CS261 Winter 2018 - 2019
Lecture 4: Matrix Rounding*

Instructor: Ashish Goel
Scribe: Kaidi Yan
Edited: Geoff Ramseyer

January 18, 2019

1 Problem Definition

Given a matrix with noninteger entries, we would like to find another matrix with the same row and column sums but with integer entries. We will show that such a matrix always exists and how to find it in polynomial time, via a reduction to a max-flow problem.

More formally:

We are given an $n \times n$ matrix A where each entry A_{ij} is non-negative. Furthermore, the sum of each row or column of A is an integer, i.e., $\forall i = 1, ..., n$:

$$\sum_{j=1}^{n} A_{ij} \in \mathbb{Z}$$
$$\sum_{j=1}^{n} A_{ji} \in \mathbb{Z}$$

Our goal is find a *rounding* of A, which is defined as a new $n \times n$ matrix B such that B_{ij} is either $\lfloor A_{ij} \rfloor$ or $\lceil A_{ij} \rceil$, and the sum of each row and column of B is the same as before, i.e., $\forall i = 1, ..., n$:

$$\sum_{j=1}^{n} B_{ij} = \sum_{j=1}^{n} A_{ij}$$
$$\sum_{j=1}^{n} B_{ji} = \sum_{j=1}^{n} A_{ji}$$

*For Dinic's Algorithm, please refer to Tim Roughgarden's Winter 2016 CS261 lecture #2 notes

There are two questions we can ask. First, does such a rounding B always exist? Second, how fast can we find such a rounding B for a given matrix A if a rounding does exist? For the remaining part of this lecture, we are going to show two proofs, the first one dedicated to show that a rounding B always exists, and the second one dedicated to show that solving matrix rounding is no harder than solving a max-flow problem.

Without loss of generality, to simplify our proofs, note that we can assume that each matrix entry is between 0 and 1. Observe that for any matrix A, we can define $\tilde{A}$ by letting $\tilde{A}_{ij} = A_{ij} - \lfloor A_{ij} \rfloor$. If we find a rounding $\tilde{B}$ of $\tilde{A}$, we can construct B by setting $B_{ij} = \tilde{B}_{ij} + \lfloor A_{ij} \rfloor$. It's easy to verify that B is a rounding of A.

2 The First Proof

For the first proof, we try to show that a rounding B always exists for every matrix A. For demonstration, we will start with a concrete example of A — and along the way hopefully show how the proof can be generalized. Let A be:

$$\begin{bmatrix} 0.9 & 0.6 & 0.5 \\ 0 & 0.2 & 0.8 \\ 0.1 & 0.2 & 0.7 \end{bmatrix}$$

We construct a bipartite graph with nodes $R_1, R_2, R_3, C_1, C_2, C_3$ and edges between R_i and C_j for each $i = 1, 2, 3$ and $j = 1, 2, 3$ (edge directions are not important here). We can think of R_i representing the ith row of A and similarly C_i representing the ith column of A. In addition, each edge (R_i, C_j) is labeled with A_{ij}. For example, the edge (R_1, C_1) is labeled with 0.9 based on our example of A. Finally we remove edges that have label 0. Figure 1 shows how the graph looks like.

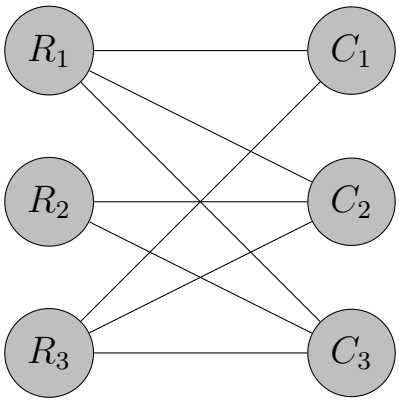

Figure 1: Bipartite graph constructed from our example A

From this bipartite graph, we pick an edge with a fractional label. If there's no such edge, then A is already a rounding of itself since every entry is an integer. In our example,

suppose we pick edge (R_1, C_1). Then, we can pick a different edge starting from C_1 with fractional label; such an edge is guaranteed to exist since $A_{11} + A_{21} + A_{31}$ is an integer and A_{11} is fractional. Let's say we pick (C_1, R_3). Then we can pick another edge starting from R_3 with fractional label since $A_{31} + A_{32} + A_{33}$ is an integer and A_{31} is fractional.

In general, we can keep on this process of picking edges — since there are only a finite number of edges in the bipartite graph, we are eventually going to encounter a cycle before we run out of edges to pick. Here, suppose we pick (R_1, C_1), (C_1, R_3), (R_3, C_2) and (C_2, R_1). This forms a cycle $R_1 - C_1 - R_3 - C_2 - R_1$. [1]

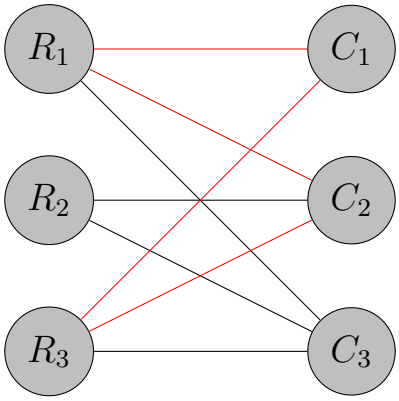

Figure 2: Edges picked from the bipartite graph, which are marked with red

Next, we alternate adding and subtracting a fixed amount δ from the label of every edge in this cycle. In particular, we add δ to the labels of (R_1, C_1) and (R_3, C_2) as well as subtract δ from the labels of (C_1, R_3) and (C_2, R_1)[2]. These alternating additions / subtractions make sure that after we change the labels of all edges in the cycle and set A_{ij} to be the new label of (R_i, C_j), the sum of each row and column of A remains the same as before. We let δ be as small as possible and also guarantee that after the additions and subtractions, at least one edge in the cycle has label equal to either 0 or 1. Based on this, the δ we choose for cycle $R_1 - C_1 - R_3 - C_2 - R_1$ is 0.1. The new labels of (R_1, C_1), (C_1, R_3), (R_3, C_2) and (C_2, R_1) become 1, 0, 0.3 and 0.5 respectively.

