# ALGORITHMS HOMEWORK 4

EVAN SIMMONS
DEPT. OF MATHEMATICS & COMPUTER SCIENCE
UNIVERSITY OF CALIFORNIA SANTA CRUZ

Homework Option. I have chosen the homework heavy option.

## 1. Problem 10

Suppose you are given a directed graph $G = (V, E)$, with a positive integer capacity $c_e$ on each edge $e$, a source $s \in V$, and a sink $t \in V$. You are also given a maximum $s - t$ flow in $G$, defined by a flow value $f_e$ on each edge $e$. The flow $f$ is acyclic and integer-valued.

Now suppose we pick a specific edge $e \in E$ and reduce its capacity by 1 unit. Show how to find a maximum flow in the resulting capacitated graph in time $O(m+n)$, where $m$ is the number of edges in $G$ and $n$ is the number of nodes.

**Algorithm.** Clearly reducing the capacity of an edge that is not saturated under max flow will not reduce the maximum flow of the network. Therefore if $e*$ is not saturated under maximum flow, then the max flow remains the same. Conversely if the edge $e*$ is saturated under maximum flow, we proceed as follows:

(1) Propagate the reduction in flow.
(2) Run a single iteration of Ford-Fulkerson on the new residual graph.

*Proof.* (Of Correctness) In the first case, clearly the correct max flow is determined. For the second, consider the residual graph produced by step (1). The flow in the new graph will be either the flow of the residual, or one more. It then suffices to show that if one unit of flow can be re-routed, then one iteration of Ford-Fulkerson will reroute it. But a single iteration of Ford-Fulkerson does exactly that, if there is an augmenting path, Ford-Fulkerson will take it, and by integrality it must have flow 1.

**Claim.** *The given algorithm runs in $O(m + n)$ time.*

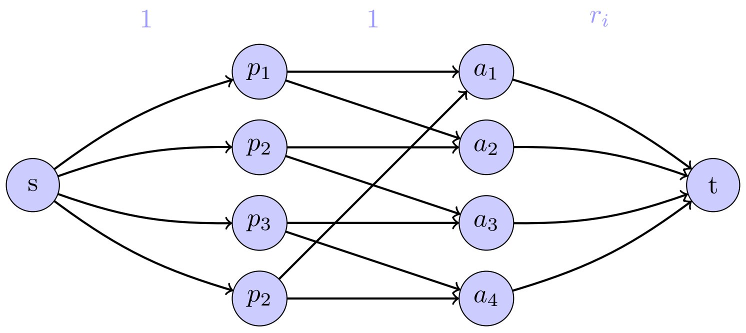
*Proof.* (Of Time) We pay $m$ time to traverse the maximum flow path and identify if $e*$ is saturated. If $e*$ was saturated we pay a second $m$ time to propagate the reduction in flow. On the residual graph produced by the previous step we run one iteration of Ford-Fulkerson which takes $O(m + n)$. Therefore the proposed algorithm runs in $O(m + n)$.

## 2. Problem 16

Suppose that the managers of a popular Web site have identified k distinct demographic groups $G_1, G_2, ..., G_k$. The site has contracts with m different advertisers, to show a certain number of copies of their ads to users of the site.

(1) For a subset $X_i \subseteq G_1, ..., G_k$ of the demographic groups, advertiser $i$ wants its ads shown only to users who belong to at least one of the demographic groups in the set $X_i$.
(2) For a number $r_i$, advertiser $i$ wants its ads shown to at least $r_i$ users each minute.

The problem is: Is there a way to show a single ad to each user so that the sites contracts with each of the m advertisers is satisfied for a given minute? Consider the problem cast as a network flow.



To do this, we create a bipartite graph with a vertex for each person and a vertex for each advertiser. Then from the source there is capacity one for each person, from each person to advertiser there is capacity one (one ad per person), and from each advertiser to the sink there is capacity $r_i$ (the total ads for an advertiser). Certainly if we solve for the maximum flow in the network built in the proposed manner and if maximum flow is $\sum r_i$ then we are able to satisfy all contracts and show at most one ad per person..

