Binary Matrix - 2

Solution sketch:

Brute force over all expected number of ones in rows and columns, so cost of this brute force will be r x c. But this can be optimized. We'll only brute force over number of expected ones in rows. For example, if it is x, then, number of 1s in columns must be (x * r) / c. If it's a fraction, x is not a valid option. So this loop is O(r).

Step 1.
Now the following solution is for converting the matrix to a matrix such that each column will have x 1s and each rows will have y 1s.
For each row we know how many more 1s required in this row and similarly for each column we know how many 1s required. We can put these into two arrays, req_row[], req_col[], negative values indicate number of 1s should be decreased for that row or column.

If the cell value of Matrix(i, j) is 0, and both of req_row[i] and req_col[j] is positive, then we can convert it to 1. Increase result by 1, and update req_row and req_col accordingly and if the cell value of Matrix(i, j) is 1, and both of the req_row[i] and req_col[j] is negative, then we can convert it to 0. Increase result by 1, and update req_row and req_col accordingly. By this way, we can by only one flip we can move, one row req closer to zero and one column req closer to zero. We should try to do as much as this kind of operation to minimize the number of flips. This part can be solved by maximum flow. Maximum flow will calculate number of such flip, where both of one row_req and one col_req will move closer to zero by 1. After this max flow, we'll flip the cells, we found from flow, and update row_req and col_req.

Step 2.
Our target was to make all elements of req_row and req_col zero, and by these above two steps, we ensure, that if it is possible to make one element of req_row and one element of req_col can be moved closer to zero by one flip only, we do that.

Now if we reach such a state that, where all of the elements of req_col is zero, the sum of req_row elements will be zero. The solution of this state is summation of all absolute values of req_row. (Because, if for any i, req_row[i] is positive, there will be a j for which req_row[j] is negative. In that case, we will find a column k for which, row[i][k] is 0 and row[j][k] is 1. So we will flip both of row[i][k] and row[j][k]. So req_row[i] will be decrease by one and req_row[j] will be increased by one. By repeating this we'll eventually reach all zero in req_row, and this is our target.). Similarly if we reach any state, where all elements of req_row is zero, then result is summation of all elements of req_col.

But if both of req_row and req_col has non zero values for some element. we'll convert one of the array to zero first. Then we will follow the previous procedure to get the result. For example, if req_col[i] is -1, if we want to convert it to zero, we actually have to flip one 1 to 0, in column i, so one element of req_row will be increased by 1. You can increase any non negative value of req_row for this, because, ultimate result will depend only on sum of absolute values. (but you cannot increase any negative value, because, this case will not occur after step 1 where we can remove 1 zero and both of the row and column will move closer to zero.) This way, we can convert all req_col to zero, and get the result by above procedure. We'll convert all req_col to zero first or all req_row to zero first depending on which costs us less.

So overall complexity is O(n ^ 4).