sum_{i=j+2}^{n} c_i$$

and we know that $c_{j+1} < k$ by how we defined the sets. So, because the column nodes are descending, we can state that $c_{j+1} = k - \epsilon$ with $\epsilon \geq 0$ and in the same way we did before, we can rewrite the P_1 with this new notion:

$$\sum_{i=k+1}^{n} r_i + k \cdot j + \sum_{i=j+2}^{n} c_i + c_{j+1}$$

$$\sum_{i=k+1}^{n} r_i + k \cdot j + \sum_{i=j+2}^{n} c_i + k - \epsilon$$

$$\sum_{i=k+1}^{n} r_i + k \cdot (j+1) + \sum_{i=j+2}^{n} c_i - \epsilon$$

So now, by comparing the two capacities, we have that

$$\sum_{i=k+1}^{n} r_i + k \cdot (j+1) + \sum_{i=j+2}^{n} c_i - \epsilon \leq \sum_{i=k+1}^{n} r_i + k \cdot (j+1) + \sum_{i=j+2}^{n} c_i$$

and therefore $\epsilon \leq 0$. □

3 The Algorithm

What we achieved with the previous two demonstrations, is that we now have defined a way to determine a subset between all the possible cuts eligible to be *min cuts*. What we are going to do now, is just iterate over the k value that goes from 0 to n, and in each step we check if the value of the capacity of the cut we are studying is actually lesser than the

$$min(\sum_{i=1}^{n} r_i, \sum_{i=1}^{n} c_i)$$

that we previously said it was an *upper bound* to our problem. So what the algorithm does is basically

$$min_{k=0,1,\cdots,n} = (\sum_{i=k+1}^{n} r_i + k \cdot j + \sum_{i=j+1}^{n} c_i)$$

and that's the equivalent of

$$min_{k=0,1,\cdots,n} = (\sum_{i=k+1}^{n} r_i + min(c_1, k) + \cdots + min(c_n, k))$$

. The following pseudo-code is a near-linear implementation of what we discussed earlier.

Algorithm 1 Min-cut selection

```
1: procedure MinCut
2:     n ← loadMatrixSize()
3:     columns ← sort(loadColumns())
4:     rows ← sort(loadRows())
5:     lastIndex ← n
6:     columnHashMap // for storing number of occurrences of column values
7:
8:     for each column c do
9:         if c ∈ columnHashMap then
10:            columnHashMap.increment(c)
11:        else
12:            columnHashMap.add(c, 1)
13:        end if
14:    end for
15:
16:    rowSum ← 0
17:
18:    for row in rows do
19:        rowSum += row
20:    end for
21:
22:    minSum ← rowSum
23:    midSolution ← 0
24:    partialColumnSum ← 0
25:
26:    for k = 1; k < n; k++ do
27:        rowSum = rowSum - rows[k - 1];
28:        for i = lastIndex; i >= 0; i−− do
29:            if columns[i] >= k then
30:                midSolution+ = k · (i + 1)
31:                break;
32:            else
33:                lastIndex = lastIndex - columnHashMap.get(columns[i])
34:                partialColumnSum=columns[i]·columnHashMap.get(columns[i])
35:                i = lastIndex + 1
36:            end if
37:        end for
38:
39:        currentSolution = midSolution + partialColumnSum + rowSum
40:        midSolution ← 0
41:        if minSum >= currentSolution then
42:            minSum = currentSolution
43:        end if
44:    end for
45:
46: return minSum
47:
48: end procedure
```