## 3. Problem 18

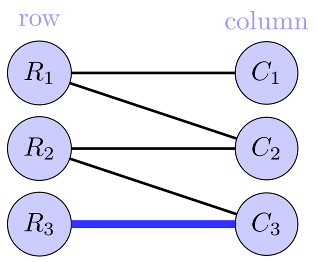
Consider the bipartite matching problem on a graph $G = (V, E)$. Let us call the partitions $X$ and $Y$.

The coverage expansion problem is defined as follows: We are given $G$ and a matching $M$ in $G$. For a given number $k$, we want to decide if there is a matching $M$ in $G$ so that

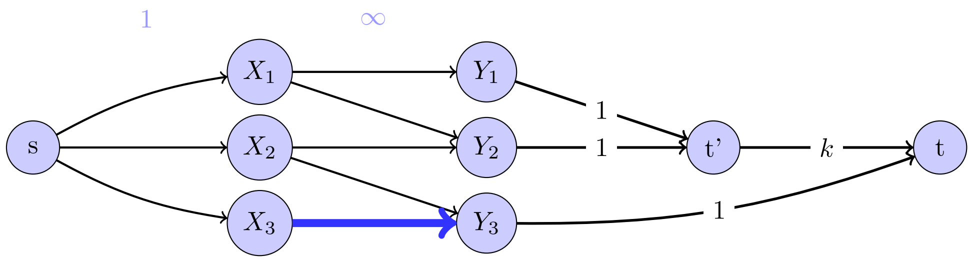
(1) The expansion $M'$ has $k$ more edges than $M$ does.
(2) Every node $y \in Y$ that is covered by $M$ is also covered by $M'$

**a.** Give a polynomial-time algorithm that takes an instance of Coverage Expansion and either returns a solution $M'$ or reports (correctly) that there is no solution.

We will cast the problem as a flow problem. Consider the matching $M = (V_M, E_M)$ below:



To find a $k$ coverage expansion of the above matching we will create the following graph:



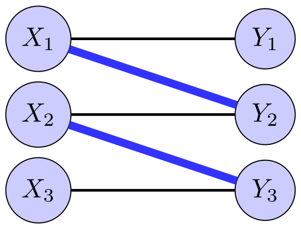
Intuitively this is exactly like using max flow to determine a matching, but with the twist that we want to push $k$ more flow through the graph without losing vertices. To do this (working from right to left) at the sink we constrain the possible flow through the new vertices to be at most $k$ by augmenting the graph with a "pre"-sink for the new flow. The previously matched $Y$ vertices remain incident to the actual sink. The capacitance of the edges which represent the match should have capacity 1. Finally, each $X$ node should be fed with a capacity of one just as in the matching problem.

**Claim.** *If the new network has a max flow of $|E_M| + k$, then the expansion $M'$ is feasible.*

*Proof.* Since $|E_M|$ is the number of edges in the prior matching (and the flow which is direct from $Y$'s to $t$ in the new graph), a flow of $|E_M| + k$ signifies that we have covered since we must push $|E_M|$ flow through the originally matched vertices. Additionally a flow of $|E_M| + k$ signifies that we have made $k$ more matchings.

*Remark.* With Ford-Fulkerson we can find the maximum flow of this network (building it is less) in $O(f \cdot E)$ where $f = |E_M| + k$ the maximum flow, and $E = |X| + |Y| + |X| \cdot |Y|$ is the edges in the network. Further $E_M = \max(|X|, |Y|)$, therefore the algorithm will run in $O(\max(|X|, |Y|) \cdot |X| \cdot |Y|)$.

**b.** The following matching has the desired property:

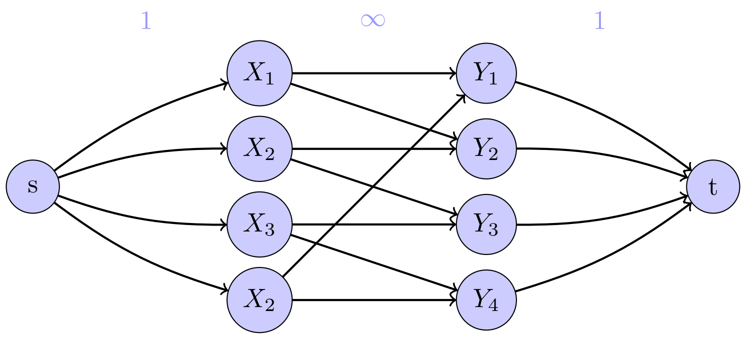


**c.**

**Claim.** *The largest matching is exactly the size of the largest coverage expansion of any matching.*

*Proof.* Let $K_1$ be the size largest subgraph expansion of a given matching $M$. Let $K_2$ be the size of the largest matching possible for the graph $G$ on which $M$ is a matching.

