CSE 6140 / CX 4140

Computational Science & Engineering Algorithms

Homework 5

Please type all answers.

1. (10 points)

The figure below shows a flow network on which an s-t flow has been computed. The capacity of each edge appears as a label next to the edge, and the numbers in boxes give the amount of flow sent on each edge. (Edges without boxed numbers have no flow being sent on them.)

(a) What is the value of this flow? Is this a maximum (s,t) flow in this graph?

(b) Find a minimum s-t cut in the flow network, and also say what its capacity is.

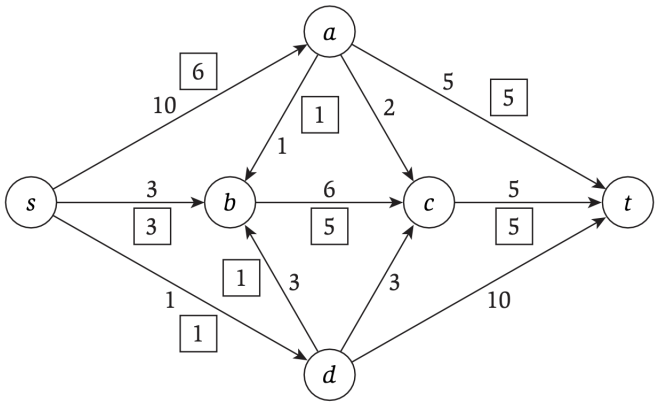

Figure 1

(a)

$$\text{Flow value} = 6(s \to a) + 3(s \to b) + 1(s \to d) = 10$$

This is not the max flow as there is one augmenting path from the source to the sink, which is $s \to a \to c \to b \to d \to t$, sending one unit on this path, making the max flow 11.

(b) A min cut in this flow network is $A = \{s, a, b, c\}$ and $B = \{d, t\}$. The capacity can be found with cut edges $s \to d, b \to d, c \to d, a \to t, b \to t$, which gives a value of $1 + 0 + 0 + 5 + 5 = 11$.

2. (10 points)

(a) Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

Let G be an arbitrary flow network, with a source s, a sink t, and a positive integer capacity c_e on every edge e. If f is a maximum $s-t$ flow in G, then f saturates every edge out of s with flow (i.e., for all edges e out of s, we have $f(e) = c_e$).

This statement is **false**. The maximum s-t flow does not necessarily saturate all edges leaving s. The flow depends on the network structure and constraints of the graph, such as bottleneck edges elsewhere in the network.

Consider a flow network with the following structure:

- $s \to a$ with capacity 10
- $s \to b$ with capacity 10
- $a \to t$ with capacity 5
- $b \to t$ with capacity 5

The total flow from s to t is at most 10, as the bottleneck is the capacities of the edges $a \to t$ and $b \to t$ (5 each). A valid maximum flow assigns:

- $f(s \to a) = 5$
- $f(s \to b) = 5$

Here, the edges $s \to a$ and $s \to b$ are not saturated, even though the flow is maximum. Therefore, the statement is false.

(b) Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

Let G be an arbitrary flow network, with a source s, a sink t, and a positive integer capacity c_e on every edge e; and let (A, B) be a mimimum $s-t$ cut with respect to these capacities $\{c_e : e \in E\}$. Now suppose we add 1 to every capacity; then (A, B) is still a minimum $s-t$ cut with respect to these new capacities $\{1 + c_e : e \in E\}$.

This statement is **true**. Adding 1 to every edge capacity increases the total capacity of any s-t cut by the same amount (equal to the number of edges crossing the cut). Since the relative capacities of all cuts remain unchanged, the minimum cut remains the same.

- The capacity of a cut (A, B) is the sum of the capacities of edges crossing from A to B

- If you increase every edge capacity by 1, the new capacity of the cut becomes:

$$\text{New capacity} = \text{Old capacity} + (\text{number of edges crossing the cut})$$

- For any other cut (A', B'), the new capacity also increases by the number of edges crossing that cut. Since the relative comparison of cut capacities does not change, the minimum cut remains the same.

3. (20 points) Consider a set of mobile computing clients in a certain town who each need to be connected to one of several possible base stations. We'll suppose there are n clients, with the position of each client specified by its (x, y) coordinates in the plane. There are also k base stations; the position of each of these is specified by (x, y) coordinates as well.

For each client, we wish to connect it to exactly one of the base stations. Our choice of connections is constrained in the following ways. There is a range parameter r – a client can only be connected to a base station that is within distance r. There is also a load parameter L – no more than L clients can be connected to any single base station.

Your goal is to design a polynomial-time algorithm for the following problem. Given the positions of a set of clients and a set of base stations, as well as the range and load parameters, decide whether every client can be connected simultaneously to a base station, subject to the range and load conditions in the previous paragraph.

```
def can_connect_all_clients(clients, base_stations, r, L):
    # construct the graph
    n = len(clients)
    k = len(base_stations)
    source = n + k
    sink = n + k + 1
    graph = FlowGraph(n + k + 2)

    # add edges from source to clients
    for i in range(n):
        graph.add_edge(source, i, 1)

    # add edges between clients and base stations within range
    for i, (x_c, y_c) in enumerate(clients):
        for j, (x_b, y_b) in enumerate(base_stations):
            if distance(x_c, y_c, x_b, y_b) <= r:
                graph.add_edge(i, n + j, 1)

    # add edges from base stations to sink
    for j in range(k):
        graph.add_edge(n + j, sink, L)

    # compute maximum flow
    max_flow = graph.max_flow(source, sink)

    # check if all clients are connected
    return max_flow == n
```

Computing distances between n clients and k base stations takes $O(nk)$. Adding edges to the graph also takes $O(nk)$. Maximum flow for a graph with $O(n + k)$ nodes and $O(nk)$ edges using Edmonds-Karp runs in $O(VE^2) = O((n + k)(nk)^2)$. Thus, the overall complexity is polynomial in n and k.