Now with the new labels for each edge in the cycle, we have a new graph that has at least one fewer edge with a fractional label than in the original graph. This means that if we convert the new graph back into a new matrix A' (where A_{ij} is set to be the label of edge (R_i, C_j)), then A' has strictly fewer fractional entries than A and each row sum and column sum of A' is the same as A. If we perform this process until there are no more fractional labels, we will eventually make all edge labels integral. The final matrix based on the last graph is therefore a rounding of matrix A.

[1] Note that the cycle need not include the starting node, e.g. it is conceivable that the path we took would be $R_1 - C_1 - R_3 - C_3 - R_2 - C_2 - R_2$, in which case the cycle we found would have been $R_2 - C_2 - R_3 - C_3$.

[2] we can also choose to subtract δ from the labels of (R_1, C_1) and (R_3, C_2) as well as add δ to the labels of (C_1, R_3) and (C_2, R_1) — the proof remains almost the same.

It's easy to see that the arguments above apply to any choices of matrix A. Hence we prove that every matrix A has a rounding.

3 The Second Proof

For the second proof, we show that the problem of matrix rounding can be reduced to a max-flow problem. This answers our question of how quickly a matrix rounding can be found.

Similar to the first proof, we first set up a graph with nodes R_i representing each row and nodes C_j representing each column ($i = 1, ..., n, j = 1, ..., n$), and edges between each R_i and C_j. In this case, the edge direction matters, and edges will flow from the R_i to C_j). In addition, we add two more nodes, s and t, into the graph. For each R_i, we add an edge from s to R_i; for each C_j, we add an edge from C_j to t.

Now we convert this graph into a flow graph. To do so we specify an edge capacity u_e to every edge e. The rules are as follow:

1. $\forall i = 1, ..., n, j = 1, ..., n, u_{(R_i,C_j)} = 1$
2. $\forall i = 1, ..., n, u_{(s,R_i)} = \sum_{j=1}^{n} A_{ij}$
3. $\forall i = 1, ..., n, u_{(C_i,t)} = \sum_{j=1}^{n} A_{ji}$

Note that rule 2 and 3 essentially mean that the capacity of (s, R_i) is the ith row sum of A and the capacity of (C_i, t) is the ith column sum of A. Based on problem definition, the edge capacities are all integers.

For this flow graph, the value of a max-flow from s to t is at most $\sum_{i=1}^{n} u_{(s,R_i)} = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij}$, as that's the maximum amount of flow going out of s or into t. Furthermore, we know that this value can be achieved — we just need to set the flow of (R_i, C_j) to be A_{ij} and saturate the edges from s to R_i and C_j to t. We can easily verfiy such a flow satisfies both capacity and conservation constraints and has value equal to $\sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij}$. Hence $\sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij}$ is indeed the value of a max-flow from s to t for this flow graph.

But remember the max-flow is not unique — in fact, let the flow we get by running Ford-Fulkerson (FF) or Edmonds-Karp (EK) be f. This flow f has the following properties:

1. $\forall e \in E, f_e \in \mathbb{Z}$.
2. $\forall i = 1, ..., n, f_{(s,R_i)} = u_{(s,R_i)}$ and $f_{(C_i,t)} = u_{(C_i,t)}$. In other words all edges going out of s or into t must be saturated.

Property 1 holds because for each augmenting step of FF or EK, the augmenting path we find must have integer capacity for each edge included, thus the increment in the amount of flow for each edge must also be integral. Therefore f is an integer flow.

Property 2 holds because we already know that the max-flow value is $\sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij}$, which means that the amount of flow going out of s or going into t must be equal to

$\sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij}$. This can only be achieved when all edges going out of s or into t are saturated.

Now define matrix B by setting $B_{ij} = f_{(R_i,C_j)}$. Since f is an integer flow, each entry of B must also be an integer. Furthermore, combining property 2 above and conservation constraints we have:

$$\forall i = 1, ..., n, \sum_{j=1}^{n} f_{(R_i,C_j)} = f_{(s,R_i)} = u_{(s,R_i)} = \sum_{j=1}^{n} A_{ij}$$
$$\forall i = 1, ..., n, \sum_{j=1}^{n} f_{(R_j,C_i)} = f_{(C_i,t)} = u_{(C_i,t)} = \sum_{j=1}^{n} A_{ji}$$

Hence replacing $f_{(R_i,C_j)}$ with B_{ij}, we see that each row / column sum of B is the same as the corresponding row / column sum of A. Therefore B is a rounding of A.

This proof provides a practical way to find a rounding of A:

1. Construct a flow graph of A;
2. Run some max-flow algorithm with polynomial runtime (EK or Dinic) on the constructed flow graph to get a max-flow f;
3. Convert f into a new matrix B, which is a rounding of A.

All these steps take polynomial time (in terms of matrix size n). Hence, we can solve the problem of matrix rounding in polynomial time in terms of matrix size.