Recall the network used to find a maximal matching of value $K_2$.



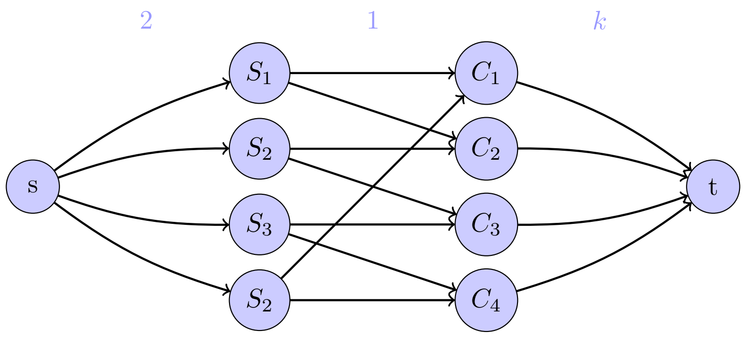
But then it is easy to see that in fact this is exactly the proposed network in part (a) of this problem with the constraint of $k$ removed, and this removal simply consists of setting $k$ to the difference between the size of a given matching and the largest matching.

## 4. Problem 20

Your friends are involved in a large-scale atmospheric science experiment. They 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 they have a set of $m$ balloons that they 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 $i = 1, \ldots, m$, they have a set $S_i$ of conditions that balloon $i$ can measure. Finally, to make the results more reliable, they plan to take each measurement from at least $k$ different balloons. (Note that a single balloon should not measure the same condition twice.) They are having trouble figuring out which conditions to measure on which balloon.

**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.

We consider the problem cast as a network:



**Claim.** *If we cast the problem as a network, then deciding if there is a way to measure each condition $k$ times is equivalent to determining if the maximum flow of the graph is equal to $k \cdot |C|$.*

*Proof.* Let the flow in the graph represent the number of measurements taken. The constraints are as follows:

(1) Each balloon takes at most 2 measurements.
(2) Each balloon takes each measurement at most once.
(3) Each measurement must be taken at least $k$ times.

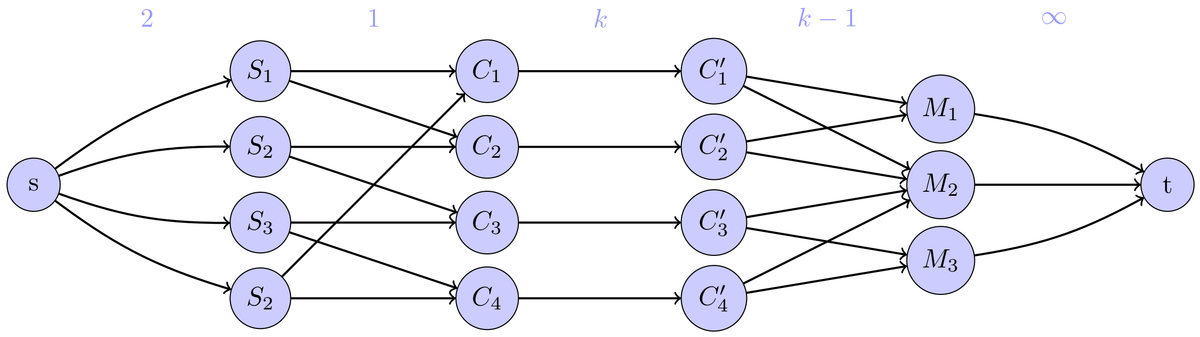
We encode the constraints as the capacities of groups of edges. For the first constraint, since each balloon should have a maximum flow of 2 through it, we say that the single incoming edge will have capacity 2. For the second constraint since each balloon should take each measurement at most once, each edge from a balloon to a measurement will have capacity 1.