4. (20 points)

Network flow issues come up in dealing with natural disasters and other crises, since major unexpected events often require the movement and evacuation of large numbers of people in a short amount of time.

Consider the following scenario. Due to large-scale flooding in a region, paramedics have identified a set of n injured people distributed across the region who need to be rushed to hospitals. There are k hospitals in the region, and each of the n people needs to be brought to a hospital that is within a half-hour's driving time of their current location (so different people will have different options for hospitals, depending on where they are right now).

At the same time, one doesn't want to overload any one of the hospitals by sending too many patients its way. The paramedics are in touch by cell phone, and they want to collectively work out whether they can choose a hospital for each of the injured people in such a way that the load on the hospitals is balanced: Each hospital receives at most $\lceil n/k \rceil$ people.

Give a polynomial-time algorithm that takes the given information about the people's locations and determines whether this is possible.

```
def can_distribute_people(n, k, people, hospitals, driving_ranges):

    # hospital capacity
    max_capacity = math.ceil(n / k)

    # make flow graph
    graph = FlowGraph(n + k + 2)
    source = n + k
    sink = n + k + 1

    # add edges from source to people
    for i in range(n):
        graph.add_edge(source, i, 1)

    # add edges from people to hospitals they can reach
    for i, reachable_hospitals in enumerate(driving_ranges):
        for hospital in reachable_hospitals:
            graph.add_edge(i, n + hospital, 1)

    # add edges from hospitals to sink
    for j in range(k):
        graph.add_edge(n + j, sink, max_capacity)

    # calculate max flow
    max_flow = graph.max_flow(source, sink)

    return max_flow == n
```

Adding edges involves $O(n \times d)$, where d is the average number of hospitals within range for a person. Using Edmonds-Karp, maximum flow runs in $O(VE^2)$. The algorithm runs in polynomial time relative to n and k.

5. (20 points)

Consider a large-scale atmospheric science experiment. You need to get good measurements on a set S of n different conditions in the atmosphere (such as the ozone level at various places), and you have a set of m balloons that you plan to send up to make these measurements. Each balloon can make at most two measurements. Unfortunately, not all balloons are capable of measuring all conditions, so for each balloon $i = 1, ..., m$, they have a set S_i of conditions that balloon i can measure. Finally, to make the results more reliable, you plan to take each measurement from at least k different balloons. (Note that a single balloon should not measure the same condition twice.)

Example. Suppose that $k = 2$, there are $n = 4$ conditions labeled $c1, c2, c3, c4$, and there are m =4 balloons that can measure conditions, subject to the limitation that $S1 = S2 = \{c1, c2, c3\}$, and $S3 = S4 = \{c1, c3, c4\}$. Then one possible way to make sure that each condition is measured at least $k = 2$ times is to have

- balloon 1 measure conditions $c1, c2$,
- balloon 2 measure conditions $c2, c3$,
- balloon 3 measure conditions $c3, c4$, and
- balloon 4 measure conditions $c1, c4$.

(a) Give a polynomial-time algorithm that takes the input to an instance of this problem (the n conditions, the sets S_i for each of the m balloons, and the parameter k) and decides whether there is a way to measure each condition by k different balloons, while each balloon only measures at most two conditions.

```
def can_measure_conditions(n, m, k, condition_sets):
    # make flow graph
    graph = FlowGraph(n + m + 2)
    source = n + m
    sink = n + m + 1

    # add edges from source to conditions
    for i in range(n):
        graph.add_edge(source, i, k)

    # add edges from conditions to balloons
    for i, conditions in enumerate(condition_sets):
        for condition in conditions:
            graph.add_edge(condition, n + i, 1)

    # add edges from balloons to sink
    for j in range(m):
        graph.add_edge(n + j, sink, 2)
```

```
    # get max flow
    max_flow = graph.max_flow(source, sink)

    return max_flow == n  k
```

Graph construction takes $O(n+m+E)$. Using Edmonds-Karp, finding max flow is $O(VE^2)$. Thus, the overall complexity is polynomial in n, m, and E.

(b) Suppose each of the balloons is produced by one of three different sub-contractors involved in the experiment. A requirement of the experiment is that there be no condition for which all k measurements come from balloons produced by a single subcontractor.

For example, suppose balloon 1 comes from the first subcontractor, balloons 2 and 3 come from the second subcontractor, and balloon 4 comes from the third subcontractor. Then our previous solution no longer works, as both of the measurements for condition c_3 were done by balloons from the second subcontractor. However, we could use balloons 1 and 2 to each measure conditions c_1, c_2, and use balloons 3 and 4 to each measure conditions c_3, c_4.

Explain how to modify your polynomial-time algorithm for part (a) into a new algorithm that decides whether there exists a solution satisfying all the conditions from (a), plus the new requirement about subcontractors.

The additional requirements are that each balloon is produced by one of three subcontractors and for every condition c_i, the k measurements must come from at least two different subcontractors. We can enforce the subcontractor constraint by introducing virtual nodes for each subcontractor:

- Add three subcontractor nodes (SC_1, SC_2, SC_3) between the balloons and the sink
- Connect each balloon b_j to its corresponding subcontractor node $(SC_{subcontractor(j)})$ with capacity 2
- Add edges from the condition nodes to each subcontractor node (SC_1, SC_2, SC_3)
- Assign a capacity of k to these edges, ensuring k flows must pass through at least two subcontractor nodes

The updated algorithm would:

- Compute the maximum flow in the updated graph
- For each condition c_i, check whether the k flows are distributed across at least two subcontractor nodes. This can be done by examining the flow distribution in the residual graph.