CS 577: Introduction to Algorithms 11/16/2015

Homework 9 Solutions (Review Problems)

Instructor: Dieter van Melkebeek TA: Kevin Kowalski

Problem 1

Part (a)

We want to decide whether we can measure our n conditions with our m balloons (of which balloon i can measure only the set of conditions S_i), measuring each condition at least k times, and with no balloon measuring more than two conditions. We can do this by looking at a maximum flow on a particular network. Note that if there is a solution that measures each condition at least k times, then there must be a solution that measures each condition exactly k times; we restrict ourselves both here and in part (b) to solutions where each condition is measured exactly k times, since it facilitates the construction of the flow network we use to solve these problems.

We construct our network G as a bipartite graph. On the left side of the graph, there is a vertex for each of the n conditions that need to be checked, while on the right, there is a vertex for each balloon. We call the left (condition) vertices $c_1 \cdots c_n$, and the right (balloon) vertices $b_1 \cdots b_m$. We include an edge of capacity 1 from a vertex c_i on the left to a vertex b_j on the right if condition i can be checked by balloon j. Then we add a source and a sink to G; the source s has an edge of capacity k to every vertex on the left (i.e., every condition vertex). The sink t has an edge of capacity 2 from every vertex on the right (i.e., every balloon vertex).

We claim that the value of a maximum flow in G is kn if and only if there is a way to measure each of the n conditions k times, with each balloon measuring at most 2 conditions.

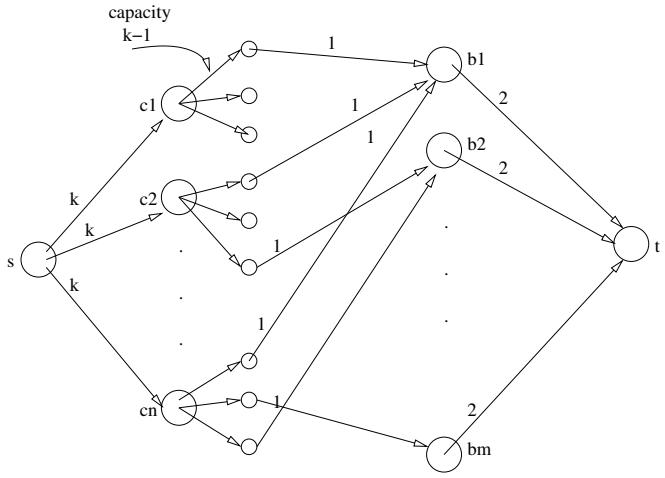
We can see that no flow with value more than kn can exist, since s has n edges leaving it, and each has capacity k. Suppose there exists a working assignment of balloons to conditions, i.e., an assignment that is both satisfactory, in that it measures enough, and valid, in that no balloon measures more than twice. Then we can construct a flow of value kn in G as follows: Push k units of flow from s to each condition vertex, then one unit from each condition vertex to each of the k balloons that are measuring that condition (there must be exactly k such balloons for each condition, since this is a satisfactory assignment); for each balloon, we then push a number of units equal to the number of measurements that balloon is making (which must be 0, 1, or 2, since this is a valid assignment) to the sink.

To see the other direction of the claim, suppose there is a flow in G with value kn. Then there is an integer-valued flow with value kn because all capacities are integer-valued. In this integer-valued flow, we must have each of the n condition vertices pushing flow into exactly k of the edges that connect it to the balloons. These are the balloons we use to check that condition. Every condition is checked by precisely k different balloons, as required. Moreover, since each balloon vertex on the right can push at most 2 units of flow to t, no balloon handles more than 2 conditions. Thus, the assignment works.

Part (b)

We can handle the extra wrinkle – that each balloon is made by one of three subcontractors, and no condition can be measured only by balloons from a single subcontractor – by augmenting the

graph we constructed in part (a) as in the illustration below.



In this illustration, the first subcontractor manufactures balloon b1, which can check c1 and c2.
The second subcontractor manufactures balloon bm, and the third subcontractor manufactures b2

To model this additional constraint, we add three more vertices for each condition (a total of $3n$ additional vertices). These vertices are interposed between the condition vertices and the balloon vertices (i.e., between the left and right sides of G). Each condition vertex has an edge of capacity $k-1$ to each of these three subcontractor vertices, and each subcontractor vertex has an edge of capacity 1 to every balloon it makes.

Thus, a condition vertex cannot send all k units of flow that it receives through the same subcontractor vertex. This prevents any condition from having all k of its measurements done by balloons from the same subcontractor. Once again, there is a way of measuring the conditions subject to all of the constraints if and only if a maximum flow in this network has value kn.

Analysis. In both parts (a) and (b), the networks involved can be constructed in linear time, when we're given the information about which conditions can be measured by which balloons (and, in (b), which balloons are made by which subcontractors). We can then apply any of the polynomial-time algorithms we know for computing the value of a maximum flow and check whether it equals kn.

Problem 2

This problem can be cast as an instance of project selection. There is a project for every friend P_i that is a Packer fan, namely "inviting P_i," with a benefit equal to the value of P_i. There is tool for every friend B_j that is a Bears fan, namely "not inviting B_j," with a cost equal to the value of B_j. The tools needed for P_i are those B_j with which P_i is on bad terms. In that setup, the project selection requirement of buying all tools needed for a selected project exactly corresponds to the given requirement of allowing all possible combinations but the ones where we invite a P_i and a B_j that are on bad terms. Also, the net gain of the project selection equals the total value of the invited friends minus the sum of the values of all friends that are Bear fans. As the latter is a constant, an optimal project selection exactly corresponds to an optimal invitation plan, i.e., we

invite P_i if the corresponding project is selected, and we invite B_j if the corresponding tool is not selected.

Up to constant factors, the running time is the one for project selection, which is the one for maximum flow on a graph with n vertices and m edges, where n denotes the total number of friends, and m the number of pairs (B_i, P_j) that are on bad terms. Using the strongly polynomial-time network flow algorithm mentioned in class, the resulting running time is $O(nm)$.