Finally consider the last constraint, we want to ensure that each measurement is taken at least $k$ times. Therefore we bound the flow through each measurement node to $k$, and if under maximum flow each of the $k$ weighted edges are saturated then we can take $k$ observations of each measurement. But this is exactly the same as if the maximum flow is equal to $k \cdot |C|$.

*Remark.* By the Ford Fulkerson algorithm we can find the maximum flow of the proposed graph in $O(|S| \cdot E)$ where $E$ is the number of edges, specifically:

$$E = |S| + |C| + \sum_{S_i \in S} |S_i|.$$

**b.** Suppose that each of the balloons is produced by one of three different subcontractors 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. 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.



**Claim.** *If we cast the problem as a network, then deciding if there is a way to measure each condition $k$ times is equivalent to determining if the maximum flow of the graph is equal to $k \cdot |C|$.*

*Proof.* Let the flow in the graph represent the number of measurements taken. The constraints are as follows:

(1) Each balloon takes at most 2 measurements.
(2) Each balloon takes each measurement at most once.
(3) Each measurement must be taken at least $k$ times.
(4) Each measurement can be taken by at most $k-1$ subcontractors M.

As in part a we encode constraints (1)-(3), but this time instead of making the outbound edges from the measurements incident to the sink $t$ we augment the previous representation with a clone of the measurements, and a one to one correspondence of edges from each measurement to its clone. As before theses have capacity $k$. In this way we encode the fact that each measurement may be observed at most $k$ times. Finally to encode the constraint that each measurement must be taken by more than one subcontractor (M) we will consider the edges from the measurement clones, and the subcontractors. Clearly each measurement may be taken at most $k-1$ times by a specific subcontractor, therefore the edges must all have capacity $k-1$. We connect the subcontractor nodes to the sink. Therefore the max flow will only be equal to $k \cdot |C|$ if each measurement can be taken $k$ times under the specified constraints.

*Remark.* By the Ford Fulkerson algorithm we can find the maximum flow of the proposed graph in $O(|S| \cdot E)$ where $E$ is the number of edges, specifically:

$$E = |S| + |C| + |M| + \sum_{S_i \in S} |S_i| + \sum_{M_i \in M} |M_i|$$

where $M_i$ is the set of measurement which can be made by a manufacturer.

## 5. Problem 22

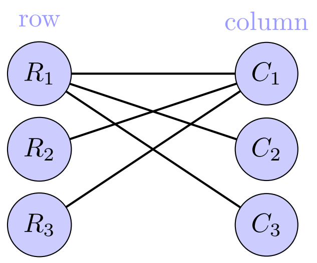
Let $M$ be an $n$ $x$ $n$ matrix with each entry equal to either 0 or 1. Let $m_{ij}$ denote the entry in row $i$ and column $j$. A diagonal entry is one of the form $m_{ii}$ for some $i$. We say that $M$ is rearrangeable if it is possible to swap some of the pairs of rows and some of the pairs of columns (in any sequence) so that, after all the swapping, all the diagonal entries of $M$ are equal to 1.

**a.** Give an example of a matrix $M$ that is not rearrangeable, but for which at least one entry in each row and each column is equal to 1.

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

**b.** Give a polynomial-time algorithm that determines whether a matrix $M$ with $0-1$ entries is rearrangeable.

We will cast the problem as a network flow. Below is an example network:

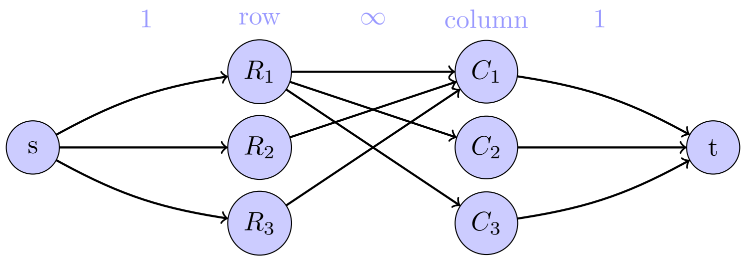


**Claim.** *Suppose we cast the matrix into a bipartite graph similar to the one above where for each* 1 *in the matrix we connect the row that it is in to the column that it is in with an edge. Then if there is a perfect matching in the bipartite graph, then the matrix is rearrangeable.*

*Proof.* Consider the matrix which has only 1's on the diagonal', each $M_{ii} = 1$, ie the lines of the bipartite graph are horizontal. Therefore clearly if we have a perfect matching, every vertex is incident to exactly one edge, then by swapping rows and columns we can achieve a graph which is exactly that of matrix which has only one's on the diagonal.

