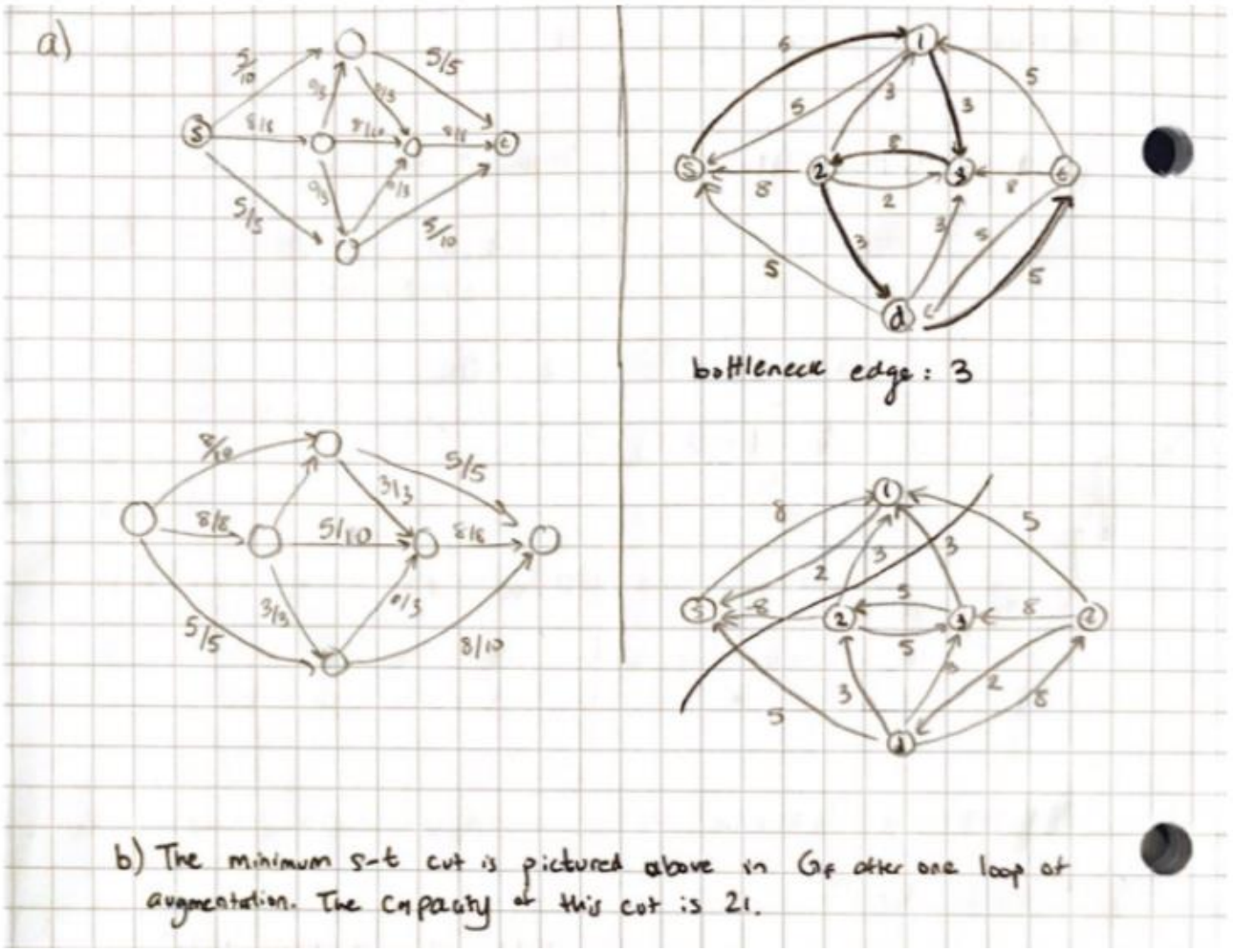
1. The value of this flow is 18, because there is 18 flow leaving the source node and 18 entering the sink node. This is not the maximum flow of the graph because we can push 3 more flow through the top left edge of the graph and make the maximum flow 21.

b. The minimum cut it displayed as follows:

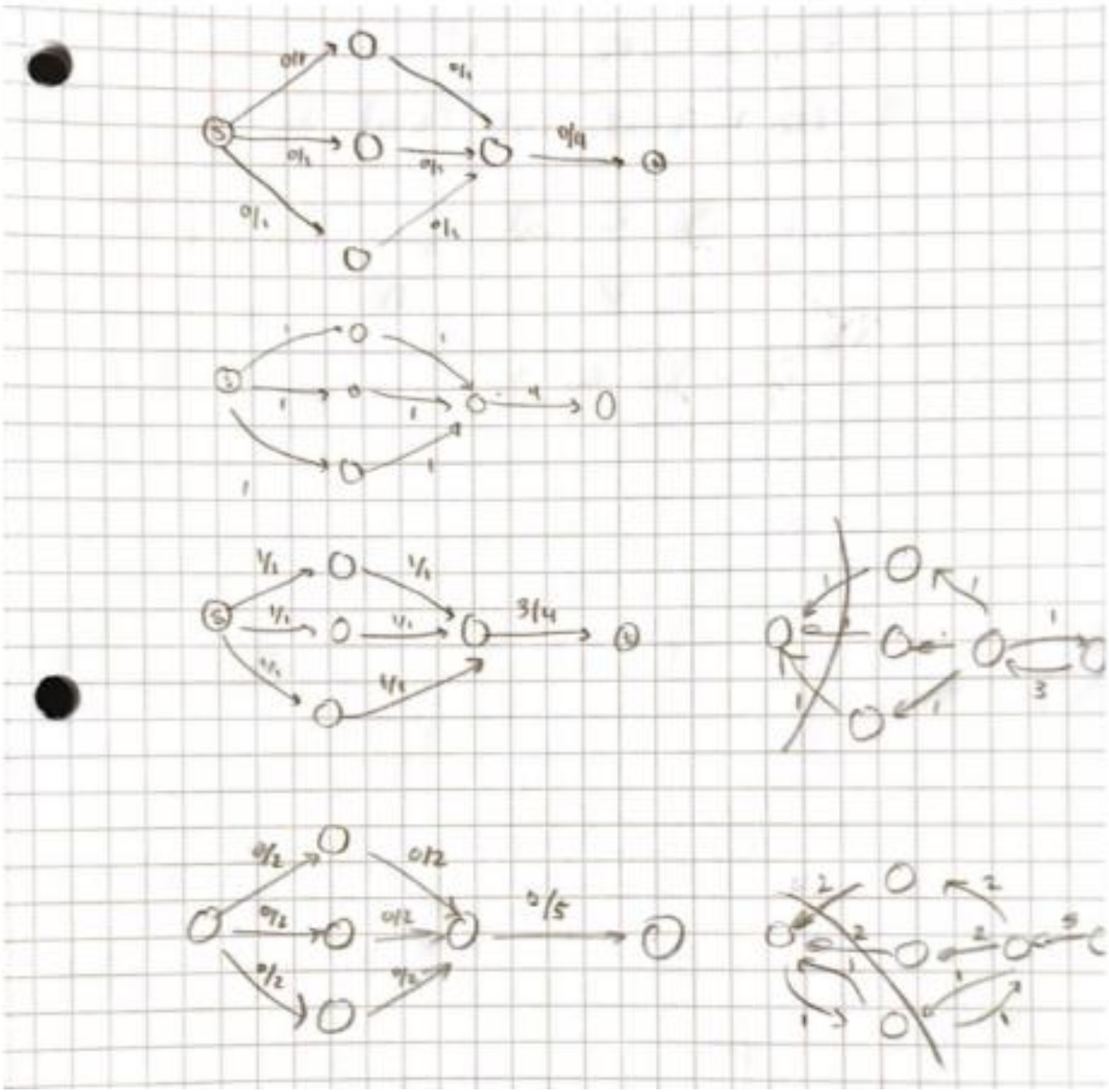
The capacity of this cut is 21



E

2. This is false, and a counter example of this can be seen in problem 1a, where not every edge going out of the start node is saturated.

b. As you can see, when the capacities of all the edges are increased in capacity by 1, in this particular graph, the Ford Fulkerson algorithm produces a different minimum cut, contradicting the statement.



3.

a. With one more unit edge added to the Graph G, we need only to execute the Fold-Fulkerson algorithm one more time. The new edge with capacity one adds one edge to the residual graph, so we must look for a path and augment the flow on the path if it is found. The time for this extra iteration would be as much as a depth first search, or O(V+E) = O(E) because the edges dominate the amount of vertices. We only need to do one iteration of the augmenting algorithm because an edge with a capacity of one increases the capacity of the minimum cut by at most 1. The max flow, which is equal to the minimum capacity is also increased by at most 1. Since all edge capacities are 1, any augmentation to the residual graph increases the flow by only one, so only 1 augmentatin is needed.

b. There are two cases with the edge e = (u,v). If e has no flow, we don't need to do anything because this will not change the residual graph and will not effect the maximum flow. If *e* has a flow, it must have a flow value of 1 since it's capacity is 1. With edge e gone, there is now one more unit of flow going into *u* than is going out, and 1 more unit of flow going out of *v* than is coming in. We must either try and change the flow values of different edges such that the flow stays the same throughout the graph, or if this is not possible, we must decrease the flow from s to u by 1 unit and v to t by 1 unit. To try and change the flow, we look for an augmenting path from u to v (without edge e), and if it exists, we augment along that path to "reroute" the flow.

If the augmenting path does not exist, we reduce the flow from s to u by finding an augmenting the path from u to s, then augmenting the flow from v to t. The time that this algorithm would take is O(V+E) = O(E) becauase we use DFS or BFS to find the path whenever we need to.

`

4. We can solve this problem through an application of a flow network. We make nodes for each of the clients $\{c_1, c_2 ..., c_n\}$ and nodes for each of the base stations $\{b_1, b_2,..., b_m\}$. We then create nodes s and t, the source noes and sink nodes respectively. We then construct the network graph as follows:

Add edges with capacity 1 from the source node to each of the client nodes. Then, for each pair of client nodes and base nodes which are within the range r, add an edge from the client noe to the base node with capacity 1. Then we add an edge from each of the base station nodes to the sink node t with a capacity of L.

Now that the network has been constructed, we solve the problem of whether or not each client can be connected to a base station at the same time by computing the maximum flow in this network. We know that every client is connected to a base station in this network if the maximum flow, or the flow value going out of the source node is equal to the amount of client nodes n since each edge from s to c_n has a capacity of 1.

In a valid solution to this problem where each client is connected to one base station, we know that each base station is connected to at most L client nodes because that is the base station's capacity and each client can send a flow value of 1. Due to the conservation rule of flow networks, the total value of flow out of s is going to be the exact same amount of flow into t, and if each of the edges going out of s is saturated, then we have a maximum flow of n.

This algorithm runs in polynomial time since adding two notes, then connecting s with n nodes and k with t nodes can be done in polynomial time. Then the maximum-flow problem can also be computed in polynomial time.