**Claim.** *The bipartite matching problem is solvable using network flow and the Ford-Fulkerson algorithm.*

*Proof.* Consider the above graph cast into a network.



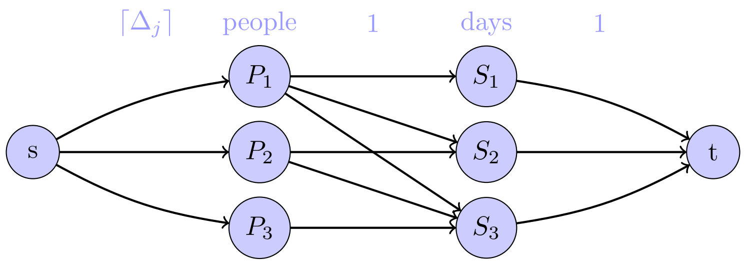
The restrictions as follows: each row can have at most one match, each column can have at most one match. If under the maximum flow the capacity of the given flow is saturated then there exists a perfect matching.

*Remark.* We can solve for the maximum flow using Ford-Fulkerson, which runs in $O(maxflow \ \cdot E) = O(n \cdot E)$ where: $E = n + n^2$. Therefore we can determine if a matrix is rearrangeable in $O(n^3)$ using Ford-Fulkerson.

## 6. Problem 27

Consider a driving schedule for carpool of $P = \{p_1, \ldots, p_k\}$ people over $d$ days. For each day, there is a subset of people who commute, on the $i$th day the subset will be $S_i \subseteq P$.

Consider the problem cast as a network:



Let the flow through the network represent the "drivings". We seek to encode the following constraints:

(1) Each person should drive at most $\lceil \Delta_j \rceil$.
(2) Each day needs at most one driver.

In the network we constrain the flow to each person to their $\lceil \Delta \rceil$, and the flow out of each day to one.

**a.** Prove that for any sequence of sets $S_1, \ldots, S_d$, there exists a fair driving schedule.

*Proof.* Consider the problem cast as a network similarly as above. Certainly the fractional flow problem stated is feasible by definition. (Each matching edge has flow $\frac{1}{S_i}$.) We will argue that in adding two constraints, the max flow is not decreased from $d$, and hence there exists a fair schedule. First consider the cut between people and days, in taking the ceiling of each edge (swapping to one) we merely increase the capacity of the cut, hence will not reduce the max flow of the graph. But now it could be the case that in taking the ceiling of edge in the prior cut, we still may not have integer flow, so we take the ceiling of the input to each person (Again this cannot reduce the flow) the $\Delta_j$'s'. Now every edge in the graph is an integer, so by the integrality theorem the maximum flow is an integer value. Since we have retained a maximum flow of $d$ we are done.

**b.** Give an algorithm to compute a fair driving schedule with running time polynomial in $k$ and $d$.

**Claim.** *A fair driving schedule corresponds to a flow through the network such that each day has flow one.*

*Proof.* By the constraints enumerated above that we have embedded into the network and by integrality, each edge has either flow 1 or 0 if we use Ford-Fulkerson, and denote a person driving on a given day.

*Remark.* We can solve for the maximum flow with Ford-Fulkerson in $O(f \cdot E)$ time. Specifically $f = d$, and

$$E = |P| + |S| + \sum_{S_i \in S} |S_i|.$$

But $|S| = d$, and $\sum_{S_i \in S} |S_i|$ is in $O(d \cdot |P|)$ Therefore the algorithm runs in $O(d^2 + d \cdot |P| + d^2 \cdot |P|) = O(d^2 \cdot |P|)$.

*Remark.* Using a maximum flow through the network, we can extrapolate a fair driving schedule by simply recording the edges between people and days. (These are the days that a given person drives.)